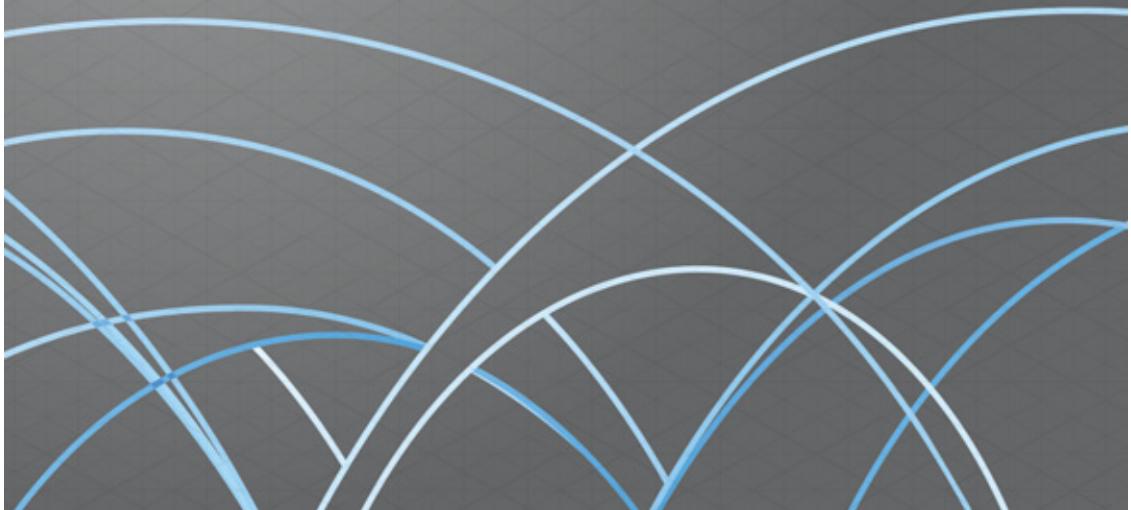




ACTIVATE.
The BIRT Company™



BIRT iHub



Integrating Applications into BIRT iHub

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 - 2014 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

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 Content Services, BIRT Data Analyzer, BIRT for Statements, BIRT iHub, BIRT Metrics Management, 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): zlib. Adobe Systems Incorporated: Flash Player, Source Sans Pro font. Amazon Web Services, Incorporated: Amazon Web Services SDK. Apache Software Foundation (www.apache.org): Ant, Axis, Axis2, Batik, Batik SVG library, Commons Command Line Interface (CLI), Commons Codec, Commons Lang, Commons Math, Crimson, Derby, Hive driver for Hadoop, Kafka, log4j, Pluto, POI ooxml and ooxml-schema, Portlet, Shindig, Struts, Thrift, Tomcat, Velocity, Xalan, Xerces, Xerces2 Java Parser, Xerces-C++ XML Parser, and XML Beans. Daniel Bruce (www.entypo.com): Entypo Pictogram Suite. Castor (www.castor.org), ExoLab Project (www.exolab.org), and Intalio, Inc. (www.intalio.org): Castor. Alessandro Colantonio: CONCISE. Day Management AG: Content Repository for Java. Eclipse Foundation, Inc. (www.eclipse.org): Babel, Data Tools Platform (DTP) ODA, Eclipse SDK, Graphics Editor Framework (GEF), Eclipse Modeling Framework (EMF), Jetty, and Eclipse Web Tools Platform (WTP). Dave Gandy: Font Awesome. Gargoyle Software Inc.: HtmlUnit. GNU Project: GNU Regular Expression. Groovy project (groovy.codehaus.org): Groovy. Guava Libraries: Google Guava. HighSlide: HighCharts. headjs.com: head.js. Hector Project: Cassandra Thrift, Hector. Jason Hsueth and Kenton Varda (code.google.com): Protocol Buffer. H2 Database: H2 database. Groovy project (groovy.codehaus.org): Groovy. IDAutomation.com, Inc.: IDAutomation. IDRolutions Ltd.:JBIG2. InfoSoft Global (P) Ltd.: FusionCharts, FusionMaps, FusionWidgets, PowerCharts. Matt Inger (sourceforge.net): Ant-Contrib. Matt Ingenthron, Eric D. Lambert, and Dustin Sallings (code.google.com): Spymemcached. International Components for Unicode (ICU): ICU library. JCraft, Inc.: JSch. jQuery: jQuery. Yuri Kanivets (code.google.com): Android Wheel gadget. LEAD Technologies, Inc.: LEADTOOLS. The Legion of the Bouncy Castle: Bouncy Castle Crypto APIs. Bruno Lowagie and Paulo Soares: iText. MetaStuff: dom4j. Microsoft Corporation (Microsoft Developer Network): CompoundDocument Library. Mozilla: Mozilla XML Parser. MySQL Americas, Inc.: MySQL Connector. Netscape Communications Corporation, Inc.: Rhino. nullsoft project: Nullsoft Scriptable Install System. OOPS Consultancy: XMLTask. OpenSSL Project: OpenSSL. Oracle Corporation: Berkeley DB, Java Advanced Imaging, JAXB, JDK, Jstl, Oracle JDBC driver. PostgreSQL Global Development Group: pgAdmin, PostgreSQL, PostgreSQL JDBC driver. Progress Software Corporation: DataDirect Connect XE for JDBC Salesforce, DataDirect JDBC, DataDirect ODBC. Quality Open Software: Simple Logging Facade for Java (SLF4J), SLF4J API and NOP. Rogue Wave Software, Inc.: Rogue Wave Library SourcePro Core, tools.h++. Sam Stephenson (prototype.conio.net): prototype.js. Sencha Inc.: Ext JS, Sencha Touch. Shibboleth Consortium: OpenSAML, Shibboleth Identity Provider. Matteo Spinelli: iscroll. StAX Project (stax.codehaus.org): Streaming API for XML (StAX). SWFObject Project (code.google.com): SWFObject. ThimbleWare, Inc.: Memcached. Twittr: Twitter Bootstrap. VMWare: Hyperic SIGAR. Woodstox Project (woodstox.codehaus.org): Woodstox Fast XML processor (wstx-asl). World Wide Web Consortium (W3C) (MIT, ERCIM, Keio): Flute, JTidy, Simple API for CSS. XFree86 Project, Inc.: (www.xfree86.org): xvfb. ZXing Project (code.google.com): ZXing.

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

Document No. 131215-2-430301 August 8, 2014

Contents

About *Integrating Applications into BIRT iHub* ix

Chapter 1

Overview of Actuate APIs 1

Introducing Actuate APIs	2
API libraries overview	2
Information Delivery API (IDAPI)	2
JavaScript API (JSAPI)	2
Online Archive Driver API	2
API plug-ins overview	3
Report Engine API	3
Design Engine API or model API	3
Chart Engine API	3
BIRT scripting	3
EasyScript	4
Dashboard and gadgets	4
BIRT emitters	4
BIRT data connectors	4
Application extensions overview	4
BIRT Viewer Java Extension	5
Data Object API	5
iHub API	5
iPortal Security Extension (IPSE)	5
Report Server Security Extension (RSSE)	5
Usage and error logging extension APIs	6
Overview of other application development tools	6

Part 1

Using Actuate BIRT APIs

Chapter 2

Using Actuate APIs to access iHub environment information 9

Writing event handlers to retrieve iHub environment information	10
Writing a JavaScript event handler	10
Writing a Java event handler	10
About the serverContext object	11
JavaScript event handler example	12
Java event handler example	13

Debugging event handlers that use the iHub API	14
iHub API reference	15

Chapter 3

Using the BIRT data object API 21

About generating data objects from an application	22
Generating data object elements for BIRT report designs	22
Creating data object data sets for BIRT report designs	24
Creating data object data cubes for BIRT report designs	24
Tutorial 1: Creating a data element using the Design Engine API	25

Part 2

Using Actuate JavaScript API in an application

Chapter 4

Creating a custom web page using the Actuate JavaScript API 37

About the Actuate JavaScript API	38
Accessing the Actuate JavaScript API	38
About the DOCTYPE tag	39
About UTF8 character encoding	39
Establishing an HTTP session with an Actuate web application	40
About Actuate JavaScript API security integration	41
Establishing a secure connection to more than one web service	41
Using a login servlet to connect to an Actuate web application	42
Using a custom servlet to connect to an Actuate web application	43
Unloading authentication information from the session	43
Viewing reports	44
Tutorial 2: Implementing the JSAPI in a web page to display the viewer.....	46
Controlling viewer user interface features	47
Accessing report content	48
Accessing HTML5 Chart features	49
Using a filter	49
Using a sorter	49
Using dashboards and gadgets	50
Tutorial 3: Implementing the JSAPI in a web page to display a dashboard	52
Tutorial 4: Implementing the JSAPI to catch exceptions with an error callback function	54
Navigating repository content using ReportExplorer	56
Displaying ReportExplorer	56
Opening files from ReportExplorer	58
Tutorial 5: Displaying repository contents and opening files	61
Using and submitting report parameters	66
Using a parameter component	66
Accessing parameter values from the viewer	68

Tutorial 6: Implementing the JSAPI in a web page to display report parameters	69
Tutorial 7: Changing parameter values and definitions	71
Retrieving report content as data	76
Using a data service component	76
Using a result set component	77
Controlling Interactive Viewer user interface features	78
Disabling UI features in a custom web page	80
Tutorial 8: Control the BIRT Interactive Viewer user interface	82

Chapter 5

Creating dynamic report content using the Actuate JavaScript API 89

About Actuate JavaScript API scripting in a BIRT report design	90
Using the Actuate JavaScript API in an HTML button	91
Tutorial 9: Adding scripted chart controls to a BIRT design	92
Tutorial 10: Using HTML buttons to apply filters to a chart	97
Using the Actuate JavaScript API in chart interactive features	103
Tutorial 11: Adding an interactive chart filter to a BIRT report.....	108
Using the Actuate JavaScript API in chart themes	110
Tutorial 12: Adding scripted HTML5 Chart controls to a BIRT design	111

Chapter 6

Working with Interactive Crosstabs 117

About cross tabs	118
Tutorial 13: Viewing and pivoting a cross tab	119
About cubes	120
Handling Interactive Crosstabs viewer events	122
Working with dimensions, measures, and levels	122
Adding a dimension with levels	123
Removing a dimension	123
Adding and removing measures.....	123
Changing measures and dimensions	124
Working with totals	125
Sorting and filtering cross tab data	126
Drilling down within a cross tab	127
Controlling the Interactive Crosstabs viewer user interface	128

Chapter 7

Actuate JavaScript API classes 131

Actuate JavaScript API overview	132
About the actuate namespace	132
Using the Actuate library	132
Actuate JavaScript API classes quick reference	132
Actuate JavaScript API reference	135

Chapter 8	
BIRT Interactive Crosstabs API classes	389
About the BIRT Interactive Crosstabs JavaScript API	390
Interactive Crosstabs API reference	391
Interactive Crosstabs JavaScript classes quick reference	393

Part 3

Introduction to the Actuate Information Delivery API

Chapter 9	
Understanding the Information Delivery API and schema	491
About the Actuate Information Delivery API	492
About web services and WSDL	493
Understanding the elements of the iHub WSDL schema	493
About the definitions element	493
About data type definitions	495
About message definitions	496
About the portType definition	497
About the binding definition	497
About the service definition	498
Accessing the Actuate schema using a web browser	499

Chapter 10	
Constructing a SOAP message	501
About SOAP messaging	502
Calling an Actuate web service	502
About SOAP message elements	503
Understanding the HTTP header	504
Understanding the SOAP envelope	505
About XML namespace declarations	506
Understanding the SOAP header	507
Understanding the SOAP message body	509
About SOAP Fault messages	511

Part 4

Developing Actuate Information Delivery API applications

Chapter 11	
Developing Actuate Information Delivery API applications using Java	515
About the Apache Axis 2 client	516

Generating the com.actuate.schemas library	516
About third-party code libraries	517
Tutorial 14: Generating com.actuate.schemas library	518
About the Actuate Information Delivery API framework	524
Using a data type from a WSDL document to generate a JavaBean	525
Using metadata to map XML to a Java type	526
Mapping the portType to a Service Definition Interface	527
Using a WSDL binding to generate a Java stub	528
Implementing the Actuate API service	529
Developing Actuate Information Delivery API applications	530
Writing a program that logs in to BIRT iHub System	532
About the auxiliary classes provided by the sample application	533
Logging in to BIRT iHub System	536
Tutorial 15: Catching a SOAP message with Axis TCPMonitor	537
Writing a simple administration application	553
Creating a user	554
About ActuateControl.createUser()	555
About ActuateControl.runAdminOperation()	556
Tutorial 16: Writing an application that creates a user	557
Performing a search operation	566
Using com.actuate.schemas.SelectFiles	566
Using ResultDef	568
Writing a batch or transaction application	570
About batch and transaction operations	570
Implementing a transaction-based application	571
Uploading a file	573
About ways of uploading a file	573
Using com.actuate.schemas.UploadFile	573
How to build an application that uploads a file	574
Tutorial 17: Writing an application that uploads a file	577
Downloading a file	587
Using com.actuate.schemas.DownloadFile	587
How to build an application that downloads a file	588
Executing a report	590
Understanding the structure of an ExecuteReport application	591
Using SelectJavaReportPage	592

Chapter 12	
Developing Actuate Information Delivery API applications	
 using Microsoft .NET	599
About the Microsoft .NET client	600
About the Actuate Information Delivery API framework	603
Using a data type from a WSDL document to generate a C# class	603

Mapping the portType to a web service interface	605
Developing Actuate Information Delivery API applications	606
Writing a program that logs in to BIRT iHub System	606
Writing a simple administration application	610
Performing a search operation	611
Writing a batch or transaction application	614
About batch and transaction operations	615
Implementing a transaction-based application	615
Uploading a file	617
About ways of uploading a file	617
Using UploadFile	617
Building an application that uploads a file	618
Downloading a file	619
Using DownloadFile	619
Building an application that downloads a file	620
Chapter 13	
Actuate Information Delivery API operations	623
IDAPI operations quick reference	623
Chapter 14	
Actuate Information Delivery API data types	695
IDAPI data types quick reference	695
Chapter 15	
Text string limits in Actuate operations	815
Part 5	
Accessing BIRT applications using URIs	
Chapter 16	
Actuate application URIs	821
Working with Actuate application URIs	822
Visualization Platform URLs	822
Using a special character in a URI	823
UTF-8 encoding	824
Actuate application URIs overview	825
Actuate application URIs quick reference	825
Common URI parameters	826
Visualization Platform Struts actions	828
Actuate application URIs reference	834

Chapter 17	
Actuate Report Studio URLs	863
Accessing Report Studio using a URI	864
Using the Report Studio servlet	864
Using the Report Studio URLs	865
Chapter 18	
Actuate BIRT Viewer URIs	867
About the BIRT Viewer servlet	868
Using open source BIRT URIs in Actuate BIRT Viewer	868
Accessing the BIRT Viewer using a URI	868
Part 6	
Using Actuate security	
Chapter 19	
Using Visualization Platform security	873
About Actuate Visualization Platform security	874
Protecting corporate data	874
Protecting corporate data using firewalls	874
Protecting corporate data using proxy servers	875
Understanding the authentication process	875
Creating a custom security adapter	876
Accessing the IPSE Java classes	877
Creating a custom security adapter class	877
Deploying a custom security adapter	878
Understanding the security adapter class	879
Creating an upload security adapter	882
Accessing the necessary Java classes	883
Creating a custom security adapter class	883
Deploying an upload security adapter	884
Understanding the upload security adapter interface	885
Chapter 20	
Using Java Report Server Security Extension	887
About the Java Report Server Security Extension	888
Implementing the Java RSSE interface	888
About installing a Java RSSE application	889
Installing a Java RSSE application	890
Configuring and deploying an LDAP configuration file	891
Installing the page-level security application	898

Migrating a Java RSSE application to a new Actuate release	899
Using page-level security	899
Creating an access control list (ACL)	900
Deploying a report to a volume	901
Report Server Security Extension (RSSE) API operations	902
Report Server Security Extension (RSSE) API data types	909
 Part 7	
Working with usage and error logging	
 Chapter 21	
Using Actuate logging and monitoring APIs	915
About usage logging and error logging extensions	916
Interpreting AC_SERVER_HOME	916
Developing usage and error logging extensions	916
Configuring usage and error logging	917
Customizing the usage logging extension	918
Customizing the error logging extension	919
About the usage log	920
About types of recorded events	920
Understanding a usage log entry	921
About the error log	923
Understanding an error log entry	924
About BIRT iHub error messages	925
About BIRT iHub usage and error log consolidator	926
 Chapter 22	
Actuate logging extension functions	943
About Usage Logging Extension functions	944
About Error Logging Extension functions	945
Index	947

About Integrating Applications into BIRT iHub

Integrating Applications into BIRT iHub provides information about integrating applications using Actuate APIs and URIs into BIRT iHub. The APIs include client-side programming using Actuate JavaScript API and server-side programming using the XML-based Actuate Information Delivery API. This guide includes an introduction to the concepts required to work with the APIs. This guide also provides information about using external security systems to provide user credentials to BIRT iHub.

Integrating Applications into BIRT iHub includes the following parts and chapters:

- *About Integrating Applications into BIRT iHub.* This chapter provides an overview of this guide.
- *Chapter 1. Overview of Actuate APIs.* This chapter lists all the public APIs that Actuate provides. Each API description includes an example of the type of application that uses the API.
- *Part 1. Using Actuate BIRT APIs.* This part describes how to use classes in the com.actuate.birt.* public packages to integrate BIRT applications and data objects into BIRT iHub.
- *Chapter 2. Using Actuate APIs to access iHub environment information.* This chapter describes how to write event handlers in a report to retrieve BIRT iHub environment information.
- *Chapter 3. Using the BIRT data object API.* This chapter describes how to work with BIRT data objects and report designs programmatically.
- *Part 2. Using Actuate JavaScript API in an application.* This part describes how to design custom reporting web applications with the Actuate JavaScript API.
- *Chapter 4. Creating a custom web page using the Actuate JavaScript API.* This chapter describes the Actuate JavaScript API requirements and common implementations.

- *Chapter 5. Creating dynamic report content using the Actuate JavaScript API.* This chapter describes using Actuate JavaScript API code in a BIRT report.
- *Chapter 6. Working with Interactive Crosstabs.* This chapter describes how to create, access, view, and modify Interactive Crosstabs.
- *Chapter 7. Actuate JavaScript API classes.* This chapter lists all the standard Actuate JavaScript API classes and their methods.
- *Chapter 8. BIRT Interactive Crosstabs API classes.* This chapter lists all the cross tab classes and their methods.
- *Part 3. Introduction to the Actuate Information Delivery API.* This part describes the structure and implementation of the Actuate Information Delivery API and how to pass messages from an application to BIRT iHub using this API.
- *Chapter 9. Understanding the Information Delivery API and schema.* This chapter introduces the features of the API and describes the Actuate Web Services Description Language (WSDL) schema.
- *Chapter 10. Constructing a SOAP message.* This chapter discusses the elements of Actuate Simple Object Access Protocol (SOAP) messages.
- *Part 4. Developing Actuate Information Delivery API applications.* This part describes how to create Java and .NET applications that use the Actuate Information Delivery API by grouping Actuate Information Delivery API operations frequently used together. This part also provides complete descriptions of the Actuate Information Delivery API data types and operations.
- *Chapter 11. Developing Actuate Information Delivery API applications using Java.* This chapter describes how to use the Actuate Information Delivery API framework to create client applications that request Actuate iHub to perform administration, search, batch, and transaction operations, upload or download files, and schedule a custom event, using the Apache Axis development environment.
- *Chapter 12. Developing Actuate Information Delivery API applications using Microsoft .NET.* This chapter describes how to use the Actuate Information Delivery API framework to create client applications that request Actuate iHub to perform administration, search, batch, and iHub transaction operations, and upload or download files, using the Microsoft .NET development environment.
- *Chapter 13. Actuate Information Delivery API operations.* This chapter summarizes the web services available through the API.
- *Chapter 14. Actuate Information Delivery API data types.* This chapter contains an alphabetical listing of the Actuate Information Delivery API data types.
- *Chapter 15. Text string limits in Actuate operations.* This chapter lists the maximum field lengths for text elements in BIRT iHub Visualization Platform

and System Console for elements the Actuate Information Delivery API creates.

- *Part 5. Accessing BIRT applications using URIs.* This part describes how to access the Visualization Platform, Interactive Viewer, and Report Studio applications using URIs.
- *Chapter 16. Actuate application URIs.* This chapter describes the JavaServer Pages (JSPs) for Visualization Platform and the URIs and parameters to access those pages.
- *Chapter 17. Actuate Report Studio URIs.* This chapter describes the servlet for Report Studio and the URI and parameters to access the servlet.
- *Chapter 18. Actuate BIRT Viewer URIs.* This chapter describes the servlet for Actuate BIRT Viewer and the URI and parameters to access the servlet.
- *Part 6. Using Actuate security.* This part describes two ways to access BIRT iHub using external security to provide user login credentials.
- *Chapter 19. Using Visualization Platform security.* This chapter introduces the iPortal Security Extension (IPSE) for accessing the Visualization Platform and explains how to use it.
- *Chapter 20. Using Java Report Server Security Extension.* This chapter describes how to create and install an Actuate iHub Java Report Server Security Extension (RSSE) application as a web service. Using the Java RSSE framework, a developer can create an application that provides external authentication, external registration, and page-level security.
- *Part 7. Working with usage and error logging.* This part describes how to work with the usage and error logs and the usage and error log extensions.
- *Chapter 21. Using Actuate logging and monitoring APIs.* This chapter describes how to enable and configure usage and error logging, how to customize the usage and error logging extensions, how to interpret the log entries, and how to use the usage and error log consolidator.
- *Chapter 22. Actuate logging extension functions.* This chapter provides an alphabetical listing of the functions and operations of the Error Logging and Usage Logging APIs, including a general description, syntax or schema, and a description of each parameter or element.

1

Overview of Actuate APIs

This chapter contains the information about Introducing Actuate APIs.

Introducing Actuate APIs

Actuate provides software development tools as a collection of APIs that support a developer designing new Actuate applications or extending or customizing existing applications. The APIs are provided as libraries, plug-ins, or application extensions.

Each API can be used independently unless there is an explicit prerequisite. For example, the iHub API, an Actuate application extension, is an extension of the BIRT Event Model, an Eclipse BIRT plug-in, and therefore cannot be implemented without the Event Model included in the server application libraries.

API libraries overview

Actuate APIs libraries are capable of supporting independent applications as well as extending functionality in applications that provide API integration points. Actuate provides:

Information Delivery API (IDAPI)

Description	IDAPI provides libraries for server-side data integration and manipulation using Simple Object Access Protocol (SOAP) messaging. IDAPI messages are defined by a Web Service Definition Language (WSDL) schema file located at <a href="http://<server name>:8000/wsdl/v11/axis/all">http://<server name>:8000/wsdl/v11/axis/all .
Purpose	IDAPI supports the Report Server Security Extension (RSSE). Use IDAPI to perform tasks on a volume, such as file administration, user administration, and running jobs. For examples of using IDAPI, see Chapter 11, “Developing Actuate Information Delivery API applications using Java” and Chapter 12, “Developing Actuate Information Delivery API applications using Microsoft .NET.”

JavaScript API (JSAPI)

Description	JSAPI provides libraries for web and client-side visualizations using the JavaScript programming language. JSAPI is available to iHub web clients using the iportal context in the iportal\jsapi folder.
Purpose	Use JSAPI in a web page to embed reports, Reportlets, and gadgets in a dashboard and to modify the behavior of the Actuate Viewer. Use JSAPI in a report design to enhance interactivity.

Online Archive Driver API

Description	Online Archive Driver API provides libraries that are a SOAP-based interface between BIRT iHub and external archive software.
Purpose	Use Online Archive Driver API for automating the aging and archiving processes for items in a volume

API plug-ins overview

Eclipse BIRT plug-ins are function-specific libraries for enhancing BIRT applications with additional features such as security, database integration, encryption, and data objects using Eclipse. Most Eclipse BIRT plug-ins are built using the Java programming language, but may support HTML, XML, JavaScript, SQL, or custom command sets.

Actuate develops and supports many Eclipse BIRT plug-ins, both open source and commercial. The APIs are available to BIRT Designer Professional as plug-ins in the `eclipse\plugins` folder and to iHub as JAR files in the BIRT iHub installation in the `$ACTUATE_HOME\modules\BIRTiHub\iHub\Jar\BIRT\lib` folder. BIRT Designer Professional also provides the JAR files in the `iportal\WEB-INF\lib` folder in the `com.actuate.birt.report.viewer` plug-in.

Report Engine API

- Description** A set of open source Java packages that provides access to all the tasks that create a report from a report design or a report document. The `org.eclipse.birt.report.engine.api.*` packages in the `engineapi.jar` file contains this API.
- Purpose** Use the Report Engine API to set parameter values, run a report design, and render a report document to one or more output formats.

Design Engine API or model API

- Description** A set of open source Java packages that provides access to the content and structure of a report design, a template, or a library. The `org.eclipse.birt.report.model.api.*` packages in the `modelapi.jar` file contain the components of this API.
- Purpose** Use the Design Engine API to modify the structure of an existing report design or to create a completely new report design.

Chart Engine API

- Description** A set of open source Java packages used to create and modify chart elements. This API supports charting in a stand-alone application or, in conjunction with the Report Engine and Design Engine APIs, in a report design. The `org.eclipse.birt.chart.*` packages in the `chartengineapi.jar` file contain the components of this API.
- Purpose** Use the Chart Engine API to modify an existing chart element or to create a completely new chart element. Extend the chart element to provide new chart types.

BIRT scripting

- Description** A set of Java packages that enables creation of custom code to control various aspects of report creation and interactivity. BIRT scripting supports JavaScript and Java functions. Script handling follows the BIRT Event model. This model

provides support for event registration and triggers initiated from BIRT content or the server environment. The org.eclipse.birt.report.engine.api.script.* packages in the scriptapi.jar file contain this API and model.

- Purpose** Use BIRT scripting in event handlers and in expressions.

EasyScript

Description An expression syntax similar to the syntax used in Excel formulas. EasyScript supports formula and data manipulations using its own markup. The com.actuate.script.* plug-ins provide this scripting syntax and user interface.

- Purpose** Use EasyScript to write expressions in BIRT Designer Professional, Interactive Viewer, and Report Studio.

Dashboard and gadgets

Description A plug-in that supports arranging data into interactive visualizations that can be deployed individually. Dashboards and gadgets use JavaScript to control their functionality. Supported gadgets include Google gadgets that display dynamic web content using HTML and JavaScript. The com.actuate.birt.report.model.api.dashboard package in the modelapi.jar file contains this API.

- Purpose** Use the dashboard and gadgets API to enhance dashboard and gadget functionality.

BIRT emitters

Description Plug-ins that support generating content in different formats, such as PDF, HTML, CSV, and so on. The org.eclipse.birt.report.engine.emitter.* and com.actuate.birt.report.engine.emitter.* plug-ins provide the content emitters.

- Purpose** Use the emitter APIs to customize the built-in emitters to change their behavior or to create a new emitter to generate output in an additional format.

BIRT data connectors

Description Connectors that implement drivers to establish connections with specific database or other data source types. For example, Actuate provides connectors for Oracle, MySQL, and flat file data source types. You can use database connectors in IDAPI applications, Report Engine applications, Data Objects, or as the basis for custom database connectors. The org.eclipse.birt.report.data.oda.* and com.actuate.data.oda.* plug-ins provide the data connectors.

- Purpose** Use the Open Data Access (ODA) standard to create custom data connectors to access data from additional data sources.

Application extensions overview

Actuate application extensions provide additional functionality to commercial Actuate products, such as BIRT Viewer or iHub.

BIRT Viewer Java Extension

Description A commercial implementation of the BIRT Design Engine API that expands the capabilities of the Actuate BIRT Viewer. This API is provided by the com.actuate.birtviewer.extension package in com.actuate.iportal.jar, which is located in the \$ACTUATE_HOME\modules\BIRTiHub\iHub\web\iportal\WEB-INF\lib folder.

Use the BIRT Viewer Java extension API to implement event handlers that fire when a user views or interacts with the report.

Data Object API

Description A Java API extension used to create and alter data objects. The com.actuate.birt.report.model.api package in the modelapi.jar file contains this API extension.

Purpose Use the Data Object API to create, alter, and refresh data objects, based on custom conditions or events.

iHub API

Description An Event Model extension that provides a report with access to the iHub server environment for specific logistical operations. This extension is provided in \$ACTUATE_HOME\modules\BIRTiHub\iHub\reportengines\lib\jrem.jar.

Purpose Use the iHub API in an event handler in a report design to customize report content based on conditions such as the user credentials, volume, or the web browser used to view the report.

iPortal Security Extension (IPSE)

Description An API extension that supports custom security adapters to implement single-sign on and download security features. The iPortalSecurityAdapter class is provided in com.actuate.iportal.jar, which is located in the \$ACTUATE_HOME\modules\BIRTiHub\iHub\web\iportal\WEB-INF\lib folder.

Purpose Use IPSE to customize and control the user login and authentication process.

Report Server Security Extension (RSSE)

Description A SOAP-based security module that creates individual security protocols for consumable content, for example by using an existing Lightweight Directory Access Protocol (LDAP) implementation to control access to a volume. RSSE is based on the IDAPI and can implement a highly granular security scheme for all BIRT content. The BIRT iHub Integration Technology package provides an example of using RSSE.

Purpose Use RSSE to implement external authentication or external registration, or to customize page-level security.

Usage and error logging extension APIs

Description	API extensions that support interpreting the BIRT iHub usage and error logs. The BIRT iHub Integration Technology package provides examples of using these APIs in the User Activity Logging Extension and Error Logging Extension folders.
Purpose	Use the usage and error logging extension APIs to understand how BIRT iHub uses system resources and to troubleshoot problems.

Overview of other application development tools

Actuate provides an extension to ANSI Structured Query Language (SQL) syntax, called Actuate SQL. This syntax is supported by Actuate information objects. Information Object Query Builder provides a graphical interface to build queries using Actuate SQL. For a complete description of Actuate SQL syntax, see *Designing Information Objects*.

Actuate reports and dashboards support the use of the Highcharts JavaScript API to customize HTML 5 charts, such as providing interactivity or changing the appearance of an axis, a series, or a data point. For more information about this API, see the Highcharts reference documentation at <http://api.highcharts.com/highcharts>.

Part One

Using Actuate BIRT APIs

2

Using Actuate APIs to access iHub environment information

This chapter contains the following topics:

- Writing event handlers to retrieve iHub environment information
- Debugging event handlers that use the iHub API
- iHub API reference

Writing event handlers to retrieve iHub environment information

Report developers distribute reports to users by publishing them to BIRT iHub. Sometimes a report requires information about the iHub environment to implement application or business logic based on, for example, the security credentials of the user running the report, the browser in which the report is viewed, the server volume on which the report is run, and so on. BIRT provides an API, referred to in this chapter as the iHub API, that enables access to this type of information.

To use the iHub API in a report, you write event handler scripts in either Java or JavaScript. BIRT event handlers are associated with all the elements that make up a report, such as data sources, data sets, tables, charts, and labels. When a report is run, BIRT fires events and executes event handlers in a specific sequence to generate and render the report.

Writing event handlers in a report requires knowledge of the BIRT event model. For information about the event model and details about writing event handlers in Java and JavaScript, see *Integrating and Extending BIRT*. This chapter describes the additional requirements for accessing and debugging the iHub API in an event handler.

Writing a JavaScript event handler

You write a JavaScript event handler that uses the iHub API the same way you write other event handlers. In Actuate BIRT Designer Professional, you select an element, such as the report design or a table, then use the script editor to select an event, such as beforeFactory or onCreate, for which to write an event handler.

Figure 2-1 shows the script editor displaying event-handling code written for the report design's beforeFactory event.

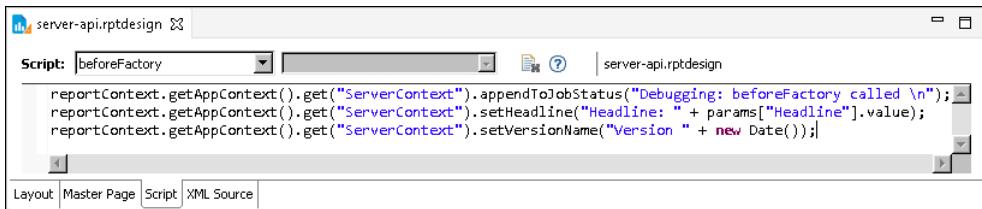


Figure 2-1 Event-handling code in the script editor

Writing a Java event handler

Writing a Java event handler that uses the iHub API is similar to writing other types of event handlers. You create a Java event handler class, make the class

available to BIRT, and associate the class with a report element. The difference is the additional JAR files required to access the iHub API.

You must add the following JAR files in the build path and classpath when configuring the Java event handler project:

- \$ACTUATE_HOME\modules\BIRTiHub\iHub\Jar\BIRT\lib\scriptapi.jar
This JAR file provides the event handler classes and access to the reportContext object. If you use the ULocale methods, com.ibm.icu_version.jar is also required. \$ACTUATE_HOME is the location where iHub is installed.
- \$ACTUATE_HOME\modules\BIRTiHub\iHub\reportengines\lib\jrem.jar
This JAR file contains the definitions of the classes and methods in the iHub API.

Figure 2-2 shows the build path of a Java project that uses the iHub API. In this example, BIRT Designer Professional is installed on the same machine where iHub is installed. If Actuate BIRT Designer Professional is installed on a different machine, you must copy the JAR files from the iHub machine to your workspace.

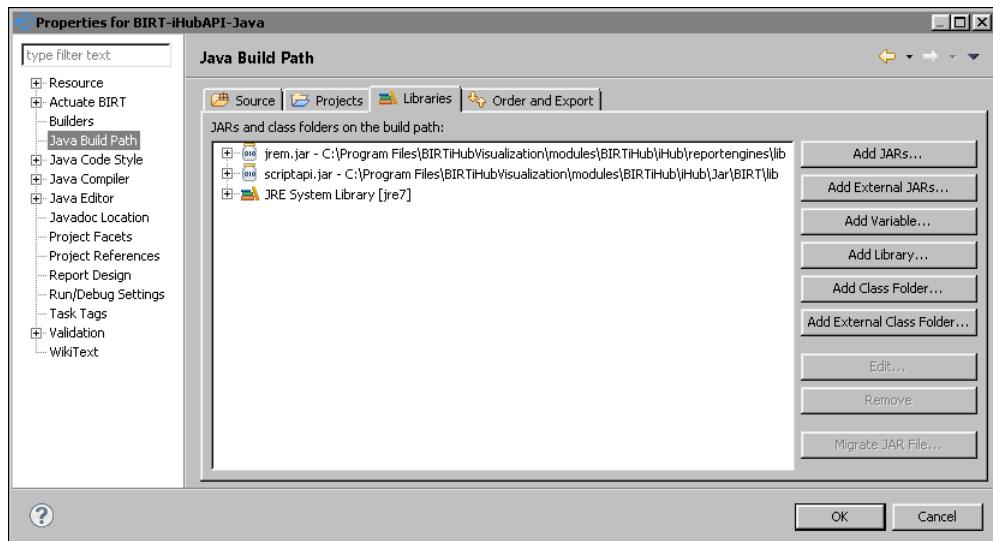


Figure 2-2 Build path of a Java project that uses the iHub API

About the serverContext object

The BIRT engine uses an object called serverContext to store information about the iHub environment. The serverContext methods and properties are defined in the IServerContext interface. The container for the serverContext object is the application context object appContext. The appContext object stores objects and values that are used in all phases of report generation and presentation.

The appContext object, in turn, is a property of the reportContext object. This object stores information associated with the instance of the report that is being generated or viewed. For example, the reportContext object stores information about report parameters, global variables, report output format, locale, the request that runs the report, and the application context. The report context class defines methods for setting and retrieving these properties. Every event handler in a BIRT report has access to the reportContext object. In Java, the report context object is an argument to all event-handler methods.

To call a method to retrieve iHub environment information, the code must reflect the relationships between the serverContext, appContext, and reportContext objects.

The following JavaScript code snippet shows how to call the getVolumeName() method to retrieve the name of the iHub volume in which a report runs:

```
reportContext.getAppContext().get("ServerContext").getVolumeName()
```

The following example shows the equivalent code snippet in Java:

```
IApplicationContext scontext;
scontext = (IApplicationContext)
    reportContext.getAppContext().get("ServerContext");
scontext.getVolumeName();
```

JavaScript event handler example

The code example in Listing 2-1 uses the getUserRoles() method to retrieve the user's roles and displays the contents of a report element if the user role is Manager. This code can be used, for example, in the onPrepare event of a table element to hide or display the table depending on the user role. The code example also uses the appendToJobStatus() method to write messages about the user's roles to the server job status.

Listing 2-1 JavaScript event handler

```
userRoles = reportContext.getAppContext().get("ServerContext")
    .getUserRoles();

reportContext.getAppContext().get("ServerContext")
    .appendToJobStatus("The user roles are:" + userRoles + "\n");

if (userRoles != null)
{
    for (i = 0; i < userRoles.size(); i++)
    {
        if (userRoles.get(i) == "Manager")
        {
            reportContext.setGlobalVariable("HideDetails", "false");
            reportContext.getAppContext().get("ServerContext")
                .appendToJobStatus("The user has a Manager role\n");
```

```
        }  
    }  
}
```

Java event handler example

Like the JavaScript event handler in the previous section, the Java code example in Listing 2-2 uses the `getUserRoles()` method to retrieve the user's roles and displays the contents of a table if the user role is Manager. The `TableEH` class extends the `TableEventAdapter` class and implements the event-handler script in the `onPrepare` event method.

Listing 2-2 Java event handler class

```
package server.api.eh;

import java.util.List;

import org.eclipse.birt.report.engine.api.script.IReportContext;
import org.eclipse.birt.report.engine.api.script.element.ITable;
import org.eclipse.birt.report.engine.api.script.eventadapter
    .TableEventAdapter;

import com.actuate.reportapi.engine.IServerContext;

public class TableEH extends TableEventAdapter {

    public void onPrepare(ITable tbl, IReportContext reportContext)
    {
        IServerContext scontext;
        scontext = (IServerContext)
            reportContext.getAppContext().get("ServerContext");
        List<String> userRoles = scontext.getUserRoles();
        scontext.appendToJobStatus("The user roles are:" + userRoles
            +"\n");
        for (int i = 0; i < userRoles.size(); i++)
        {
            if ( userRoles.get(i).contentEquals("Manager") )
            {
                reportContext.setGlobalVariable("HideDetails", "false");
                scontext.appendToJobStatus("The user has a Manager role
                    \n");
                break;
            }
        }
    }
}
```

Debugging event handlers that use the iHub API

A report that uses the iHub API returns the expected results only when it is run on iHub. When the report is run in Actuate BIRT Designer Professional, the report cannot access the iHub to retrieve the server information, and the report typically returns null values. Therefore, you cannot debug the iHub API calls in the same way you debug other event handlers in Actuate BIRT Designer Professional.

To debug iHub API calls, use the `appendToJobStatus()` method to write a debugging message for each event handler. For example, in a JavaScript event handler for the `beforeFactory` event, add the following line of debugging code:

```
reportContext.getAppContext().get("ServerContext")
    .appendToJobStatus("Debugging: beforeFactory called.\n");
```

In a Java event handler, write:

```
IServerContext scontext;
scontext = (IServerContext)
    reportContext.getAppContext().get("ServerContext");
scontext.appendToJobStatus("Debugging: beforeFactory called.\n");
```

The `appendToJobStatus()` method writes a specified string message in the status section of a job-completion notice. After running the report on iHub, you can view these messages in iHub client.

In iHub client, choose My Jobs->Completed, then choose the job's details. The job's Status page displays the debug messages in the Status section, as shown in Figure 2-3.

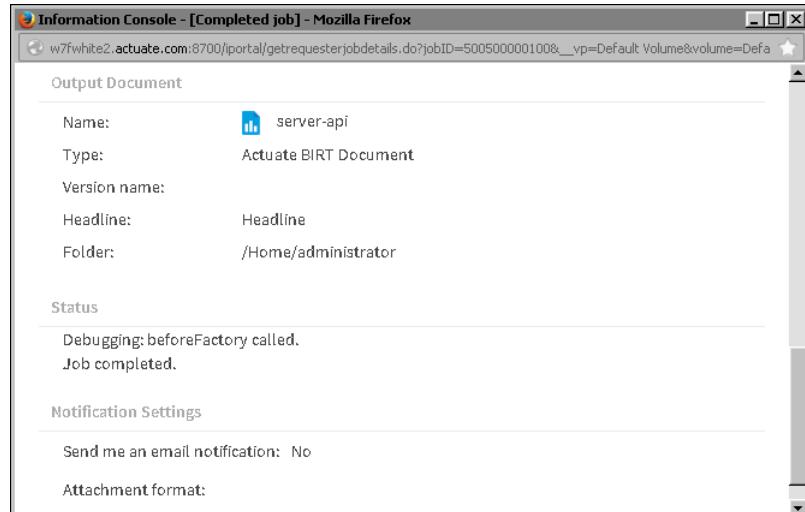


Figure 2-3 Debug message in the job status page in iHub client

iHub API reference

This section lists all the methods in the iHub API in alphabetical order. Each method entry includes a general description of the method, the JavaScript and Java syntaxes, the result the method returns, and examples.

appendToJobStatus()

Appends a specified string to the status of the current job. iHub writes status messages for each report-generation job.

JavaScript syntax `appendToJobStatus(statusString)`

Java syntax `public void appendToJobStatus(String statusString)`

Argument **statusString**
The string to add to the job status.

Usage Provide information for debugging purposes. For example, to verify that an event handler is executed, write a message indicating that the event method is called.

JavaScript example `reportContext.getAppContext().get("ServerContext").appendToJobStatus("This message appears when beforeFactory is called.\n");`

Java example `IApplicationContext scontext;
scontext = (IApplicationContext)
 reportContext.getAppContext().get("ServerContext");
scontext.appendToJobStatus("This message appears when
beforeFactory is called.\n");`

getAuthenticationId()

Retrieves the current user's authentication ID.

JavaScript syntax `getAuthenticationId()`

Java syntax `public String getAuthenticationId()`

Usage Use in cases when the report application needs to pass the ID to another application, such as IDAPI calls to iHub.

Returns An authentication ID in String format.

getServerWorkingDirectory()

JavaScript example reportContext.getApplicationContext().get("ServerContext") .getAuthenticationId();

Java example IServerContext scontext;
scontext = (IServerContext) reportContext.getApplicationContext().get("ServerContext");
scontext.getAuthenticationId();

getServerWorkingDirectory()

Retrieves the path to the folder in the file system where temporary files are stored.

JavaScript syntax getServerWorkingDirectory()

Java syntax public String getServerWorkingDirectory()

Usage Use to read or write information from and to the file system.

Returns The full path to the iHub working directory.

JavaScript example reportContext.getApplicationContext().get("ServerContext") .getServerWorkingDirectory();

Java example IServerContext scontext;
scontext = (IServerContext) reportContext.getApplicationContext().get("ServerContext");
scontext.getServerWorkingDirectory();

getUserName()

Retrieves the current user name.

JavaScript syntax getUserName()

Java syntax public String getUserName()

Usage Use in cases when an application requires different code for different users.

Returns The current user name.

JavaScript example reportContext.getApplicationContext().getUserName()

Java example IServerContext scontext;
scontext = (IServerContext) reportContext.getApplicationContext().get("ServerContext");
String currentUser = scontext.getUserName();

getUserAgentString()

Identifies the browser used to view a report.

JavaScript syntax `getUserAgentString()`

Java syntax `public String getUserAgentString()`

Usage Use in cases when an application requires different code for different browsers. The browser information is available only when the report is rendered, so use `getUserAgentString()` in a report element's `onRender` event.

Returns The browser type in String format. For Internet Explorer, for example, `getUserAgentString()` might return a string, such as:

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.1;
MS-RTC LM 8)
```

JavaScript example `reportContext.getAppContext().get("ServerContext")
.getUserAgentString();`

Java example `IApplicationContext scontext;
scontext = (IApplicationContext)
reportContext.getAppContext().get("ServerContext");
scontext.getUserAgentString();`

getUserRoles()

Retrieves the roles assigned to the current user.

JavaScript syntax `getUserRoles()`

Java syntax `public List<String> getUserRoles()`

Usage Use in cases when an application requires different code for different iHub security roles.

Returns The current user's security roles.

JavaScript example `reportContext.getAppContext().get("ServerContext").getUserRoles();`

Java example `IApplicationContext scontext;
scontext = (IApplicationContext)
reportContext.getAppContext().get("ServerContext");
List<String> userRoles = scontext.getUserRoles();`

```
getVolumeName( )
```

getVolumeName()

Retrieves the name of the iHub volume on which the report runs.

JavaScript syntax `getVolumeName()`

Java syntax `public String getVolumeName()`

Usage Use in cases when an application running in a multi-volume environment requires volume information.

Returns The name of the iHub volume running a report.

JavaScript example `reportContext.getAppContext().get("ServerContext").getVolumeName();`

Java example `IApplicationContext scontext;
scontext = (IApplicationContext)
 reportContext.getAppContext().get("ServerContext");
scontext.getVolumeName();`

setHeadline()

Sets the headline of a generated report. A headline appears in a job completion notice that iHub writes to a channel.

JavaScript syntax `setHeadline(headline)`

Java syntax `public void setHeadline(String headline)`

Argument **headline**

A string that represents the headline of a completed job.

Usage Use to specify a headline based on the contents of a report, or on the value of a report parameter.

JavaScript example `reportContext.getAppContext().get("ServerContext").setHeadline("Sales Report for " + params["Region"].value);`

Java example `String region = (String)reportContext.getParameterValue("Region");
IApplicationContext scontext;
scontext = (IApplicationContext)
 reportContext.getAppContext().get("ServerContext");
scontext.setHeadline("Sales Report for " + region);`

setVersionName()

Sets the version name of a generated report.

JavaScript syntax setVersionName(versionName)

Java syntax public void setVersionName(String versionName)

Argument **versionName**

A string that represents the report's version name.

Usage Use to specify a version name that includes dynamic data, such as the contents of a report, the value of a report parameter, or the report-generation date.

JavaScript example reportContext.getAppContext().get("ServerContext")
 .setVersionName("Version " + new Date());

Java example IServerContext scontext;
scontext = (IServerContext)
 reportContext.getAppContext().get("ServerContext");
scontext.setVersionName("Version " + new Date());

```
setVersionName( )
```

3

Using the BIRT data object API

This chapter contains the following topics:

- About generating data objects from an application
- Generating data object elements for BIRT report designs
- Creating a data element using the Design Engine API

About generating data objects from an application

BIRT iHub provides a Design Engine API extension to create and alter data objects programmatically with Java, allowing programs to refresh their data from the database as needed. This mechanism can retrieve data based on user controls at run time, such as the availability or status of certain goods and services, or generate content dynamically according to criterion outside of a usual query, such as a network outage or an escrow account closing. In this way, an application can respond to important events and incorporate any outside factors, with or without the direct involvement of any users.

This Actuate extension provides Java classes to automate data object generation, retrieving important information required regularly for any application. The classes that support BIRT data objects, DataMartCubeHandle, DataMartDataSetHandle, and DataMartDataSourceHandle, are contained in the com.actuate.birt.report.model.api package.

Like the BIRT data objects implemented in the BIRT data explorer, BIRT data objects generated by the Design Engine API generate data sources and data sets from a .datadesign or .data file. Using the extension requires programming in Java. Knowledge of XML is also helpful.

Handling data objects for BIRT reports requires knowledge of programming using the BIRT reporting API and the report object model. For information about the BIRT reporting API and the report object model, see *Integrating and Extending BIRT*. This chapter describes the additional requirements for generating data objects for reports.

Generating data object elements for BIRT report designs

To generate data object data sources, data sets, and cubes for a BIRT report design, first configure BIRT_HOME to access the Actuate commercial model API Java archive (JAR) files from Actuate iHub. To accomplish this task, generate a DesignConfig object with a custom BIRT_HOME path, as shown in the following code:

```
// Create an DesignConfig object.  
DesignConfig config = new DesignConfig();  
// Set up the path to your BIRT Home Directory.  
config.setBIRTHome("C:/Program Files/BIRTiHubVisualization  
/modules/BIRTiHub/iHub/Jar/BIRT/platform");
```

Use the path to the iHub installation specific to your system.

Using this design configuration object, create and configure a Design Engine object, open a new session, and generate or open a report design object, as shown in the following code:

```
// Create the engine.  
DesignEngine engine = new DesignEngine( config );  
SessionHandle sessionHandle = engine.newSessionHandle(  
    ULocale.ENGLISH );  
ReportDesignHandle designHandle = sessionHandle.createDesign( );
```

These objects are contained in the model API package
`org.eclipse.birt.report.model.api`.

The ElementFactory class supports access to all the elements in a report. The following code generates an Element Factory object:

```
ElementFactory factory = designHandle.getElementFactory( );
```

To generate data sources, data sets, and cubes, use the datamart methods of an ElementFactory object: `newDataMartCube()` for a new cube, `newDataMartDataSet()` for a data set, and `newDataMartSource()` for a new data source. For example, to instantiate a new data source, use the following code:

```
DataMartDataSourceHandle dataSource =  
factory newDataMartDataSource( "Data Object Data Source" );
```

Associate a handle for a data object data source with an actual data source from the contents of a data or data design file. For example, to associate a data source handle with a data source from `test.datadesign`, use the following code:

```
dataSource.setDataMartURL( "test" );  
dataSource.setAccessType(  
    DesignChoiceConstants.ACCESS_TYPE_TRANSIENT );
```

Finally, add the data element to the report design, as shown in the following code:

```
designHandle.getDataSources( ).add( dataSource );
```

To complete the data source assignment, output the report design into a file and close the design handle object, using code similar to the following:

```
FileOutputStream fos = new FileOutputStream( "output.rptdesign" );  
designHandle.serialize( fos );  
// Close the document.  
fos.close( );  
designHandle.close( );
```

The resulting output file, `output.rptdesign`, contains the new data source, retrieved from `test.datadesign`. This data source appears in Data Sources in Data Explorer and establishes a link to the `.datadesign` file, `test.datadesign`. The XML source for `output.rptdesign` includes markup similar to the following lines:

```

<datamart-node location="file:/MyProject/test.datadesign">
...
<data-sources>
  <data-mart-data-source name="Data Object Data Source" id="7">
    <property name="datamartURL">test</property>
    <property name="accessType">transient</property>
  </data-mart-data-source>
</data-sources>

```

When exporting this report design to a volume, also export test.datadesign to maintain the reference to the data source.

Creating data object data sets for BIRT report designs

To create a data object data set, use the newDataMartDataSet() method from ElementFactory. For example, to instantiate a new data set, use the following code:

```

DataMartDataSetHandle dataSet =
  factory.newDataMartDataSet("Data Set");

```

Associate the data object data cube with a DataMartDataSourceHandle object and then add the name of a data set from the data or data design file. For example, to access a data set called "SetName", use the following code:

```

dataSet.setDataSource( dataSource.getName( ) );
dataSet.setDataObject( "SetName" );

```

DataMartDataSetHandle inherits the setDataSource() method from DataSetHandle.

Finally, add the data element to the report design, as shown in the following code:

```

designHandle.getDataSets( ).add( dataSet );

```

Creating data object data cubes for BIRT report designs

To create a data object data cube, use the newDataMartDataCube() method from ElementFactory. For example, to instantiate a new data cube, use the following code:

```

DataMartDataCubeHandle dataCube =
  factory.newDataMartDataCube("Data Cube");

```

Associate the data object data cube with a DataMartDataSourceHandle object and assign a data cube from the data or data design file. For example, to access a data cube called "CubeName", use the following code:

```
dataCube.setDataSource( dataSource.getName( ) ) ;  
dataCube.setDataObject( "CubeName" ) ;
```

Finally, add the data element to the report design, as shown in the following code:

```
designHandle.getDataCubes( ) .add( dataCube ) ;
```

Tutorial 1: Creating a data element using the Design Engine API

This tutorial provides step-by-step instructions for creating a Java class that generates a BIRT report design with a BIRT data source generated from a BIRT data design file. You perform the following tasks:

- Set up a project.
- Create a GenerateDataObject Java class.
- Create the main() method to test the code.
- Run the code.

Task 1: Set up a project

To compile a Design Engine API application, the design engine Java archive (JAR) files from Actuate iHub must be in your classpath. You can find the design engine JAR files in the <Actuate home>/modules/BIRTiHub/iHub/Jar/BIRT/lib directory folder. The main JAR files that contain the design engine classes are coreapi.jar and modelapi.jar files. In addition, you need a data design file from which to generate the data objects. For this tutorial, the data design file is include.datadesign.

- 1 In Java perspective, select File->New->Project. New Project appears, as shown in Figure 3-1.

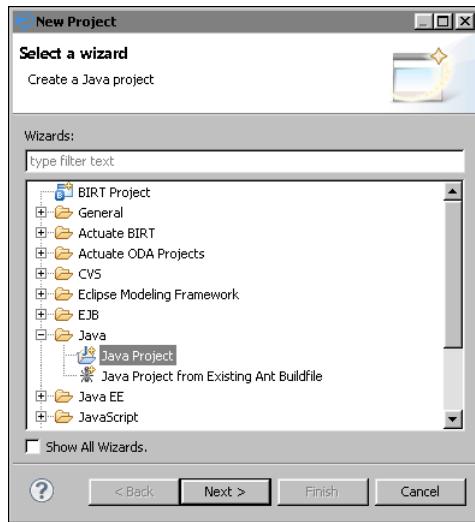


Figure 3-1 Creating a new project

- 2** Expand Java, select Java Project, and choose Next. New Java Project appears, as shown in Figure 3-2.

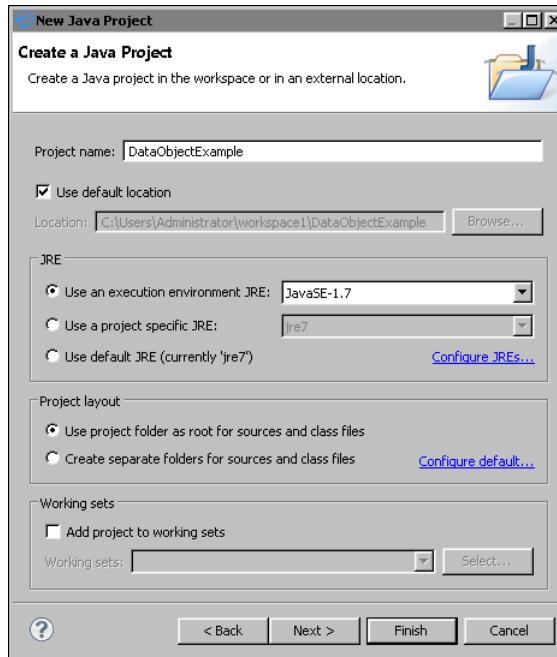


Figure 3-2 Creating the DataObjectExample project

3 In Project Name type:

DataObjectExample

4 In Project layout, select:

Use project folder as root for sources and class files

5 Choose Next. Java Settings appears.

6 Set the project build path.

1 Select the Libraries tab.

2 Choose Add External JARs.

3 In JAR Selection, navigate to the iHub\Jar\BIRT\lib directory. For the default installation of BIRT on Windows XP, this directory is:

C:\Program Files\BIRTiHubVisualization\modules\BIRTiHub\iHub\Jar\BIRT\lib

4 In JAR Selection, select all of the JAR files in the directory.

5 Choose Open. The libraries are added to the classpath as shown in Figure 3-3.

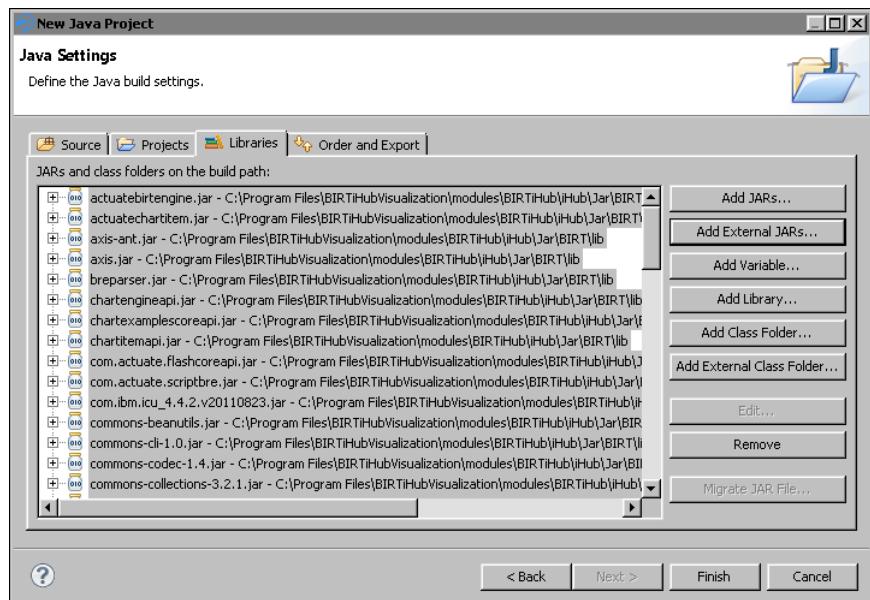


Figure 3-3 DataObjectsAPI project build path

6 Choose Finish.

7 Import the data design file.

1 In the Package Explorer, right-click the DataObjectExample project.

- 2 Choose Import from the context menu.
- 3 In Import, choose General->File System and then choose Next.
- 4 In File System, next to the From Directory field, choose Browse.
- 5 Navigate to and select a data design file. Then choose Finish. The data design file appears in the project as shown in Figure 3-4.

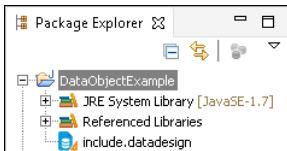


Figure 3-4 DataObjectExample project showing the data design file

Task 2: Create a GenerateDataObject Java class

This Java class creates a simple report design, with table, list, and image elements.

- 1 Choose File->New->Class. New Java Class appears.
- 2 In Name type:
GenerateDataObject
- 3 In Package, as shown in Figure 3-5, type:
myPackage

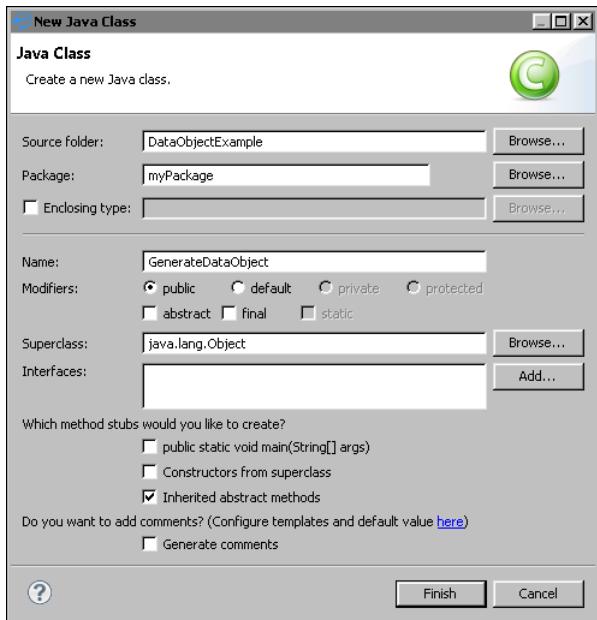


Figure 3-5 Creating a GenerateDataObject class

- 4 Choose Finish. GenerateDataObject.java opens in the Java editor.
- 5 Add a BIRT_HOME static variable to the class. For the default installation of iHub on a Windows system, use the following line in the body of the GenerateDataObject class body:

```
private static final String BIRT_HOME = "C:/Program Files  
/BIRTiHubVisualization/modules/BIRTiHub/iHub/Jar/BIRT  
/platform";
```

Task 3: Create the main() method to test the code

Add a main() method to run the class.

- 1 Type the following main method:

```
public static void main( String[] args ) throws Exception{}
```

An error indicating that the BirtException class is not defined appears.

- 2 Use Quick Fix (Ctrl+1) to import the BirtException class definition.
- 3 Add the main method body shown in Listing 3-1 to your main() method.

Listing 3-1 main() method code

```
DesignConfig config = new DesignConfig( );
```

```

config.setBIRTHome( BIRT_HOME ) ;

DesignEngine engine = new DesignEngine( config );
SessionHandle sessionHandle = engine.newSessionHandle(
    ULocale.ENGLISH );
ReportDesignHandle designHandle = sessionHandle.createDesign();
ElementFactory factory = designHandle.getElementFactory( );

DataMartDataSourceHandle dataSource =
    factory.newDataMartDataSource( "Data Source" );
dataSource.setDataMartURL( "include" );
dataSource.setAccessType(
    DesignChoiceConstants.ACCESS_TYPE_TRANSIENT );
designHandle.getDataSources( ).add( dataSource );

FileOutputStream fos = new FileOutputStream("test.rptdesign");
designHandle.serialize( fos );
fos.close( );

designHandle.close( );
System.out.println("Done");

```

Read the code explanations:

- To access a data source and its contents, the application must first generate and configure a design engine object.
- After creating the engine object, the code instantiates a new session. The SessionHandle object manages the state of all open data and report designs. Use SessionHandle to open, close, and create data designs, and to set global properties, such as the locale and the units of measure for data elements. Create the session handle only once. BIRT supports only a single SessionHandle.
- Generate a new design handle using the SessionHandle object. Create a design engine element factory using the DesignHandle object.
- Create a new instance of DataMartDataSourceHandle and set the datamart URL to the name of a datamart file, include, which corresponds to the include.datadesign file added to the project. Then, configure the access type and add the data source handle to the design handle object.
- Finally, open a file output stream to a report design, test.rptdesign, that uses the data object. Export the data design element to the report design.

- 4 Add the import statements shown in Listing 3-2 to the beginning of the file.

Listing 3-2 import statement code

```
import java.io.FileOutputStream;
import org.eclipse.birt.report.model.api.DesignConfig;
import org.eclipse.birt.report.model.api.DesignEngine;
import org.eclipse.birt.report.model.api.ElementFactory;
import org.eclipse.birt.report.model.api.ReportDesignHandle;
import org.eclipse.birt.report.model.api.SessionHandle;
import org.eclipse.birt.report.model.api.elements
    .DesignChoiceConstants;
import com.actuate.birt.report.model.api
    .DataMartDataSourceHandle;
import com.ibm.icu.util.ULocale;
```

Task 4: Run the code

- 5 Create a Run configuration for GenerateDataObject.java class.

1 In Package Explorer, select:

GenerateDataObject.java

2 From the main menu, choose Run->Run Configurations.

3 Double-click the Java Application link in the left frame of Run Configurations. The GenerateDataObjects configuration gets created.

4 Choose Run. Save and Launch appears. Choose OK.

- 6 After the execution completes, refresh the contents of the DataObjectExample project. test.rptdesign appears.

- 7 Open the report design and view the XML source. The XML contains a datamart element that points to include.datadesign and a data source called include, as shown in the following code:

```
<datamart-node
location="file:/DataObjectExample/include.datadesign">
...
<data-sources>
    <data-mart-data-source name="Data Source" id="4">
        <property name="datamartURL">include</property>
        <property name="accessType">transient</property>
    </data-mart-data-source>
</data-sources>
```

- 8 In Data Explorer, expand Data Sources to view the new data source, as shown in Figure 3-6.

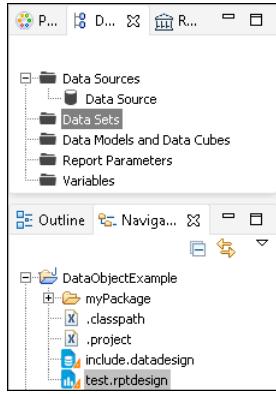


Figure 3-6 Data Source in test.rptdesign

The final code for `GenerateDataObject` is shown in Listing 3-3.

Listing 3-3 GenerateDataObject.java

```
package myPackage;
import java.io.FileOutputStream;
import org.eclipse.birt.report.model.api.DesignConfig;
import org.eclipse.birt.report.model.api.DesignEngine;
import org.eclipse.birt.report.model.api.ElementFactory;
import org.eclipse.birt.report.model.api.ReportDesignHandle;
import org.eclipse.birt.report.model.api.SessionHandle;
import org.eclipse.birt.report.model.api.elements
    .DesignChoiceConstants;
import com.actuate.birt.report.model.api.DataMartDataSourceHandle;
import com.ibm.icu.util.ULocale;

public class GenerateDataObject {

    private static final String BIRT_HOME =
        "C:/Program Files/BIRTiHubVisualization/modules/BIRTiHub/iHub
        /Jar/BIRT/platform";

    public static void main( String[] args ) throws Exception {
        DesignConfig config = new DesignConfig();
        config.setBIRTHome( BIRT_HOME );
        DesignEngine engine = new DesignEngine( config );
        SessionHandle sessionHandle = engine.newSessionHandle(
            ULocale.ENGLISH );
        ReportDesignHandle designHandle =
            sessionHandle.createDesign();
        ElementFactory factory = designHandle.getElementFactory();
    }
}
```

```
        DataMartDataSourceHandle dataSource =
factory.newDataMartDataSource( "Data Source" );
dataSource.setDataMartURL( "include" );
dataSource.setAccessType( DesignChoiceConstants
    .ACCESS_TYPE_TRANSIENT );
designHandle.getDataSources( ).add( dataSource );

        FileOutputStream fos = new
            FileOutputStream("test.rptdesign");
designHandle.serialize( fos );
fos.close();

        designHandle.close();
System.out.println("Done");
}
}
```


Part **TWO**

**Using Actuate JavaScript API
in an application**

4

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
- Using dashboards and gadgets
- Navigating repository content using ReportExplorer
- Using and submitting report parameters
- Retrieving report content as data
- Controlling Interactive Viewer user interface features

About the Actuate JavaScript API

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

To use the Actuate JavaScript API, connect to Actuate iHub Visualization Platform client 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>\iportal\jsapi
```

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

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

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

Accessing the Actuate JavaScript API

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

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
      "http://www.w3.org/TR/html4/strict.dtd">  
<html>  
  <head>  
    <meta http-equiv="content-type" content="text/html;  
          charset=utf-8" />
```

```

</head>
<body>
  <div id="viewer1">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/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://127.0.0.1:8700/iportal/jsapi">
</script>

```

where

- 127.0.0.1:8700 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://127.0.0.1:8700/iportal", reqOps, null,  
    null, runReport, null);
```

- http://127.0.0.1:8700/iportal is a URL for the Actuate report application service. This URL must correspond to an Actuate Deployment Kit for BIRT Reports application or iHub Visualization Platform client application.
- reqOps specifies an actuate.RequestOptions object, which is required for most operations. A default request options object can be generated by calling the constructor without any input parameters, as shown in the following code:

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

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

```
var reqOps = new actuate.RequestOptions();  
reqOps.setVolumeProfile("Default Volume");  
reqOps.setRepositoryType(  
    actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA);
```

- 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:

- **BIRT Viewer Toolkit:** Actuate BIRT Viewer Toolkit is a freeware BIRT Viewer that is secured by the web server that runs it. BIRT Viewer Toolkit does not perform an authentication step initially, which enables the Actuate JavaScript API to integrate smoothly.
- **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.
- **Deployment Kit using a volume repository:** Volumes are managed by Actuate BIRT iHub. To connect to a Deployment Kit that accesses a volume, an Actuate JavaScript API web page prompts the user for a user name and password if a secure session has not been established.
- **iHub Visualization Platform client:** iHub Visualization Platform client connects to a volume and requires authentication. To connect to an iHub Visualization Platform client, an Actuate JavaScript API web page prompts the user for a user name and password if a secure session has not been established. iHub Visualization Platform client provides a login page to establish the secure session.

See Chapter 19, “Using Visualization Platform security,” for information about customizing security for a Deployment Kit or iHub Visualization Platform client.

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 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);
```

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

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

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

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

Using a login servlet to connect to an Actuate web application

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

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

```
<input type="text" name="password" />  
...  
</form>
```

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

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

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 Chapter 19, “Using Visualization Platform security,” 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 );
```

- 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 iHub Visualization Platform client 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 the viewer component 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.rptdocument"` indicates the Customer List report document in the `/public` directory.

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

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

The `submit()` function submits all the asynchronous operations that previous viewer functions prepare and triggers an AJAX request for the report. The Actuate web application returns the report and the page displays the report in the assigned `<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://127.0.0.1:8700/iportal/jsapi"></script>

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

        function init( ){
            actuate.load("viewer");
            var reqOps = new actuate.RequestOptions( );
            actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
                null, null, runReport);
        }

        function runReport( ) {
            var viewer = new actuate.Viewer( "viewerpane" );
            viewer.setReportName(
                "/Public/UnshippedOrders1H2013.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");
    var reqOps = new actuate.RequestOptions( );
    actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
        null, null, runReport);
}

function runReport( ) {
    var viewer = new actuate.Viewer( "viewerpane" );
    viewer.setReportName(
        "/Public/UnshippedOrders1H2013.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.

Tutorial 2: Implementing the JSAPI in a web page to display the viewer

This tutorial provides step-by-step instructions for authoring a web page to display a BIRT report in BIRT Viewer. Write a function to display a report in the Actuate Viewer embedded in a web page.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```
<!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>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here --&gt;
    &lt;/script&gt;
  &lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

```
JSAPI Template
```

to:

```
Report Viewer Page
```

- 3 Navigate to the following line:

```
<div id="sample">
```

In id, change:

```
sample
```

to:

```
content
```

- 4 Navigate to the empty line after the following line:

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

- 5 Add the following code:

```
function init( ) {
    actuate.load("viewer");
    var reqOps = new actuate.RequestOptions( );
    actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
        null, null, runReport);
}
function runReport( ) {
    var viewer = new actuate.Viewer( "viewerpane" );
    viewer.setReportName(
        "/Public/UnshippedOrders1H2013.rptdesign" );
    viewer.setReportletBookmark( "FirstTable" );
    viewer.submit( callback );
}
```

- 6 Save the file as reportviewer.html.

- 7 In Internet Explorer, open reportviewer.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

- 8 A Login required dialog appears, as shown in Figure 4-1. Type administrator into the User name field and choose Sign in.

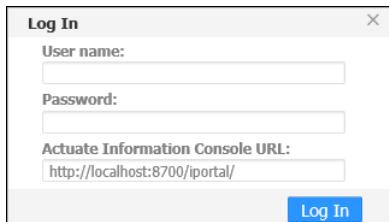


Figure 4-1 Login required dialog

If the page is blank, refresh the browser.

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 4-2.



Figure 4-2 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.

Using dashboards and gadgets

The actuate.Dashboard class loads and displays dashboards and provides access to the gadgets contained in dashboards. The Dashboard class requires a pre-existing actuate.RequestOptions object with the repository type set loaded with initialize. To use the RequestOptions to access dashboard content residing in a volume, use the RequestOptions constructor, use setRepositoryType() to set the repository to encyclopedia, and provide the object as a parameter to the initialize call, as shown in the following code:

```
var reqOps = new actuate.RequestOptions( );
reqOps.setRepositoryType(
  actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA);
actuate.initialize("http://127.0.0.1:8700/iportal", reqOps, null,
  null, runDashboard );
```

To use actuate.Dashboard, load the class with actuate.load() before initialize(), as shown in the following code:

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

Load the dashboard a 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/sharedtab.dashboard"` indicates the shared tab dashboard 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://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
function init(){
    actuate.load("dashboard");
    var reqOps = new actuate.RequestOptions();
    reqOps.setRepositoryType(
```

```

        actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA) ;
actuate.initializeApp("http://127.0.0.1:8700/iportal", reqOps,
    null, null, runDashboard );
} function runDashboard( ){
var dash = new actuate.Dashboard("dashboardpane");
dash.setDashboardName ("/sharedtab.dashboard");
dash.setIsDesigner(false);
dash.submit( );
}
</script>
</div>
</body>
</html>

```

Tutorial 3: Implementing the JSAPI in a web page to display a dashboard

This tutorial provides step-by-step instructions for authoring a web page to display a BIRT dashboard.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```

<!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>JSAPI Template</title>
</head>
<body onload="init( )">
<div id="sample">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">

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

```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

Dashboard Display Page

- 3** Navigate to the following line:

```
<div id="sample">
```

In id, change:

sample

to:

dashboard

- 4** Navigate to the empty line after the following line:

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

- 5** Add the following code:

```
function init( ){
    var reqOps = new actuate.RequestOptions( );
    reqOps.setRepositoryType(
        actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA);
    actuate.initialize("http://127.0.0.1:8700/iportal",
        reqOps, "administrator", "", displayDashboard);
}
function displayDashboard( ){
    var mydashboard = new actuate.Dashboard("dashboard");
    mydashboard.setDashboardName(
        "/Dashboard/Contents/Documents.DASHBOARD");
    mydashboard.submit();
}
```

- 6** Save the file as dashboarddisplay.html.

- 7** In Internet Explorer, open dashboarddisplay.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

If you receive a scripting error, choose OK. This error is generated by the Documents.DASHBOARD file, not dashboarddisplay.html.

If the dashboard does not finish loading, in Internet Explorer, choose Tools->Internet Options. Select the Privacy tab, then choose Advanced. Select “Override automatic cookie handling.” Choose Apply. Refresh the browser.

Tutorial 4: Implementing the JSAPI to catch exceptions with an error callback function

This tutorial provides step-by-step instructions for authoring a web page to catch exceptions and test them by generating a specific error to trigger the exception handler. Write an errorHandler function to catch exceptions in a web page.

- 1 Using a code editor, open or create a BasicViewer.html file that contains a basic implementation of the viewer.

```
<!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>Basic Viewer Example</title>
</head>
<body onload="init( )">
<table>
<tr><td class="jsapi_example_label">Report File:<td
colspan="4">
<input type="text" size="80" id="reportfile" value="/Applications/BIRT Sample App/Customer Dashboard.rptdesign"
name="Report_name">
<tr><td><input type="button" class="btn" id="run" value="Run
Viewer" onclick="run()" disabled="true">
</table>

<div id="content">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
var viewer;
function init( ){
    actuate.load("viewer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        null, null, initViewer);
}
function initViewer( ) {
    viewer = new actuate.Viewer("content");
    document.getElementById("run").disabled = false;
}
function run(){
    viewer.setReportName(document.getElementById(
        "reportfile").value);
    viewer.submit();
}
```

```

        }
        <!-- write an event handler to catch exceptions -->
        </script>
    </div>
</body>
</html>

```

For more information about implementing the JSAPI viewer, see “Viewing reports,” earlier in this chapter.

- 2** Navigate to the following line:

```
<!-- write an event handler to catch exceptions -->
```

Replace the line with the following code:

```
function errorHandler(exception) {
    alert("Your application encountered an exception: \n" +
          exception.getMessage());
}
```

- 3** Navigate to the following line:

```
actuate.initialize( "http://127.0.0.1:8700/iportal", null,
                    null, null, initViewer);
```

Add errorHandler to the parameter list as the optional exception handler for initialize, as shown in the following code:

```
actuate.initialize( "http://127.0.0.1:8700/iportal", null,
                    null, null, initViewer, errorHandler);
```

- 4** Save the file as basicexception.html.

- 5** In Internet Explorer, open basicexception.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

- 6** Open basicexception.html. The errorHandler function runs because no login credentials have been provided, as shown in Figure 4-3.

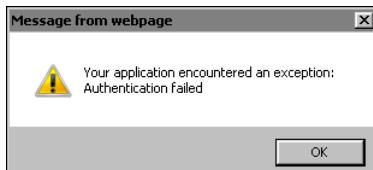


Figure 4-3 Authentication message from errorHandler

Navigating repository content using ReportExplorer

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

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

Call actuate.ReportExplorer functions to identify the root directory to display then call the ReportExplorer's submit function to display the content in the assigned <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();
requestOpts.setRepositoryType(
    actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA );
actuate.initialize( "http://127.0.0.1:8700/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://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">

function init( ) {
    actuate.load("reportexplorer");
    var reqOps = new actuate.RequestOptions( );
    requestOpts.setRepositoryType(
        actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA );
    reqOps.setVolumeProfile( "Default Volume" );
    actuate.initialize( "http://127.0.0.1:8700/iportal" , reqOps,
        null, null, runReportExplorer);
}

function runReportExplorer( ) {
    var explorer = new actuate.ReportExplorer("explorerpane");
    explorer.setFolderName( "/Public" );
```

```

var resultDef =
    "Name|FileType|Version|VersionName|Description";
explorer.setResultDef( resultDef.split(" | ") );
explorer.submit( );
}
</script>
</div>
</body>
</html>

```

The report explorer component displays the contents of the set folder, as shown in Figure 4-4.

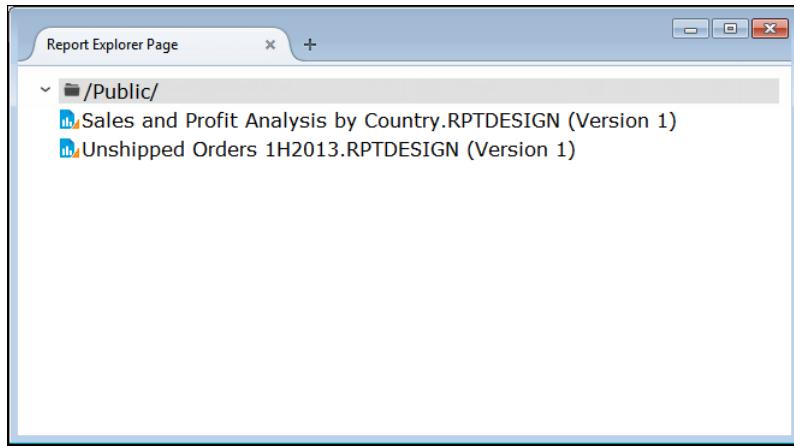


Figure 4-4 Report Explorer page

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

Opening files from ReportExplorer

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

```

var file;
...
explorer.registerEventHandler(
    actuate.reportexplorer.EventConstants.ON_SELECTION_CHANGED,
    selectionChanged );
...

```

```

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

```

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

In this implementation, the file path is updated each time a file selected. To open the file currently selected, implement a button on the page that runs a separate function that opens the file. The following code example shows a button that calls the custom displayReport() function, which attempts to open the file using an actuate.viewer object:

```

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

```

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

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

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text
      /html; charset=utf-8" />
<title>Report Explorer Page</title>
</head>
<body onload="init( )">
<input type="button" style="width: 150pt;" value="View Report"
       onclick="javascript:displayReport( )"/>
<hr />
<div id="explorerpane">
<script type="text/javascript" language="JavaScript"
       src="http://127.0.0.1:8700/iportal/jspapi"></script>

```

```

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

    var file = "unknown";

    function init( ) {
        actuate.load("reportexplorer");
        actuate.load("viewer");
        var reqOps = new actuate.RequestOptions( );
        actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
            null, null, runReportExplorer);
    }

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

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

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

```

Tutorial 5: Displaying repository contents and opening files

In this tutorial, you author a web page that displays a navigable user interface that displays repository contents and allows a user to open a selected file. You perform the following tasks:

- Display the report explorer.
- Track user selections.
- Open a selected file.

Task 1: Display the report explorer

In this task, you create or edit JSAPITemplate.html and edit its contents to display the contents of the /Public directory from the local volume.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```
<!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>JSAPI Template</title>
</head>
<body onload="init( )">
<div id="sample">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
<!-- Insert code here -->
</script>
</div>
</body>
</html>
```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

Report Explorer Page

- 3** Navigate to the following line:

```
<div id="sample">
```

In id, change:

```
sample
```

to:

```
explorer
```

- 4** Navigate to the empty line after the following line:

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

- 5** Add the following code:

```
function init( ) {
    actuate.load("reportexplorer");
    requestOpts = new actuate.RequestOptions();
    actuate.initialize( "http://127.0.0.1:8700/iportal",
        requestOpts, null, null, runReportExplorer);
}
function runReportExplorer( ) {
    var explorer = new actuate.ReportExplorer("explorer");
    explorer.setFolderName( "/Applications" );
    var resultDef = "Name|FileType|Version|VersionName|
        Description";
    explorer.setResultDef( resultDef.split(" | "));
    explorer.submit( );
}
```

- 6** Save the file as repositorydisplay.html.

- 7** In Internet Explorer, open repositorydisplay.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

- 8** If a Login required dialog appears, as shown in Figure 4-5, type administrator into the User name field and choose Sign in.

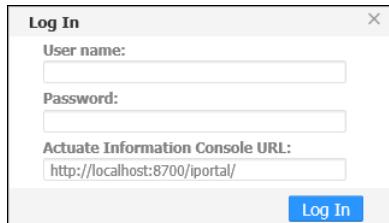


Figure 4-5 Login required dialog

Task 2: Track user selections

In this task, you create or edit JSAPITemplate.html to display the repository contents and track the selections made by the user.

1 Using a code editor, open or create the JSAPITemplate.html file.

2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

Report Selection Page

3 Navigate to the following line:

```
<div id="sample">
```

In id, change:

sample

to:

explorer

4 Navigate to the empty line after the following line:

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

5 Add the following code:

```
function init( ){
    actuate.load("reportexplorer");
    requestOpts = new actuate.RequestOptions();
    actuate.initialize( "http://127.0.0.1:8700/iportal",
        requestOpts, null, null, runReportExplorer);
}
function runReportExplorer( ) {
    var explorer = new actuate.ReportExplorer("explorer");
    explorer.registerEventHandler(
        actuate.reportexplorer.EventConstants
        .ON_SELECTION_CHANGED, selectionChanged );
    explorer.setFolderName( "/Applications" );
    var resultDef = "Name|FileType|Version|VersionName|
        Description";
    explorer setResultDef( resultDef.split("|") );
    explorer.submit( );
}
```

```

function selectionChanged( selectedItem, pathName ) {
    file = pathName;
    alert ("File selected: " + file);
}

```

- 6 Save the file as reportselection.html.
- 7 In Internet Explorer, open reportselection.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

Type administrator into the User name field of the Login required dialog, and choose Sign in.

- 8 On the report explorer, select a file. An alert for the file selected appears, as shown in Figure 4-6.

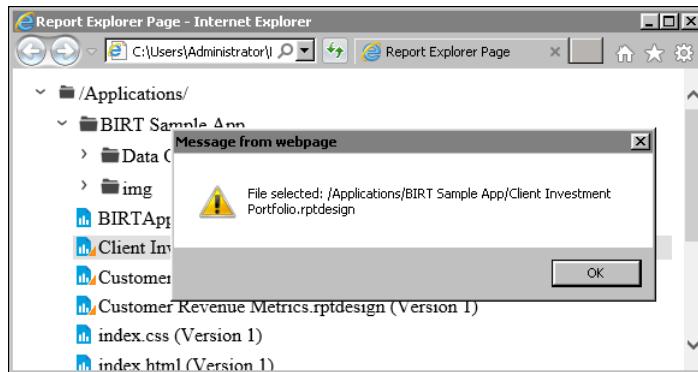


Figure 4-6 File selected alert

Task 3: Open a selected file

In this task, you create a copy of reportselection.html and edit its contents to include a button that opens the selected file in the report explorer.

- 1 Using a code editor, open reportselection.html from the previous task.
- 2 Navigate to the following line:

```
<title>Report Selection Page</title>
```

In title, change:

```
Report Selection Page
```

to:

```
View Selection Page
```

- 3** Navigate to the line after the following line:

```
<body onload="init( )">
```

Add the following code:

```
<input type="button" style="width: 150pt;" value="View Report"
onclick="javascript:displayReport( )"/>
<hr />
```

- 4** Navigate to and delete following line:

```
alert ("File selected: " + file);
```

- 5** After the selectionChanged function definition, add the following code:

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

- 6** Save the file as viewselection.html.

- 7** In Internet Explorer, open viewselection.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

Type administrator into the User name field of the Login required dialog, and choose Sign in.

- 8** Select a report in the report explorer and choose View Report to view the report.

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://127.0.0.1:8700/iportal/jsapi"></script>

<script type="text/javascript" language="JavaScript">
function init( ){
    actuate.load("viewer");
    actuate.load("parameter");
    var reqOps = new actuate.RequestOptions( );
    actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
        null, null, displayParams);
}
function displayParams( ) {
    param = new actuate.Parameter("param1");
    param.setReportName("/Applications/BIRT Sample App/Customer
        Order History.rptdesign");
    param.submit(function ( ) { this.run.style.visibility=
        'visible';});
}
function processParameters( ) {
...
}
</script></div>

```

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

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

```

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

```

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

Accessing parameter values from the viewer

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

```

function runReport (ParameterValues) {

```

```

    var viewer = new actuate.Viewer("viewerpane");
    viewer.setReportName("/Applications/BIRT Sample App
        /Customer Order History.rptdesign");
    viewer.setParameterValues(ParameterValues);
    viewer.submit( );
}

```

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

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

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <meta http-equiv="content-type" content="text/
        html; charset=utf-8" />
    <title>Viewer With Parameters Page</title>
</head>
<body onload="init( )">
    <div id="parampane">
        <script type="text/javascript" language="JavaScript"
            src="http://127.0.0.1:8700/iportal/jsapi"></script>
        <script type="text/javascript" language="JavaScript">
            function init( ){
                actuate.load("viewer");
                actuate.load("parameter");
                var reqOps = new actuate.RequestOptions( );
                actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
                    null, null, displayParams);
            }
            function displayParams( ) {
                param = new actuate.Parameter("parampane");
                param.setReportName("/Applications/BIRT Sample App/Customer
                    Order History.rptdesign");
                param.submit(
                    function ( ) {this.run.style.visibility = 'visible';});
            }
            function processParameters( ) {
                param.downloadParameterValues(runReport);
            }
        </script>
    </div>
    <hr><br />
    <input type="button" class="btn" name="run"
        value="Run Report" onclick="processParameters( )"

```

```

        style="visibility: hidden">

<div id="viewerpane">
<script type="text/javascript" language="JavaScript"
       src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
function runReport(paramvalues) {
    var viewer = new actuate.Viewer("viewerpane");
    viewer.setReportName("/Applications/BIRT Sample App/Customer
                           Order History.rptdesign");
    viewer.setParameterValues(paramvalues);
    viewer.submit();
}
</script>
</div>
</body>
</html>

```

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

/Applications/BIRT Sample App/Customer Order History.rptdesign

Tutorial 6: Implementing the JSAPI in a web page to display report parameters

This tutorial provides step-by-step instructions for authoring a web page to display a BIRT report with parameters in BIRT Viewer.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```

<!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>JSAPI Template</title>
</head>
<body onload="init( )">
    <div id="sample">
        <script type="text/javascript" language="JavaScript"
               src="http://127.0.0.1:8700/iportal/jsapi"></script>
        <script type="text/javascript" language="JavaScript">
        <!-- Insert code here -->

```

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

- 2** Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

```
JSAPI Template
```

to:

```
Report Parameters Page
```

- 3** Navigate to the following line:

```
<div id="sample">
```

In id, change:

```
sample
```

to:

```
parameters
```

- 4** Navigate to the empty line after the following line:

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

- 5** Add the following code:

```
function init( ){
    actuate.load("viewer");
    actuate.load("parameter");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        "administrator", "", displayParams);
}
function displayParams( ) {
    param = new actuate.Parameter("parameters");
    param.setReportName("/Applications/BIRT Sample App
        /Customer Order History.rptdesign");
    param.submit( function ( )
        {document.getElementById("run").style.visibility =
            'visible';});
}
function processParameters( ) {
    param.downloadParameterValues(runReport);
}
</script>
</div>
```

```

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

<div id="viewer">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
    function runReport(paramvalues) {
        var viewer = new actuate.Viewer("viewer");
        viewer.setReportName("/Applications/BIRT Sample App/
Customer Order History.rptdesign");
        viewer.setParameterValues(paramvalues);
        viewer.submit();
    }

```

- 6** Save the file as parameterviewer.html.
- 7** In Internet Explorer, open parameterviewer.html.
If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.
- 8** Choose Run Report to view the report.

Tutorial 7: Changing parameter values and definitions

This tutorial provides step-by-step instructions for authoring a web page that implements the `parameterValue` and `parameterDefinition` classes to set parameter values and control the visibility of parameters in a BIRT report displayed in BIRT Viewer. You perform the following tasks:

- Create a web page that processes parameters.
- Display a hidden parameter using ParameterDefinition.

The file in this tutorial with a hidden parameter is Sales by Territory.rptdesign.

Task 1: Create a web page that processes parameters

In this task, you open or create a copy of JSAPITemplate.html and edit its contents to display a parameter and pass the value selected by the user to a BIRT report displayed in BIRT Viewer.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```
<!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>JSAPI Template</title>
</head>
<body onload="init( )">
<div id="sample">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
<!-- Insert code here --&gt;
&lt;/script&gt;
&lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

Viewer with Parameters Page

- 3 Navigate to the following line:

```
<div id="sample">
```

In id, change:

sample

to:

parampane

- 4 Navigate to the empty line after the following line:

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

- 5 Add the following code:

```
function init( ){
    actuate.load("viewer");
    actuate.load("parameter");
```

```

        actuate.initialize( "http://127.0.0.1:8700/iportal", null,
            "administrator", "", displayParams);
    }
    function displayParams( ) {
        param = new actuate.Parameter("parampane");
        param.setReportName("/Applications/BIRT Sample App
            /Sales by Territory.rptdesign");
        param.submit(function ( )
            {document.getElementById("run").style.visibility =
            'visible';} );
    }
    function processParameters( ) {
        param.downloadParameterValues(runReport);
    }
</script>
</div>

<hr><br />
<input type="button" class="btn" id="run" name="run" value="Run
    Report" onclick="processParameters( )">

<div id="viewerpane">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
function runReport(paramvalues) {
    var viewer = new actuate.Viewer("viewerpane");
    viewer.setReportName("/Applications/BIRT Sample App
        /Sales by Territory.rptdesign");
    viewer.setParameterValues(paramvalues);
    viewer.submit( );
}

```

6 Save the file as processparameters.htm.

7 In Internet Explorer, open processparameters.htm.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

8 On Please specify a territory, choose Japan, then choose Run Report. The Sales by Territory report for Japan appears, as shown in Figure 4-7.

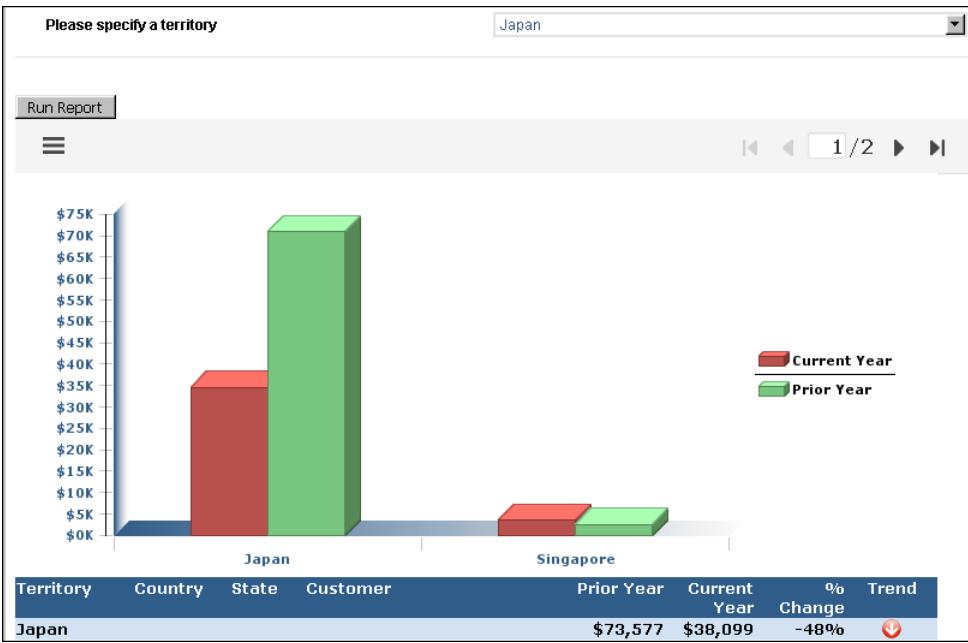


Figure 4-7 Sales by Territory for Japan

Task 2: Display a hidden parameter using ParameterDefinition

In this task, you display a hidden parameter. Sales by Territory.rptdesign has two parameters, currentYear and Territory, but currentYear is a hidden parameter. To display currentYear, you use parameterDefinition to show the parameter normally.

- 1 Using a code editor, open processparameters.htm.
- 2 Navigate to the following line:

```
param.submit(function ()  
{document.getElementById("run").style.visibility =  
'visible';} );
```

Replace the line with the following two lines:

```
param.submit();  
param.downloadParameters(changeText);
```

- 3 Add the changeText function to the parampane div element, as shown in the following code:

```
<div id="parampane">
```

```

<script type="text/javascript" language="JavaScript"
    src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
    function init( ) {
        actuate.load("viewer");
        actuate.load("parameter");
        actuate.initialize( "http://127.0.0.1:8700/iportal", null,
            "administrator", null, displayParams);
    }
    function displayParams( ) {
        param = new actuate.Parameter("parampane");
        param.setReportName("/Applications/BIRT Sample App
            /Sales by Territory.rptdesign");
        param.submit( );
        param.downloadParameters(changeText);
    }
    function processParameters( ) {
        param.downloadParameterValues(runReport);
    }
    function changeText( paramdef ) {
        paramdef[0].setIsHidden(false);
        param.renderContent( paramdef );
    }
</script>
</div>

```

- 4 Save the file as displayparameters.htm.
- 5 In Internet Explorer, open displayparameters.htm.
If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.
- 6 The currentYear parameter for Sales by Territory.rptdesign appears, as shown in Figure 4-8.

The screenshot shows a web interface for a report. At the top, there is a label 'currentYear' followed by a dropdown menu. The dropdown menu is open, displaying three items: '...', 'NA', and 'EU'. Below the dropdown is a button labeled 'Run Report'.

Figure 4-8 currentYear parameter displayed on a web page

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 the viewer component 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.

Controlling Interactive Viewer user interface features

The BIRT Interactive Viewer enables users to perform a number of custom operations on a BIRT design or document and save or print changes as a new design or document. The file and print features for Interactive Viewer are available in the main menu of the BIRT viewer, as shown in Figure 4-9.

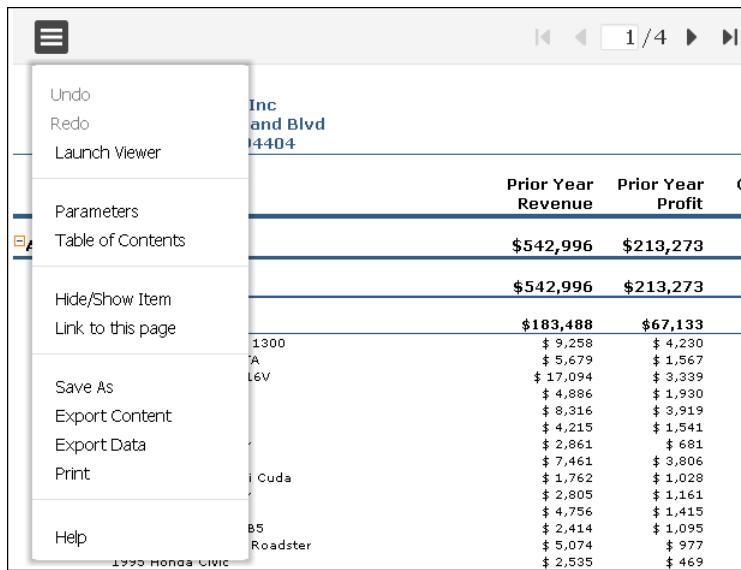


Figure 4-9 Interactive Viewer menu

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

All `actuate.viewer.UIOptions` enable functions accept a Boolean input parameter, which if true enables an interactive feature and if false disables an interactive feature. To display the list of currently enabled and disabled features, use `actuate.viewer.UIOptions.getFeatureMap()`. Table 4-1 contains a complete list of `actuate.viewer.UIOptions` enable functions that control features.

Table 4-1 UIOptions enable feature functions

Function	Interactive feature
<code>enableAdvancedSort()</code>	Enables the advanced sort feature
<code>enableAggregation()</code>	Enables the aggregation feature
<code>enableCalculatedColumn()</code>	Enables the calculated column feature

Table 4-1 UIOptions enable feature functions (continued)

Function	Interactive feature
enableChartProperty()	Enables the chart properties feature.
enableChartSubType()	Enables the chart subtype selection feature
enableCollapseExpand()	Enables the collapse/expand feature
enableColumnEdit()	Enables the column editing feature
enableColumnResize()	Enables the column resizing feature
enableContentMargin()	Enables the content margin feature
enableDataAnalyzer()	Enables the Launch Interactive Crosstab feature
enableDataExtraction()	Enables the data extraction feature
enableEditReport()	Enables the report editing/interactivity feature
enableExportReport()	Enables the export report feature
enableFilter()	Enables the filter feature
enableFlashGadgetType()	Enables the flash gadget type change feature
enableFormat()	Enables the format editing feature
enableGroupEdit()	Enables the group editing feature
enableHideShowItems()	Enables the hide/show item feature
enableHighlight()	Enables the highlight feature
enableHoverHighlight()	Enables the hover highlight feature
enableLaunchViewer()	Enables the Launch Viewer feature
enableLinkToThisPage()	Enables the "link to this page" feature
enableMainMenu()	Enables the main menu feature
enableMoveColumn()	Enables the column moving feature
enablePageBreak()	Enables the page break editing feature
enablePageNavigation()	Enables the page navigation feature
enableParameterPage()	Enables the parameter page feature
enablePrint()	Enables the print feature
enableReorderColumns()	Enables the column reordering feature
enableRowResize()	Enables the row resizing feature

(continues)

Table 4-1 UIOptions enable feature functions (continued)

Function	Interactive feature
enableSaveDesign()	Enables the report design save feature
enableSaveDocument()	Enables the report document save feature
enableServerPrint()	Enables the server-side printing feature
enableShowToolTip()	Enables the show tooltip feature
enableSort()	Enables the sort feature
enableSuppressDuplicate()	Enables the duplication suppression feature
enableSwitchView()	Enables the switch view feature
enableTextEdit()	Enables the text editing feature
enableTOC()	Enables the table of contents feature
enableToolBar()	Enables the toolbar feature
enableToolbarContextMenu()	Enables the show toolbar features in a context menu
enableToolbarHelp()	Enables the toolbar help feature
enableTopBottomNFilter()	Enables the top N and bottom N filter feature
enableUndoRedo()	Enables the undo and redo feature

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

Disabling UI features in a custom web page

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

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

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

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

The viewer configured with the parameter page feature disabled does not show the Parameters option in the main menu, as shown in Figure 4-10.

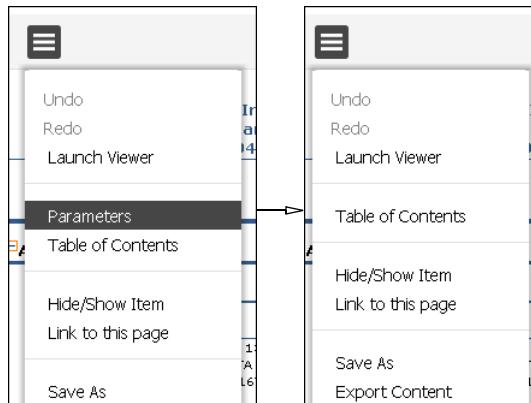


Figure 4-10 The main menu with parameters disabled

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

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

Set the user interface options with the enable functions in the `actuate.viewer.UIOptions` class. For example, a toolbar appears in the viewer by default, as shown in Figure 4-11.



Figure 4-11 The default toolbar for the JavaScript API viewer

To disable this toolbar, use the following code:

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

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

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

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

Tutorial 8: Control the BIRT Interactive Viewer user interface

In this tutorial, you create web pages that customize the user interface based on data collected about the user, the user's browser, and the user's actions. You perform the following tasks:

- Adjust UI configuration according to browser.
- Adjust UI configuration according to user actions.
- Adjust viewer options based on a user input.

Task 1: Adjust UI configuration according to browser

In this task, you open or create a copy of JSAPITemplate.html and edit its contents to collect data from the user's browser and add the BrowserPanel UIConfig to the viewer if the browser is Internet Explorer.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```
<!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>JSAPI Template</title>
</head>
<body onload="init( )">
<div id="sample">
<script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>
<script type="text/javascript" language="JavaScript">
<!-- Insert code here --&gt;
&lt;/script&gt;
&lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

Interactive Viewer with browser-specific Controls

- 3** Navigate to the following line:

```
<div id="sample">
```

In id, change:

sample

to:

viewer

- 4** Navigate to the empty line after the following line:

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

- 5** Add the following code:

```
function init( ) {
    actuate.load("viewer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
                      "administrator", "", runReport);
}
function runReport( ) {
    var browserName = navigator.appName;
    if (browserName == "Microsoft Internet Explorer"){
        var ieUIConfig = new actuate.viewer.UIConfig( );
        ieUIConfig.setContentPanel(new
            actuate.viewer.BrowserPanel());
        var myviewer = new actuate.Viewer("viewer", ieUIConfig);
    } else {
        var myviewer = new actuate.Viewer("viewer");
    }
    myviewer.setReportName("/Applications/BIRT Sample App
                           /Top 5 Sales Performers.rptdesign");
    myviewer.submit( );
}
```

- 6** Save the file as browsercontrols.html.

- 7** In Internet Explorer, open browsercontrols.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

Task 2: Adjust UI configuration according to user actions

In this task, you implement buttons in HTML to collect user decisions regarding the interactive viewing feature.

1 Using a code editor, open or create the JSAPITemplate.html file.

2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

```
JSAPI Template
```

to:

```
Viewer Options Page
```

3 Navigate to the empty line after the following line:

```
<body onload="init()">
```

4 Add the following code:

```
<input type="button" id="runviewer" value="Run BIRT Viewer"
       onclick="runViewer( )">
<input type="button" id="runinteractive" value="Run Interactive
       Viewer" onclick="runInteractive( )">
```

5 Navigate to the following line:

```
<div id="sample">
```

In id, change:

```
sample
```

to:

```
contentpane
```

6 Navigate to the empty line after the following line:

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

7 Add the following code:

```
var myViewer;
function init(){
    actuate.load("viewer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
                      "administrator", "", initViewer);
}
function initViewer(){
    myviewer = new actuate.Viewer("contentpane");
```

```

}
function runViewer() {
    myviewer.setReportName("/Applications/BIRT Sample App
        /Sales by Customer.rptdesign");
    myviewer.submit(function() {myviewer.disableIV();});
}
function runInteractive() {
    myviewer.setReportName("/Applications/BIRT Sample Apps
        /Sales by Customer.rptdesign");
    myviewer.submit(function() {myviewer.enableIV();});
}

```

- 8** Save the file as vieweroptions.html.
- 9** In Internet Explorer, open vieweroptions.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

Task 3: Adjust viewer options based on a user input

In this task, you create custom UIOptions depending upon a name the user provides. This is not a secure method of controlling features, as the JSAPI initializes using the administrator credentials, but demonstrates a way to restrict viewer options using a value on the page.

- 1** Using a code editor, open or create the JSAPITemplate.html file.
- 2** Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

User Restricted Page

- 3** Navigate to the empty line after the following line:

```
<body onload="init()">
```

- 4** Add the following code:

```

User Name:
<input type="text" size="80" id="username" value="Guest"
    name="user_name">
<input type="button" class="btn" id="run" value="Run Viewer"
    onclick="run()>
```

- 5** Navigate to the following line:

```
<div id="sample">
```

In id, change:

```
sample
```

to:

```
contentpane
```

- 6** Navigate to the empty line after the following line:

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

- 7** Add the following code:

```
function init(){
    actuate.load("viewer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        "administrator", "", null);
}
function run(){
    var manUIOptions = new actuate.viewer.UIOptions( );
    var username = document.getElementById("username").value;
    if (username != "administrator" && username !=
        "Administrator"){
        manUIOptions.enableMainMenu(false);
    }
    var myviewer = new actuate.Viewer("contentpane");
    myviewer.setUIOptions(manUIOptions);
    myviewer.setReportName("/Applications/BIRT Sample App
        /Crossstab Sample Revenue.rptdesign");
    myviewer.submit();
}
```

- 8** Save the file as userrestrictions.html.

- 9** In Internet Explorer, open userrestrictions.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

- 10** For User Name, use the default value, Guest, and choose Run Viewer. The viewer loads but does not display the main menu, as shown in Figure 4-12.

- 11** To enable the main menu, in User Name, type Administrator, and choose Run Viewer. The viewer displays the main menu, as shown in Figure 4-13.

User Name: Guest

Run Viewer

1 / 1

PRODUCTLINE			Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Grand Total
Year	Quarter	Month	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue
Year/QTR /Month 2012	1	1	\$109,562	\$39,987	\$31,159	\$26,310	\$6,387		\$42,909	\$256,315
		2	\$108,232	\$45,694	\$34,000	\$24,894	\$4,763	\$35,749	\$48,688	\$302,021

Figure 4-12 Entering Guest disables the main menu in the viewer

User Name: administrator

Run Viewer

1 / 1

PRODUCTLINE			Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Grand Total
Year	Quarter	Month	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue
Year/QTR /Month 2012	1	1	\$109,562	\$39,987	\$31,159	\$26,310	\$6,387		\$42,909	\$256,315
		2	\$108,232	\$45,694	\$34,000	\$24,894	\$4,763	\$35,749	\$48,688	\$302,021

Figure 4-13 Entering Administrator enables the main menu in the viewer

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

About Actuate JavaScript API scripting in a BIRT report design

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

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

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

A BIRT report or Reportlet renders in the following ways:

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

All of these products load the `actuate.Viewer` and `actuate.Dialog` classes when they render a report, except for the preview functionality in BIRT Designer. Use the View Report in Web Viewer function to view and test Actuate JavaScript API scripts with BIRT Designer, as shown in Figure 5-1.

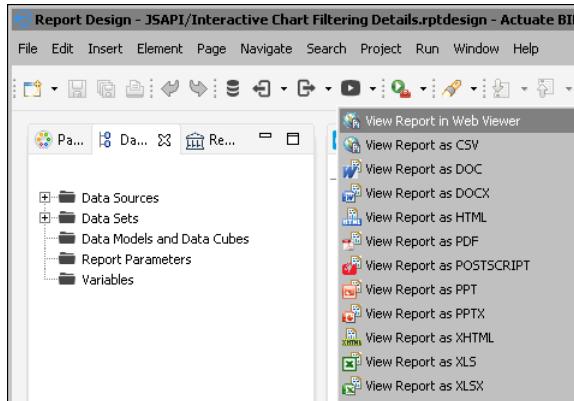


Figure 5-1 Accessing Web Viewer in Actuate BIRT Designer

Most of the classes and functions in the `actuate.Viewer` class can be used in a BIRT report design without loading or initializing the `actuate.Viewer` class and. Most of the viewers also load the `actuate.Parameters` and `actuate.DataService` classes by

default. Define the classes loaded for Actuate JavaScript API mashup page explicitly. Load the DataService, Parameters, and Viewer classes before the API initializes the connection to the reporting web service.

Using the Actuate JavaScript API in an HTML button

The HTML button element can execute client-side JavaScript code based on button events. Access the HTML button in the BIRT designer by selecting a button element, choosing the script tag, and selecting the event from the event drop-down list, as shown in Figure 5-2.

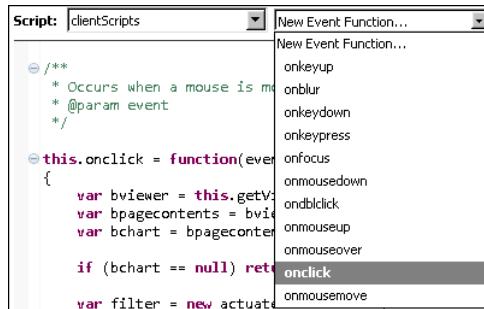


Figure 5-2 Choosing HTML button event handlers

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 5-3.

HTML buttons can be arranged into sets of controls for the user to use once a report runs. For example, when these buttons are used in the header for a table, the header can provide controls similar to those in the header shown in Figure 5-4.

Australian		
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

Australian Coll		
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 5-3 Using a GroupBy HTMLButton control

Figure 5-4 HTML button header

Tutorial 9: Adding scripted chart controls to a BIRT design

In this tutorial, you add HTML buttons to a BIRT design that implement controls for a chart in the BIRT design. You perform the following tasks:

- Add bookmarks.
- Add a filter script to chart interactivity.
- Script the chart size controls.
- Test the scripts.

Task 1: Add HTML buttons

In this task, you review a BIRT report design called ChartWithHTMLButtons.rptdesign and create a grid of HTML buttons.

- 1 Open BIRT Designer Professional. In Navigator, navigate to and open ChartWithHTMLButtons.rptdesign.
- 2 Preview the report, as shown in Figure 5-5.

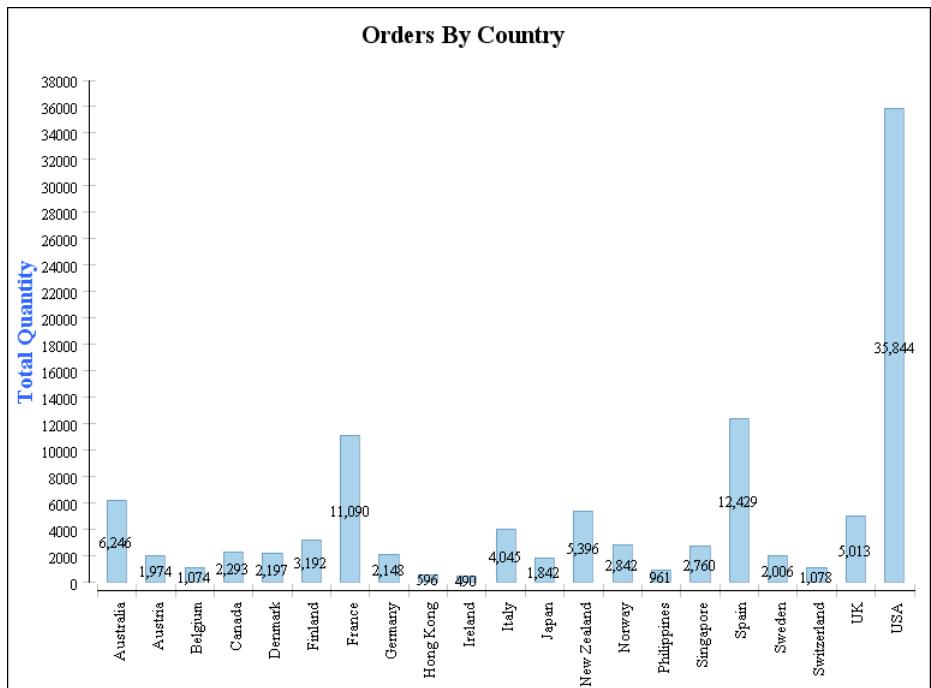


Figure 5-5 Previewing the report

- 3 Choose Layout to return to the layout editor.
- 4 Right-click the first cell of the table. Choose Insert->Grid. On Insert Grid, set Number of columns to 2 and Number of rows to 2, then choose OK. A new grid appears above the chart, as shown in Figure 5-6.



Figure 5-6 Inserting a grid

- 5 To create HTML buttons for the report, perform the following steps:
 - 1 Right-click the first cell of the grid. Choose Insert→HTML Button.
 - 2 On HTML Button, type "2D with Depth" into the value field.
 - 3 Choose OK. If a warning message appears, choose OK.
- 6 Repeating the process of step 5, create an HTML button in the remaining empty cells of the grid with the values "2D", "resize 400x600", and "resize 600x400".

Task 2: Script the chart sub_type controls

In this task, you add event handler scripts to HTML buttons that change the subtype controls of the chart.

- 1 Select the chart. In the property editor, open Properties→Bookmark. Set the bookmark value to "ChartBookmark" as shown in Figure 5-7.

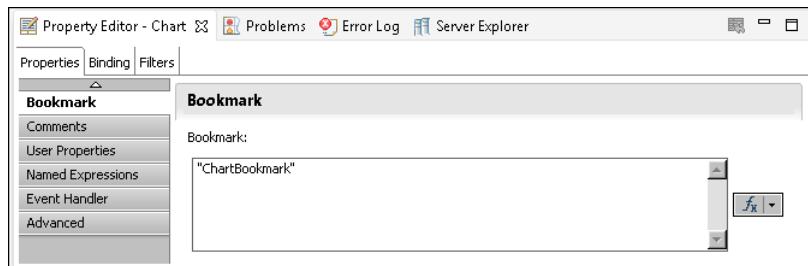


Figure 5-7 Setting the chart bookmark property

- 2 Select the 2D with Depth HTML button and choose Script.
- 3 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 4 After the first curly brace ({), add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
  .getChartByBookmark("ChartBookmark");
bchart.setChartTitle("Orders by Country (2D with Depth)");
bchart.setDimension(actuate.report.Chart
  .CHART_DIMENSION_2D_WITH_DEPTH );
bchart.submit();
```

- 5 Return to the layout editor. Select the 2D HTML button and choose Script.
- 6 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 7 After the first curly brace ({), add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
```

```
        .getChartByBookmark("ChartBookmark");
bchart.setTitle("Orders by Country");
bchart.setDimension(actuate.report.Chart.CHART_DIMENSION_2D);
bchart.submit();
```

Task 3: Script the chart size controls

In this task, you add event handler scripts to HTML buttons that change the display dimensions of the chart.

- 1 Select the Resize 400x600 HTML button and choose Script.
- 2 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 3 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
        .getChartByBookmark("ChartBookmark");
bchart.setTitle("Orders by Country (400x600)");
bchart.setSize(400,600);
bchart.submit();
```

- 4 Return to the layout editor. Select the Resize 600x400 HTML button and choose Script.
- 5 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 6 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
        .getChartByBookmark("ChartBookmark");
bchart.setTitle("Orders by Country (600x400)");
bchart.setSize(600,400);
bchart.submit();
```

Task 4: Test the scripts

In this task, you run the report and test the HTML button scripts.

- 1 Save the report.
- 2 View the report by choosing Run->View Report->In Web Viewer.
- 3 In the Actuate viewer, choose Resize 600x400. The report title changes and the report changes size, as shown in Figure 5-8.
- 4 In the Actuate viewer, choose 2D with Depth. The report title changes and the report subtype changes to 2D with depth, as shown in Figure 5-9.

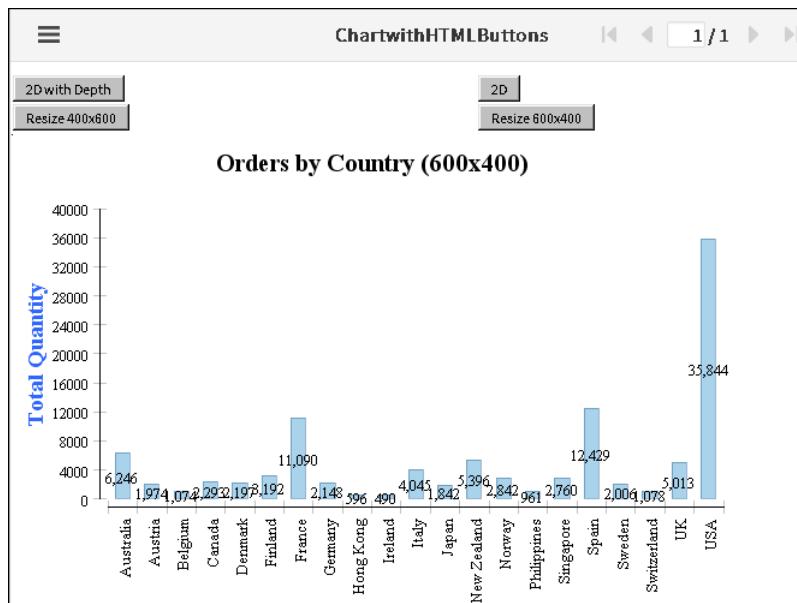


Figure 5-8 A chart displaying 600x400 size

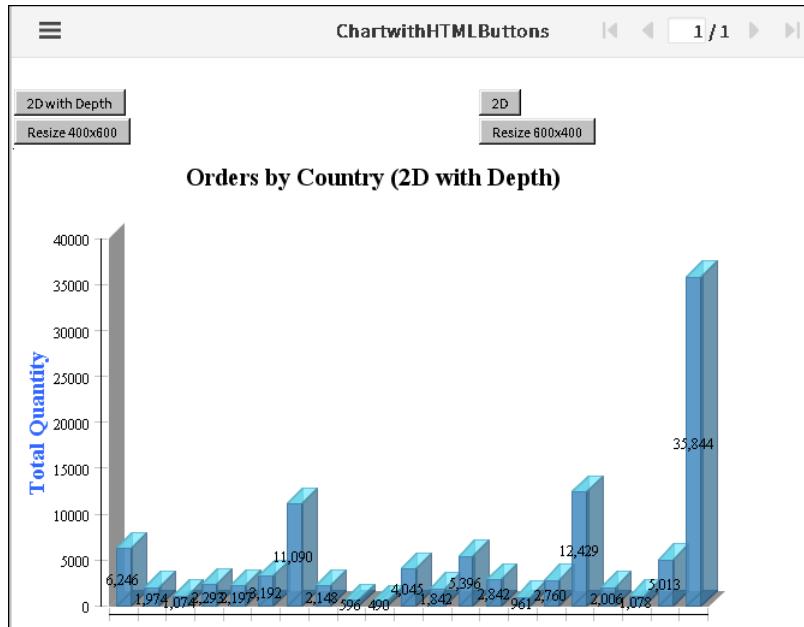


Figure 5-9 A chart with a 2D with Depth subtype

- 5 Choose other buttons to test scripted changes to the report display.

Tutorial 10: Using HTML buttons to apply filters to a chart

In this tutorial, you add multiple HTML buttons to an existing report that each add a filter for a different product line. An additional HTML button removes all filters to display data for all product lines. You perform the following tasks:

- Add a filter button to the report.
- Add HTML buttons for the remaining product lines.
- Add the final HTML button to the report.
- Test the report.

Task 1: Add a filter button to the report

In this task, you preview a report called ButtonFilterChart.rptdesign and add the first HTML button that implements event handlers to apply a filter to the chart.

- 1 In Navigator, open ButtonFilterChart.rptdesign.
- 2 Choose Run>View Report>In Web Viewer to view the report, as shown in Figure 5-10.

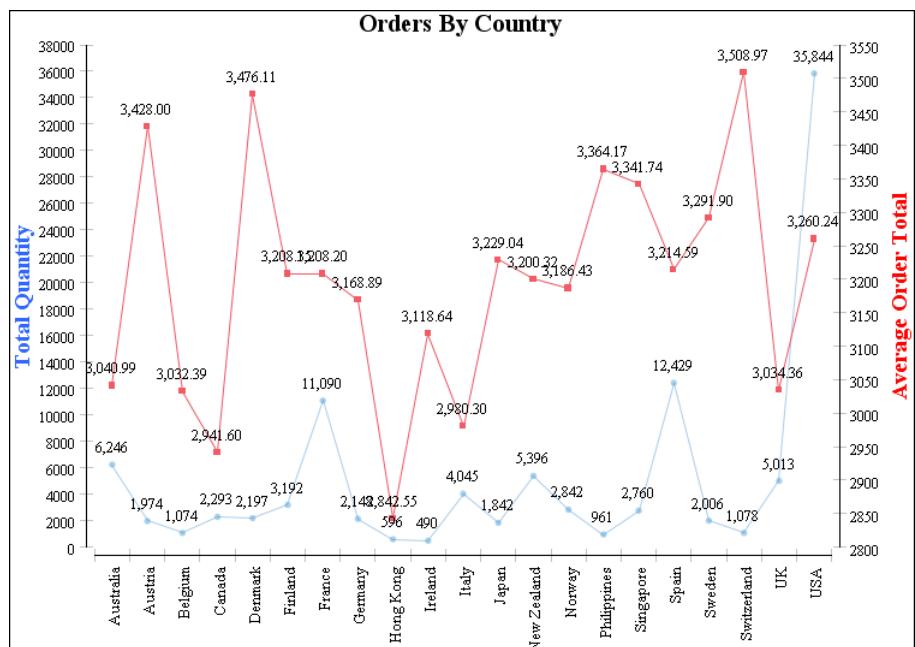


Figure 5-10 Previewing the line chart report

- 3** Close the Web Viewer window. From Palette, drag an HTML button element into the first grid cell, as shown in Figure 5-11.

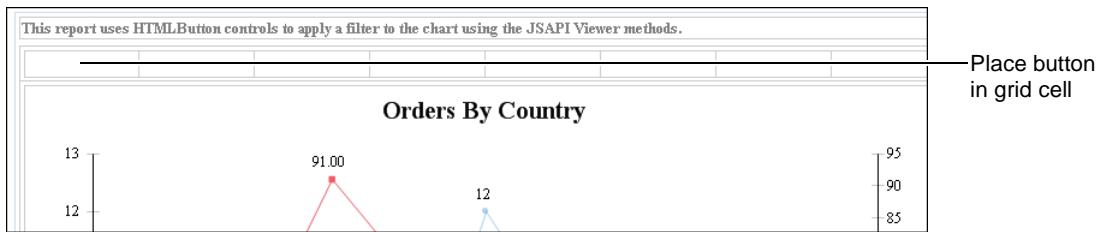


Figure 5-11 Viewing the location of the first HTML button

- 4** In HTML Button, type the following text for Value:
Classic Cars
- 5** Choose OK. If a warning appears displaying a message about adding functionality, choose OK.
- 6** Choose Script, as shown in Figure 5-12, to access the script editor.



Figure 5-12 Choosing Script

- 7** In New Event Function, select onclick.
- 8** In the function body for the onclick event handler, copy the code for the Classic Cars button shown in Listing 5-1.

Listing 5-1 Classic Cars JSAPI code

```
var bviewer = this.getViewer( );
var bpagecontents = bviewer.getCurrentPageContent( );
var bchart = bpagecontents.getChartByBookmark("ChartBookmark");

if (bchart == null) return;// unable to get handle to chart in
// case where chart becomes hidden
var filter = new actuate.data.Filter("PRODUCTLINE",
    actuate.data.Filter.EQ, "Classic Cars");
var filters = new Array( );
filters.push(filter);

bchart.setFilters(filters);
bchart.setChartTitle("Orders By Country (Classic Cars)");
bchart.submit( );
```

- 9** Preview the report in the web viewer by choosing Run->View Report
->In Web Viewer. Click on the Classic Cars HTML button that appears in the top left corner and the filtered chart appears as shown in Figure 5-13.

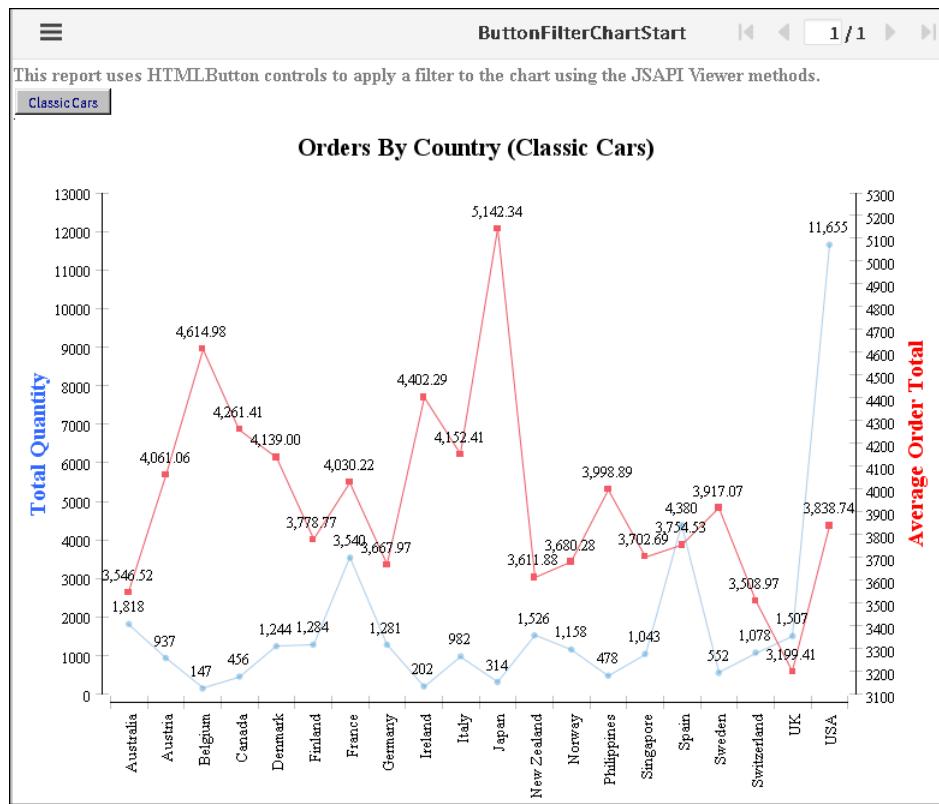


Figure 5-13 Previewing the report with the Classic Cars data

10 Close Actuate Viewer.

Task 2: Add HTML buttons for the remaining product lines

In this task, you add six HTML buttons, one for each of the remaining product lines.

- 1 Choose Layout to return to the layout editor. From Palette, drag an HTML button element into the next available grid cell. In HTML Button, for Value, type:

Motorcycles

Choose OK. If a warning appears displaying a message about adding functionality, choose OK. The HTML button appears in the layout editor.

- 2 Choose the Script tab. In New Event Function, select onclick.

- 3** Copy the code for the Classic Cars button shown in Listing 5-1, and paste the code in the function body of the onclick event handler.
- 4** In Script, replace Classic Cars with Motorcycles in the following two lines:

```
var filter = new actuate.data.Filter("PRODUCTLINE",
    actuate.data.Filter.EQ, "Classic Cars");
```

and:

```
bchart.setChartTitle("Orders By Country (Classic Cars)");
```

The edited event handler appears as shown in Listing 5-2.

Listing 5-2 Motorcycles JSAPI code

```
var bviewer = this.getViewer();
var bpagecontents = bviewer.getCurrentPageContent();
var bchart = bpagecontents.getChartByBookmark("ChartBookmark");

if (bchart == null) return; // unable to get handle to chart in
// case where chart becomes hidden
var filter = new actuate.data.Filter("PRODUCTLINE",
    actuate.data.Filter.EQ, "Motorcycles");
var filters = new Array();
filters.push(filter);

bchart.setFilters(filters);
bchart.setChartTitle("Orders By Country (Motorcycles)");
bchart.submit();
```

- 5** Repeat steps 1 through 4 of this task for the following five buttons and data values:

- Planes
- Ships
- Trains
- Trucks and Buses
- Vintage Cars

When complete, the report layout appears as shown in Figure 5-14.

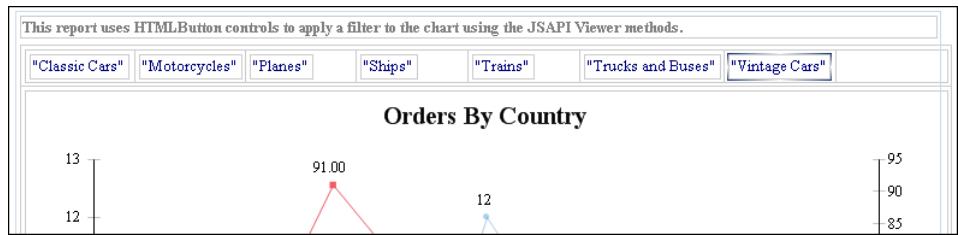


Figure 5-14 Viewing filter buttons in Layout

Task 3: Add the final HTML button to the report

In this task, you add the final filter button to the report. This button is different from the previous buttons, in that it will clear any filter and display summary data for all product lines.

- 1 From Palette, drag an HTML button element into the remaining grid cell. In HTML Button, in Value, type:
Show All
Choose OK.
- 2 Choose Script. In New Event Function, select onclick.
- 3 In the function body for the onclick event handler, copy the code for the Show All button shown in Listing 5-3.

Listing 5-3 JSAPI code to remove filters from charts

```

var bviewer = this.getViewer();
var bpagecontents = bviewer.getCurrentPageContent();
var bchart = bpagecontents.getChartByBookmark("ChartBookmark");

if (bchart == null) return;// unable to get handle to chart in
// case where chart becomes hidden

bchart.clearFilters("PRODUCTLINE");
bchart.setChartTitle("Orders By Country");
bchart.submit();

```

- 4 Choose Layout. The Show All button appears as shown in Figure 5-15.

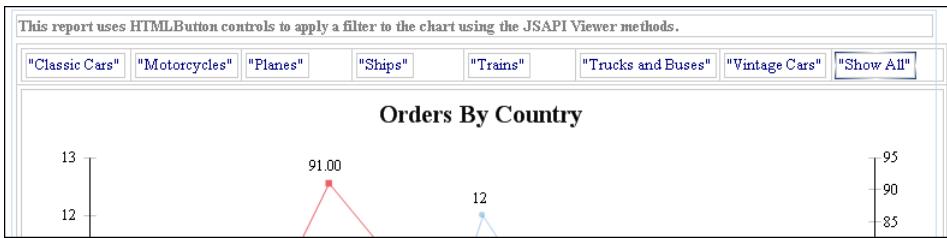


Figure 5-15 Viewing all buttons in Layout

Task 4: Test the report

In this task, you test the report by selecting the various product line buttons.

- 1 Choose Run->View Report->In Web Viewer.
- 2 Choose the Planes HTML button. The chart changes, as shown in Figure 5-16.

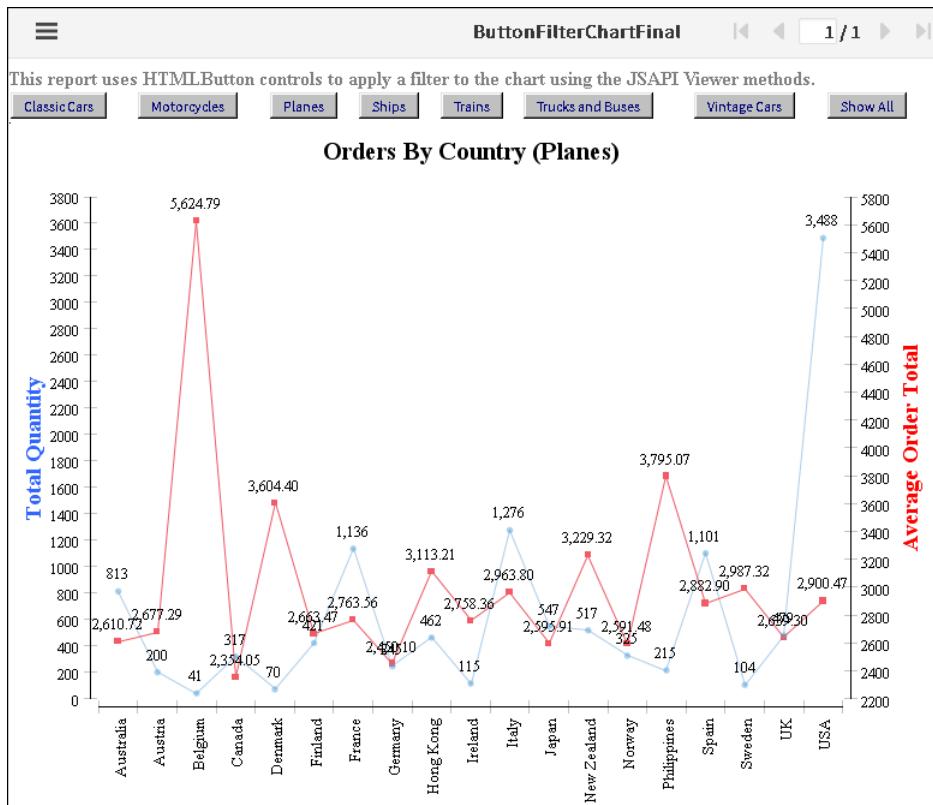


Figure 5-16 Viewing the report after selecting the Planes HTML button

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 5-17.

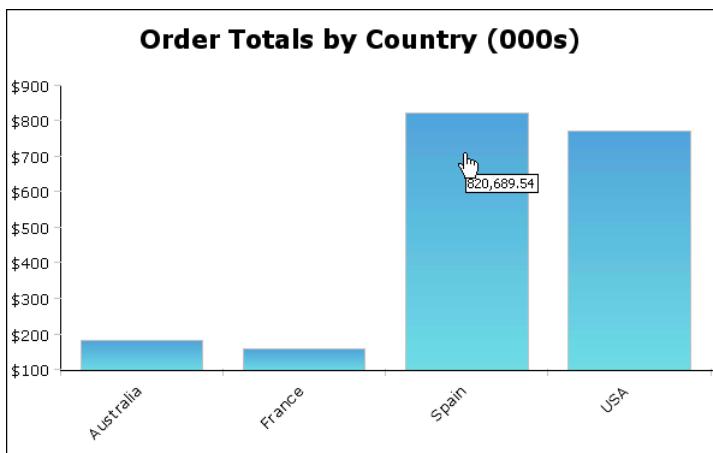


Figure 5-17 Chart showing a tooltip

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 5-18 identifies these elements.

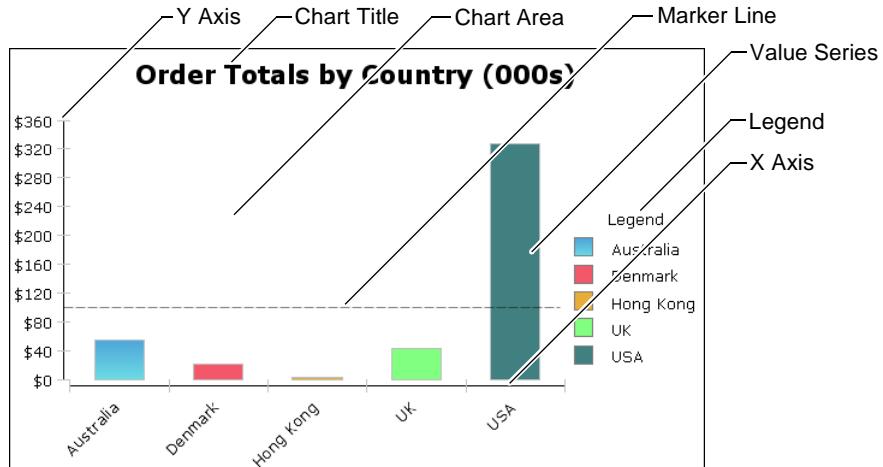


Figure 5-18 Elements selectable for chart interactivity

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 5-19 shows the location of the Interactivity button for a value series.

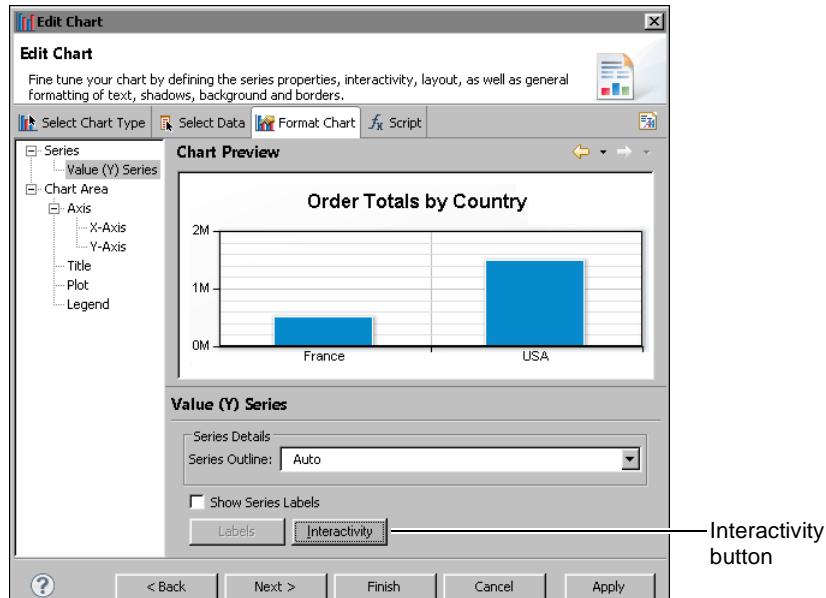


Figure 5-19 Accessing interactivity for a value series

Figure 5-20 shows the elements accessible using the script feature.

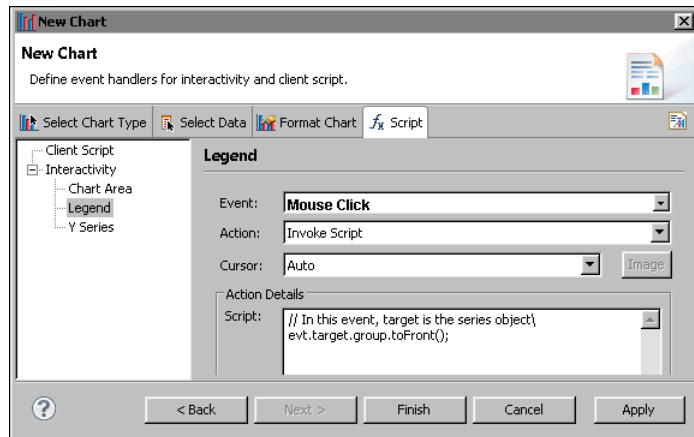


Figure 5-20 Accessing interactivity for a legend

The location of the Interactivity button varies by chart element. Click the Interactivity button to display the interactivity editor. Figure 5-21 shows the interactivity editor.

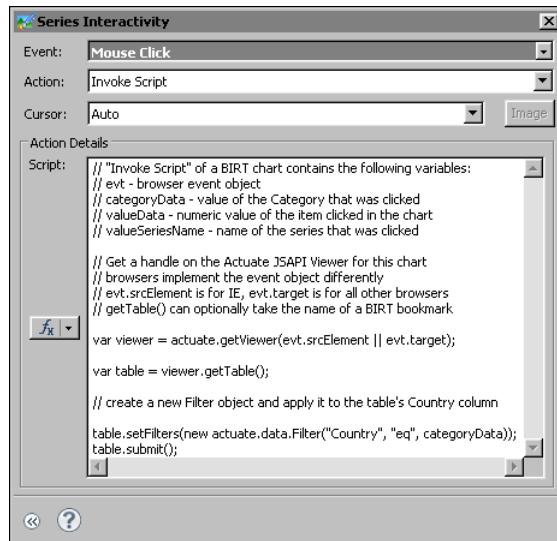


Figure 5-21 Interactivity editor

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

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

```

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

```

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

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

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

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

- Performs an operation on the target:

```

table.setFilters(new actuate.data.Filter("Country",
    actuate.data.Filter.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: actuate.data.Filter.EQ is the constant definition for the equal to operator.
- 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 5-22 shows the chart before interaction.

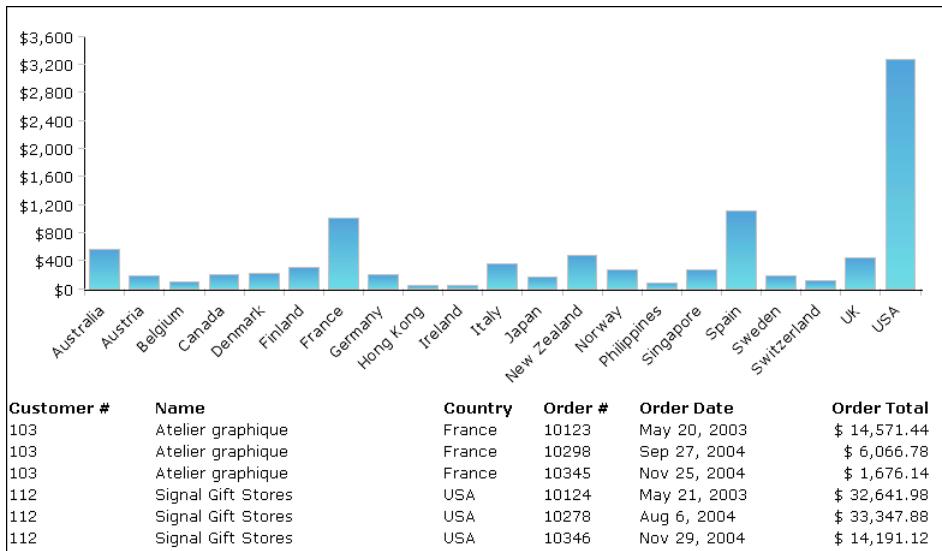


Figure 5-22 An interactive chart and table before any user action

When the user selects the bar for Australia in the value series, the table is filtered for Australia, as shown in Figure 5-23.

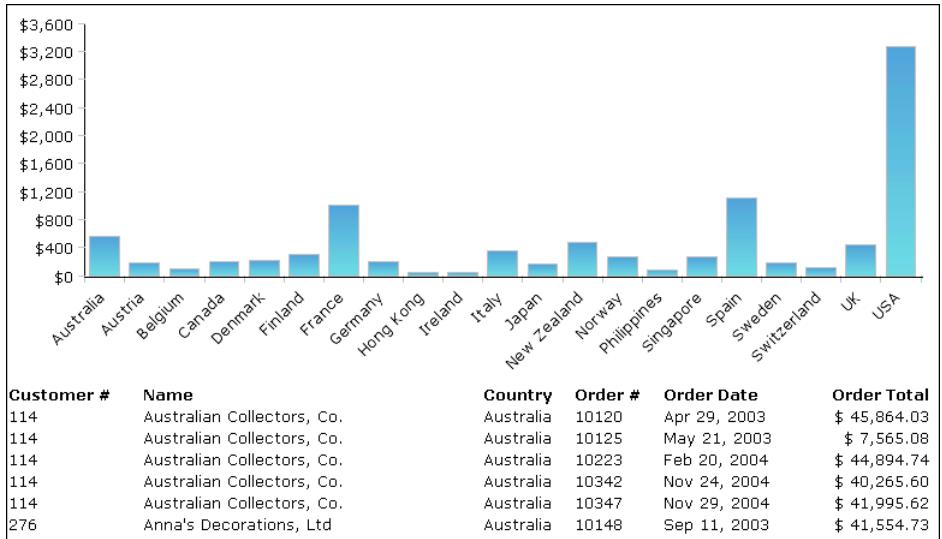


Figure 5-23 An interactive chart and table after the user selects Australia

Tutorial 11: Adding an interactive chart filter to a BIRT report

In this tutorial, you add an interactive chart control to a BIRT report design that implements a filter on the other charts in the report design. You perform the following tasks:

- Add bookmarks.
- Add a filter script to chart interactivity.

Task 1: Add bookmarks

In this task, you preview a report called InteractiveChartandTable.rptdesign and add a bookmark to the chart and table.

- 1 In Navigator, open InteractiveChartandTable.rptdesign.
- 2 Choose Run->View Report->In Web Viewer to view the report, as shown in Figure 5-24.
- 3 Choose Layout to return to the layout editor.

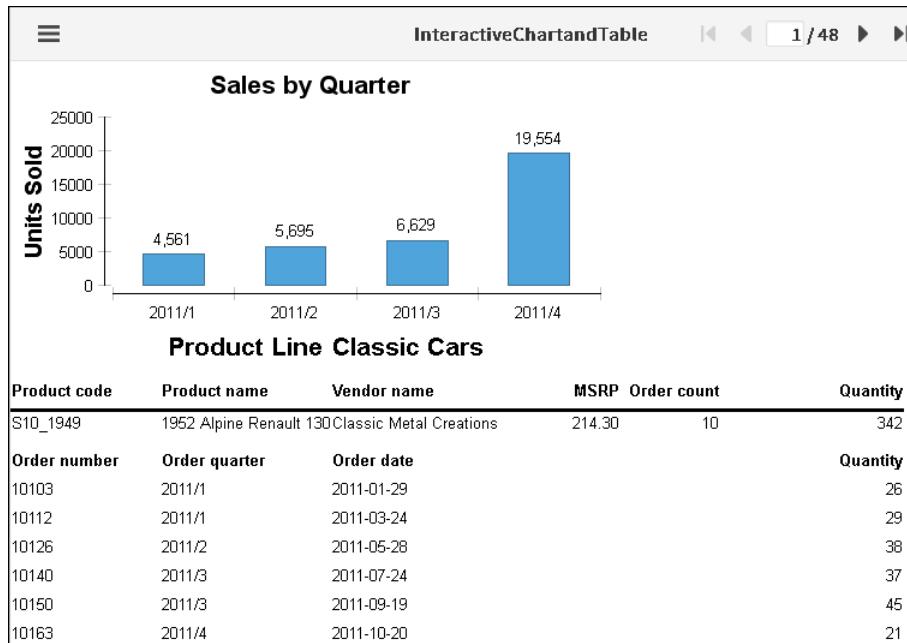


Figure 5-24 Previewing the report

- Select the chart entitled Sales by Quarter. In the property editor, open Properties → Bookmark. Set the bookmark value to "SalesChart", as shown in Figure 5-25.



Figure 5-25 Setting the chart bookmark property

- Repeating the process of step 5, for the table entitled Product Line, set the bookmark value to "ProductTable".

Task 2: Add a filter script to chart interactivity

In this task, you add a filter script to the Sales by Quarter chart to affect the other charts.

- Double-click on the Sales by Quarter chart. In Edit Chart, select Format Chart → Series → Value (Y) Series. Then choose Interactivity.
- On Series Interactivity, select Mouse Click for event, and Invoke Script for action, as shown in Figure 5-26.

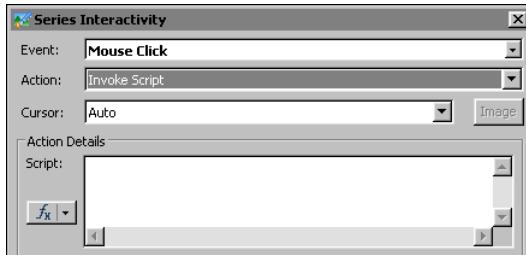


Figure 5-26 Interactivity settings for invoking a script on mouse click

- In the Script text box, add the following code:

```
var atable = actuate.getViewer(evt.srcElement || evt.target)
    .getCurrentPageContent().getTableByBookmark("ProductTable");
atable.setFilters(new actuate.data.Filter("QUANTITYORDERED",
    actuate.data.Filter.GREATER_THAN, valueData/200));
atable.submit();
```

In Edit Chart, choose Finish.

- View the report by choosing Run → View Report → In Web Viewer.

- 5 Select a bar in the table to activate the filter, as shown in Figure 5-27.

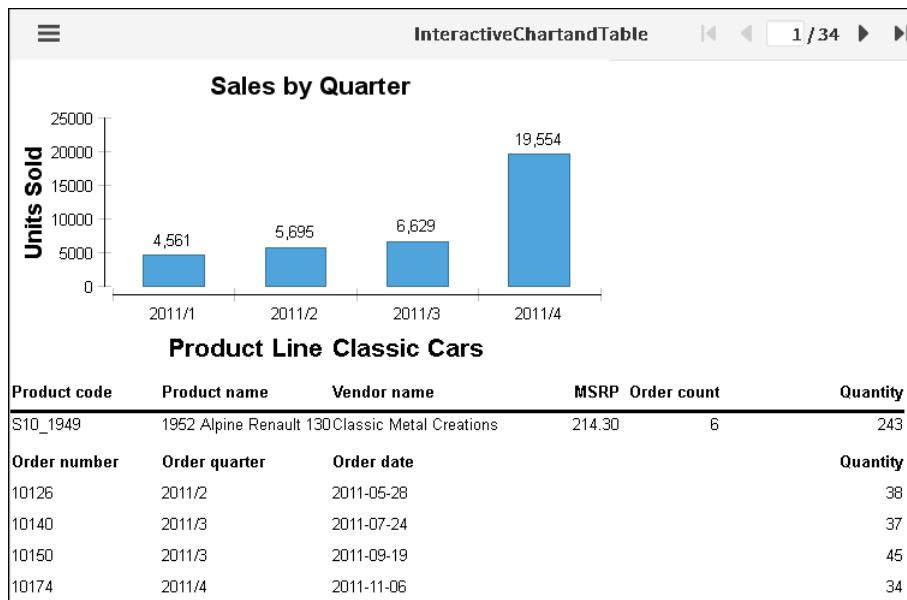


Figure 5-27 Filtered product table after selecting a chart value

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 5-28.

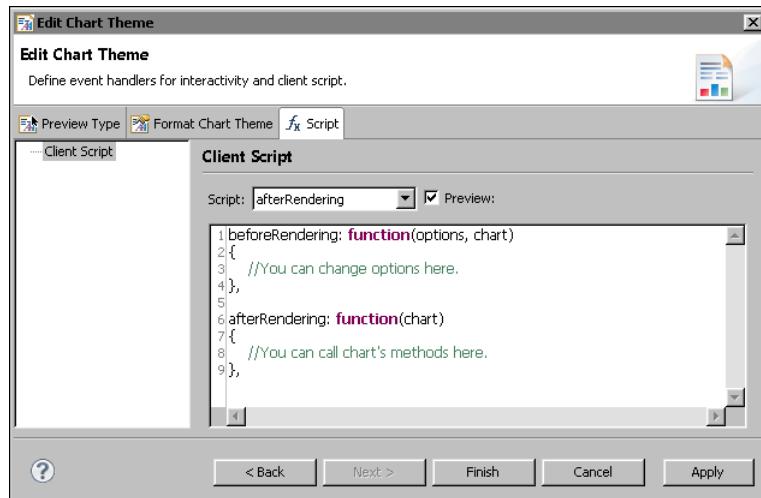


Figure 5-28 Adding script elements in edit chart theme

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';
        }
    }
}

```

Tutorial 12: Adding scripted HTML5 Chart controls to a BIRT design

In this tutorial, you add HTML buttons to a BIRT design that implement controls for an HTML5 chart in the BIRT design. You perform the following tasks:

- Adding HTML buttons
- Scripting the client chart controls
- Scripting the client option controls
- Testing the scripts

Task 1: Adding HTML buttons

In this task, you preview a report called `HTML5ChartWithHTMLButtons.rptdesign` and create a grid of HTML buttons.

- 1 Open BIRT Designer Professional. In Navigator, navigate to and open `HTML5ChartWithHTMLButtons.rptdesign`.
- 2 Preview the report, as shown in Figure 5-29.

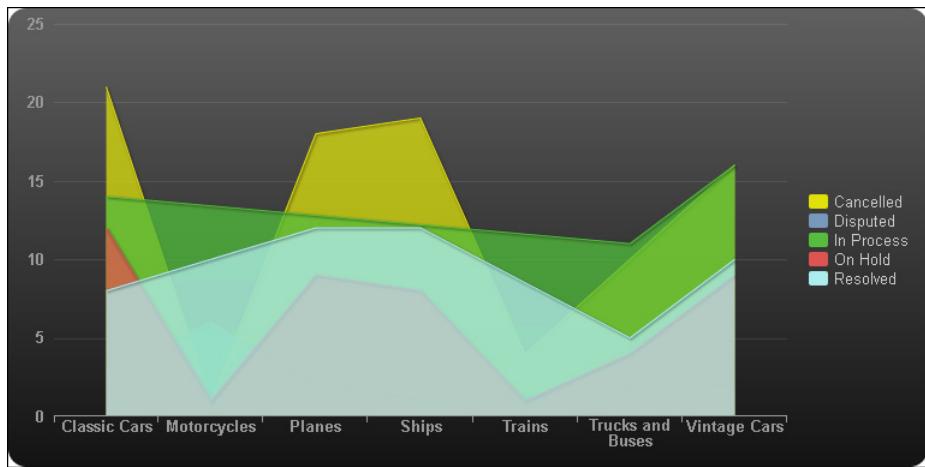


Figure 5-29 Previewing the HTML5 Chart report

- 3 Choose Layout to return to the layout editor.
- 4 Right-click the first cell of the table. Choose Insert->Grid. On Insert Grid, set the Number of columns to 2 and Number of rows to 2, then choose OK. A new grid appears at the top of the table, as shown in Figure 5-30.
- 5 To create HTML buttons for the report, perform the following steps:
 - 1 Right-click the first cell of the grid. Choose Insert->HTML Button.
 - 2 On HTML Button, type "Hide On Hold" into the value field.
 - 3 Choose OK. If a warning message appears, choose OK.
- 6 Repeating the process of step 5, create an HTML button in the remaining empty cells of the grid with the values "Show On Hold", "Line Chart", and "Area Chart".

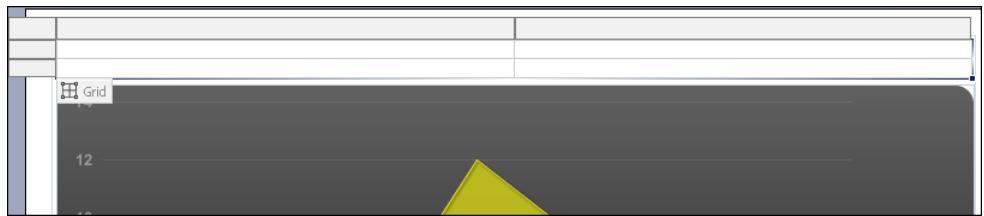


Figure 5-30 Inserting a grid

Task 2: Scripting the client chart controls

In this task, you add event handler scripts to HTML buttons that change the client chart series of the HTML5 chart.

- 1 Select the chart. In the property editor, open Properties→Bookmark. Set the bookmark value to "HTML5ChartBookmark" as shown in Figure 5-31.

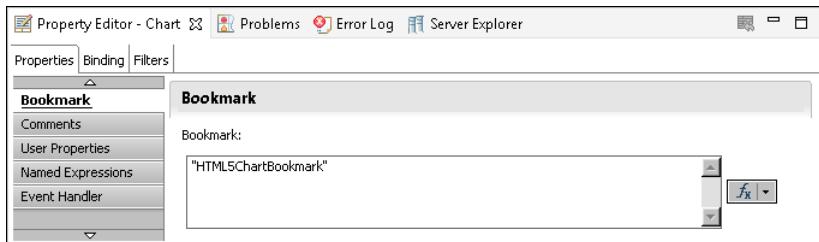


Figure 5-31 Setting the chart bookmark property

- 2 Select the Hide On Hold HTML button and choose Script.
- 3 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 4 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
    .getChartByBookmark("HTML5ChartBookmark");
var clientChart = bchart.getClientChart();
clientChart.setTitle("HTML5 Chart: On Hold series is
    invisible");
clientChart.setSeriesVisible('On Hold', false);
```

- 5 Return to the layout editor. Select the Show On Hold HTML button and choose Script.
- 6 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 7 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
```

```
.getChartByBookmark("HTML5ChartBookmark");
var clientChart = bchart.getClientChart();
clientChart.setTitle("HTML5 Chart: On Hold series is visible");
clientChart.setSeriesVisible('On Hold', true);
```

Task 3: Scripting the client option controls

In this task, you add event handler scripts to HTML buttons that change the chart type using the client options of the HTML5 chart.

- 1 Select the Line Chart HTML button and choose Script.
- 2 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 3 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
    .getChartByBookmark("HTML5ChartBookmark");
var clientChart = bchart.getClientChart();
clientChart.getClientOptions().setChartType('line');
clientChart.getClientOptions().setTitle('Line chart');
clientChart.redraw();
```

- 4 Return to the layout editor. Select the Area Chart HTML button and choose Script.
- 5 In the New Event Function drop-down list, select onclick. The onclick event handler appears in the script editor text box.
- 6 After the first curly brace {}, add the following code:

```
var bchart = this.getViewer().getCurrentPageContent()
    .getChartByBookmark("HTML5ChartBookmark");
var clientChart = bchart.getClientChart();
clientChart.getClientOptions().setChartType('area');
clientChart.getClientOptions().setTitle('Area chart');
clientChart.redraw();
```

Task 4: Testing the scripts

In this task, you run the report and test the HTML button scripts.

- 1 Save the report.
- 2 View the report by choosing Run->View Report->In Web Viewer.
- 3 In the Actuate viewer, choose Line Chart. The chart title changes and the HTML5 chart type changes to line, as shown in Figure 5-32.



Figure 5-32 An HTML5 chart displayed as a line chart

- 4 In the Actuate viewer, choose Area Chart. The chart title changes and the HTML5 chart type changes to area, as shown in Figure 5-33.

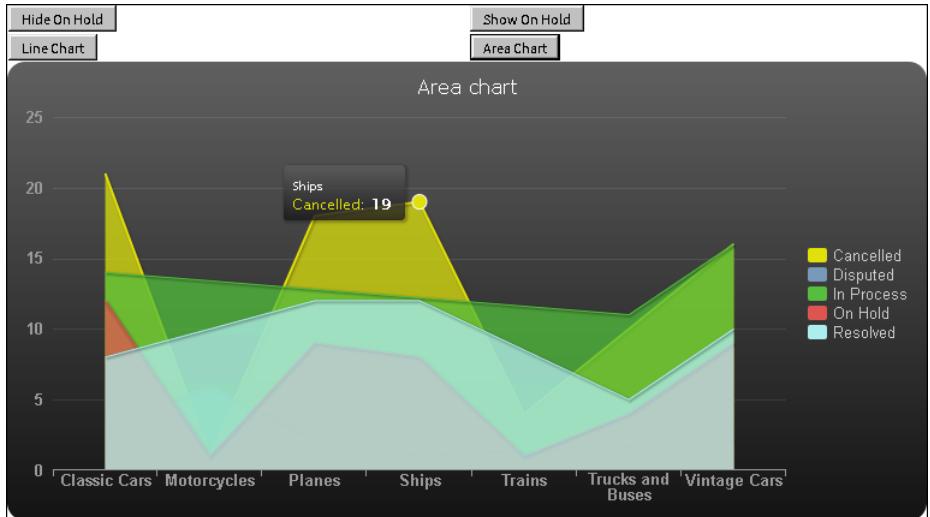


Figure 5-33 An HTML5 chart displayed as an area chart

- 5 Choose other buttons to test scripted changes to the HTML5 chart display.

6

Working with Interactive Crosstabs

This chapter contains the following topics:

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

About cross tabs

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

Figure 6-1 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.

The diagram illustrates a cross tab structure. The row area is labeled "Row area displays state groups" and contains state abbreviations: CA, CT, MA, NH, NJ, NV, NY, and PA. The column area is labeled "Column area displays product line groups" and contains product line names: Classic Cars, Motorcycles, Planes, Ships, Trains, and Grand Total. The data area displays aggregate revenue values. A callout points to the cell for New Hampshire (NH) under Classic Cars, which is highlighted with a blue oval and labeled "The revenue total for Classic Cars for New Hampshire".

	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 6-1 Viewing a cross tab

A cell displays a revenue value by product line and by state, as shown in Figure 6-2.

This figure shows a zoomed-in view of the cross tab from Figure 6-1. It focuses on the cell for New Hampshire (NH) under the "Classic Cars" column. The cell contains the value "\$69,150". An arrow points from this cell to a callout box that reads "The revenue total for Classic Cars for New Hampshire".

	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 6-2 A cell displaying a revenue total

A cross tab uses data from at least three fields. The cross tab in Figure 6-1 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 6-1, 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 6-1, the cross tab displays eight unique values from the state field: CA, CT, MA, NH, NJ, NV, NY, and PA.

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

Tutorial 13: Viewing and pivoting a cross tab

This tutorial provides step-by-step instructions for authoring a web page that displays a cross tab and provides controls to the user. The file in this tutorial that contains a cross tab is Sales by Territory.rptdesign. In this task, you open or create a copy of JSAPITemplate.html and edit its contents to open a cross tab Reportlet and display it in Interactive Crosstabs Viewer.

- 1 Using a code editor, open or create a JSAPITemplate.html file that contains the essential components for any web page that implements the JSAPI.

```
<!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>JSAPI Template</title>
</head>
<body onload="init( )">
    <div id="sample">
        <script type="text/javascript" language="JavaScript"
            src="http://127.0.0.1:8700/iportal/jsapi"></script>
        <script type="text/javascript" language="JavaScript">
            <!-- Insert code here -->
        </script>
    </div>
</body>
```

- 2 Navigate to the following line:

```
<title>JSAPI Template</title>
```

In title, change:

JSAPI Template

to:

CrossTab Analyzer Page

- 3 Navigate to the following line:

```
<div id="sample">
```

In id, change:

sample

to:

analyzer

- 4** Navigate to the empty line after the following line:

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

- 5** Add the following code:

```
var tabviewer;
function init(){
    actuate.load("xtabAnalyzer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        "administrator", "", runAnalyzer);
}
function runAnalyzer(){
    var tabviewer = new actuate.XTabAnalyzer("analyzer");
    tabviewer.setReportName("/Applications/BIRT Sample App
        /Crosstab Sample Revenue.rptdesign");
    tabviewer.setXTabBookmark("SampleRevenue");
    tabviewer.submit();
}
```

- 6** Save the file as interactivecrosstab.html.

- 7** In Internet Explorer, open interactivecrosstab.html.

If you receive a security warning that Internet Explorer has restricted this page from running scripts or ActiveX controls that could access your computer, right-click on the message and select Allow Blocked Content.

About cubes

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

- Measures

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

- Dimensions

Dimensions are groups, such as customers, product lines, or time periods, which aggregate measures. For example, a sales revenue cube contains data

that enables viewing sales volume and revenues, both of which are measures, by customers, product lines, and time periods, all of which are dimensions.

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

Use Actuate BIRT Designer Professional to create a cube using data from one or more data sources, then create a cross tab that uses the cube data and specifies the cross tab appearance. The initial cross tab that appears in Interactive Crosstabs typically displays a portion of the available cube data in a simple layout.

Figure 6-3 shows a cross tab and all of the cube measures and dimensions that are available for analysis.

The screenshot shows the Actuate BIRT Designer Professional interface. On the left, there is a tree view of 'Available cube measures and dimensions'. Under 'Measures', 'Revenue' is expanded, showing 'revenue' and 'amount'. Under 'Customer', 'CustomerCount' is listed. Under 'Dimensions', 'Product' (expanded) includes 'PRODUCTLINE' and 'PRODUCTCODE'; 'SalesDate' (expanded) includes 'Year', 'Quarter', and 'Month'; 'Region' (expanded) includes 'COUNTRY', 'STATE', and 'CITY'. In the center, there is a 'Cross tab' editor. At the top of the editor, there are dropdown menus for 'Rows' (set to 'Year'), 'Columns' (set to 'PRODUCTLINE'), and 'Measures' (set to 'revenue'). Below these are sections for 'Filters' and 'Grid/Chart' tabs. The main area is a grid table with columns for Year, Quarter, Month, and various Product categories. The grid shows revenue data for each category across the specified time periods. The bottom right corner of the grid contains the total value '\$4,300,603'.

Year	Quarter	Month	PRODUCTLINE								Grand Total
			Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Revenue	
Year/QTR /Month 2012	1	1	\$109,562	\$39,987	\$31,159	\$26,310	\$6,387		\$42,909	\$256,315	
		2	\$108,232	\$45,694	\$34,000	\$24,894	\$4,763	\$35,749	\$48,688	\$302,021	
		3	\$99,512			\$15,559	\$9,879	\$32,193	\$45,252	\$202,395	
		4	\$89,998	\$32,229	\$33,882	\$10,808			\$33,352	\$200,269	
	5	\$70,698	\$47,873	\$35,898	\$3,440	\$4,862	\$31,729	\$38,536	\$233,036		
	6	\$46,025			\$16,472		\$41,967	\$48,110	\$152,574		
	7	\$139,040	\$65,156	\$43,256	\$20,260	\$8,985	\$36,967	\$72,418	\$386,082		
	8	\$140,458	\$55,640	\$32,083	\$23,485	\$7,132	\$32,147	\$65,019	\$355,964		
	9	\$140,177	\$6,515	\$30,634	\$23,114	\$5,611	\$37,720	\$62,984	\$306,735		
	10	\$210,010	\$69,147	\$31,081	\$40,881	\$13,781	\$68,620	\$107,121	\$540,642		
	11	\$397,834	\$121,934	\$97,607	\$43,535	\$12,148	\$78,998	\$183,657	\$935,713		
	12	\$131,433	\$43,069	\$68,654	\$43,838	\$13,350	\$52,612	\$75,882	\$428,838		
Total		\$1,682,980	\$527,244	\$438,255	\$292,595	\$86,897	\$448,703	\$823,928	\$4,300,603		
Grand Total		\$1,682,980	\$527,244	\$438,255	\$292,595	\$86,897	\$448,703	\$823,928	\$4,300,603		

Figure 6-3 Interactive Crosstabs displaying a cross tab and available measures and dimensions

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

Handling Interactive Crosstabs viewer events

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

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

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

The XTabAnalyzer class supports the following events:

- ON_CONTENT_CHANGED
- ON_CONTENT_SELECTED
- ON_EXCEPTION
- ON_SESSION_TIMEOUT

To remove an event handler, call removeEventHandler().

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

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

Working with dimensions, measures, and levels

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

Adding a dimension with levels

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

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

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

Removing a dimension

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

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

Adding and removing measures

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

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

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

```
crosstab.submit( );
```

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

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

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

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

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

The web page must contain elements with the IDs of measureName and
measureExpression. Use the following HTML code to create these
elements:
<INPUT TYPE="text" SIZE="60" ID="measureName" VALUE="Quarter
Rate">
<INPUT type="text" SIZE="60" ID="measureExpression"
VALUE="[revenue]/[revenue_SalesDate/year_Product/PRODUCTLINE]">
Use removeMeasure() to remove a measure. Pass the name of the
measure to remove to removeMeasure().
crosstab.removeMeasure("Quarter Rate");
crosstab.submit();
```

Changing measures and dimensions

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

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

measure.setMeasureName("measureName");
```

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

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

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

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

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

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

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

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

Working with totals

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

A total requires a measure and an aggregation function. To add a grand total to a measure, use the actuate.xtabanalyzer.GrandTotal class. Subtotals are added with the actuate.xtabanalyzer.SubTotal class. Both classes use the actuate.xtabanalyzer.Total class. The Total class supports creating aggregated

values on a measure, calculated on either a row or a column. This example creates a total and places the SUM aggregation function on the measure located at measure index 0:

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

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

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

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

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

Sorting and filtering cross tab data

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

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

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

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

var filter = new actuate.xtabanalyzer.Filter
```

```

        ("levelName",actuate.xtabanalyzer.Filter.BETWEEN);
// Filter between the values of 1000 and 2000.
var filterValue = "1000;2000";
filter.setValues(filterValue.split(";"));
crosstab.setFilters(filter);
crosstab.submit( );
To remove a filter from a level, use
    actuate.xtabanalyzer.Crosstab.clearFilters( );
crosstab.clearFilters("levelName");
crosstab.submit( );

```

Drilling down within a cross tab

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

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

To drill up or down, use actuate.xtabanalyzer.Crosstab.drill() with the actuate.xtabanalyzer.Driller and actuate.xtabanalyzer.MemberValue classes. In this example, a cross tab has a dimension named Region with three levels: Country, State, and City. The actuate.xtabanalyzer.Driller object updates the cross tab to display the requested information, as shown in the following code:

```

driller.setAxisType(
    actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
var levelName1 = "Region/Country";
var levelValue1 = "Australia";
var levelName2 = "Region/State";
var levelValue2 = "NSW";
// Create member value objects, and place them in the driller.
var memberValue1 = new
    actuate.xtabanalyzer.MemberValue(levelName1);
memberValue1.setValue(levelValue1);
var memberValue2 = new
    actuate.xtabanalyzer.MemberValue(levelName2);
memberValue2.setValue(levelValue2);
memberValue1.addMember(memberValue2);
driller.addMember(memberValue1);

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

```

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

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

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

Controlling the Interactive Crosstabs viewer user interface

Show or hide Interactive Crosstabs viewer features with the `actuate.xtabalyzer.UIOptions` class. The `UIOptions` class includes functions that support the ability to hide or show different features of the viewer. Figure 6-4 shows what functions affect the viewer display.

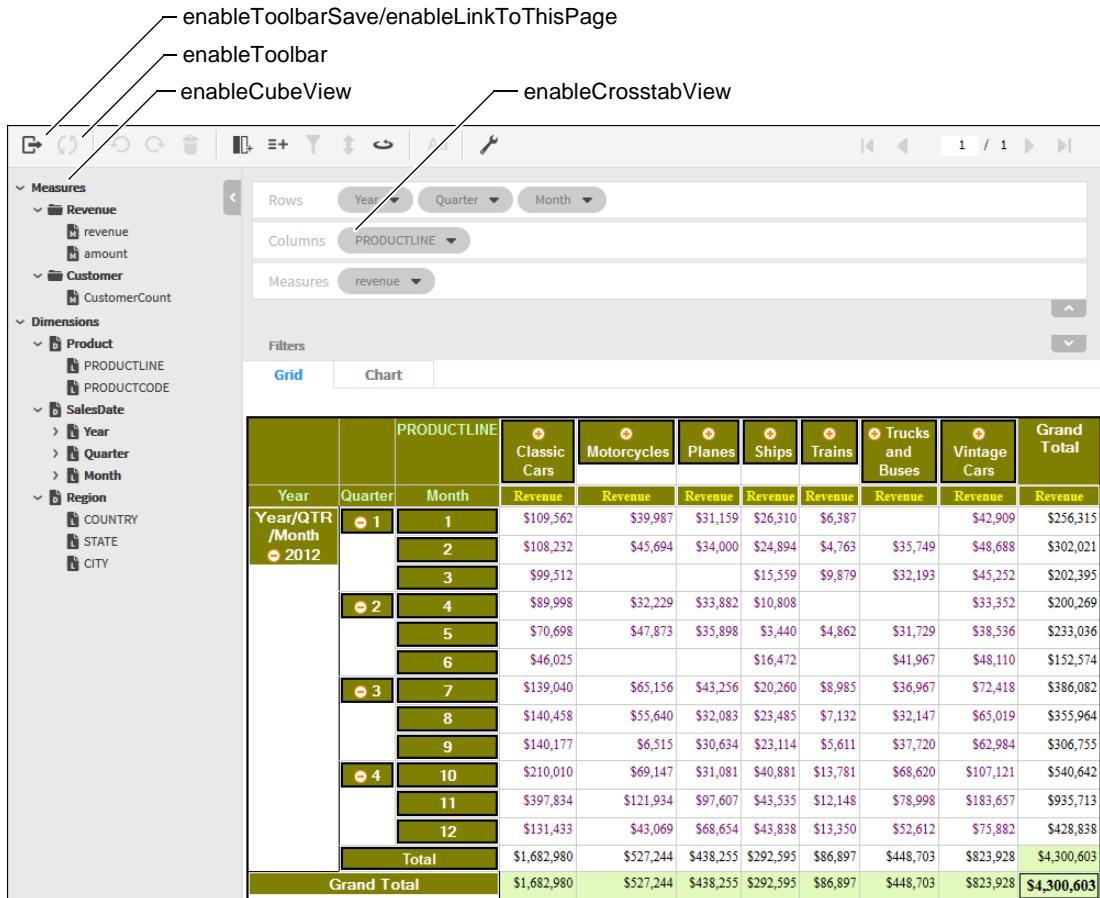


Figure 6-4 Interactive Crosstabs viewer showing areas altered with `UIOptions`

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 );
```

This code produces a viewer similar to Figure 6-5.

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

Figure 6-5 Interactive Crosstabs viewer with settable UIOptions off

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

For example, the cross tab in Figure 6-5 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 6-6.

		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 6-6 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( );
```

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 iHub Visualization Platform client and Actuate Java Components in the following file:

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

About the actuate namespace

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

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

```
actuate
```

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

Using the Actuate library

The Actuate JavaScript library is available from any iHub Visualization Platform client installation or Actuate Deployment Kit for BIRT reports. The URL for the library is:

```
http://127.0.0.1:8700/iportal/jsapi
```

- 127.0.0.1:8700 is the host name and TCP port for an available Actuate web application host.
- /iportal is the context root for the web application.
- /jsapi is the default location of the Actuate JavaScript API libraries.

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

```
<script type="text/javascript" src="http://127.0.0.1:8700  
    /iportal/jsapi">  
</script>
```

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

Actuate JavaScript API classes quick reference

Table 7-1 lists the Actuate JavaScript API classes.

Table 7-1 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.Filter	Conditions to filter data
actuate.data.ReportContent	Represents downloaded content
actuate.data.Request	Represents and processes a request for report data
actuate.data.ResultSet	Results retrieved from a report document in response to a request
actuate.data.Sorter	Sort conditions to sort data
actuate.DataService	Data services to retrieve data from a report document
actuate.Exception	Exception object passed to a callback function or exception handler
actuate.Parameter	Parameters from a report
actuate.parameter.Constants	Global navigation and layout constants used for the Parameter class
actuate.parameter.ConvertUtility	Converts parameters into specific and generic formats
actuate.parameter.EventConstants	Defines the events for parameters this API library supports
actuate.parameter.NameValuePair	Display name and the associated value

(continues)

Table 7-1 Actuate JavaScript API classes (continued)

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

Table 7-1 Actuate JavaScript API classes (continued)

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

Function summary

Table 7-2 lists actuate functions.

Table 7-2 actuate functions

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

authenticate

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

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

Parameters

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

requestOptions

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

userid

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

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 volume name

actuate

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);
```

getDefaultIportalUrl

Syntax String getDefaultIportalUrl()

Returns the default service URL.

Returns String. The default service URL.

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

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

getDefaultsRequestOptions

Syntax actuate.RequestOptions getDefaultRequestOptions()

Returns the default request options.

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

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

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

getVersion

Syntax string getVersion()

Returns the Actuate web application version.

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

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

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

getViewer

Syntax	<pre>actuate.Viewer getViewer(string bookmark)</pre> <pre>actuate.Viewer getViewer(htmlelement viewer)</pre>
	Returns a viewer instance containing the given bookmark element. Load the viewer module before calling actuate.getViewer().
Parameters	<p>bookmark This string parameter contains the name of the bookmark to retrieve or the name of an HTML <div> element.</p> <p>viewer This parameter is the DOM htmlelement object for the HTML <div> element that contains a viewer.</p>
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");
```

initialize

Syntax	<pre>void initialize(string iPortalURL, actuate.RequestOptions requestOptions, reserved, reserved, function callback, function errorCallback)</pre>
	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	<p>iPortalURL String. The target Actuate web application URL.</p> <p>requestOptions actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send in the authentication request, such as the iHub URL, volume, or repository type. It can also add custom parameters to the URL. If requestOptions is null, initialize() uses the default parameter values for the target Actuate web application URL. These default parameter values are defined in Actuate web application's web.xml file. Loading performance is improved if you create a requestOptions object to pass to initialize().</p> <p>reserved Set to null.</p>

actuate

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 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);
```

isConnected

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

Returns whether a given Actuate web application URL is connected.

Parameters **iPortalURL**

String. The target Actuate web application URL.

requestOptions

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

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

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

```
if (!actuate.isConnected("http://digits:8700/iportal", null)) {  
    actuate.authenticate("http://digits:8700/iportal", null,  
        myname, mypassword, null, null, null);  
}
```

isInitialized

Syntax boolean isInitialized()

Returns whether the library is already initialized.

Returns Boolean. True if the library is already initialized.

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

```
if (!actuate.isInitialized( )){  
    actuate.initialize("http://myhost:8700/iportal", null, null,  
                      null, init, null);  
}
```

load

Syntax void load(string componentName)

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

- 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 interactive crosstab component, including the actuate.XTabAnalyzer package

Parameter **componentName**

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

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

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

logout

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

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

Parameters **iPortalURL**

String. The target Actuate web application URL.

requestOptions

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

callback

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

errorCallback

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

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

```
if (actuate.isConnected("http://digits: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 7-3 lists `actuate.AuthenticationException` functions.

Table 7-3 `actuate.AuthenticationException` functions

Function	Description
<code>getIportalUrl()</code>	Returns the web service URL
<code>getRequestOptions()</code>	Returns the request options
<code>getUserID()</code>	Returns the user ID

getIportalUrl

Syntax `string AuthenticationException.getIportalUrl()`

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

Returns String.

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

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

getRequestOptions

Syntax `actuate.RequestOptions AuthenticationException.getRequestOptions()`

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

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

actuate.AuthenticationException

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

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

getUserId

Syntax string AuthenticationException.getUserId()

Returns the UserId used in the failed authentication attempt.

Returns String.

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

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

Class actuate.ConnectionException

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

Constructor

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

Function summary

Table 7-4 describes actuate.ConnectionException functions.

Table 7-4 actuate.ConnectionException function

Function	Description
getUrl()	Returns the whole URL

getUrl

Syntax string ConnectionException.getUrl()

Returns the complete URL sent with the connection request.

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

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

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

Class actuate.Dashboard

Description Represents a dashboard object.

Constructor

Syntax actuate.Dashboard(string container)

Constructs a dashboard object.

Parameter **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 7-5 describes actuate.Dashboard functions.

Table 7-5 actuate.Dashboard functions

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

Table 7-5 actuate.Dashboard functions

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

downloadDashboard

Syntax void Dashboard.downloadDashboard(function callback)

Downloads the dashboard metadata definitions.

Parameter **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);
}
```

embedTemplate

Syntax void Dashboard.embedTemplate(boolean isEmbedded)

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

```
actuate.Dashboard
```

Parameter **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);
```

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());
```

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());
```

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());
```

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.");
}
```

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.");
}
```

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.");
}
```

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());
```

registerEventHandler

Syntax void Dashboard.registerEventHandler(string eventName, function handler)

actuate.Dashboard

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);
```

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);
```

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);
```

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( );
```

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.

```
actuate.Dashboard
```

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

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

setActiveTab

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

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

- Parameter** **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");
```

setAutoSaveDelay

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

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

- Parameter** **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);
```

setContainer

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

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

- Parameter** **containerID**

String. The container ID.

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

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

setDashboardName

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

Sets the dashboard name to view.

Parameter **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" );
```

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

Sets the dashboard's startup height.

Parameter **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);
```

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://127.0.0.1:8700/iportal",
        myRequestOptions.setCustomParameters({myParam: "myValue"});
}
```

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.

```
actuate.Dashboard
```

- 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);
```

setTemplate

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

Sets the template path. This function overwrites the template path that is used by iHub Visualization Platform client.

- Parameter** **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");
```

setWidth

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

Sets the dashboard's startup width.

- Parameter** **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);
```

showTabNavigation

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

Shows the tab toolbar.

- Parameter** **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);
```

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.

Parameter **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( );
```

usePersonalDashboard

Syntax void Dashboard.usePersonalDashboard(boolean true|false)

Forces the dashboard framework to use the user's personal dashboard.

Parameter **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 7-6 lists the actuate.dashboard.DashboardDefinition functions.

Table 7-6 actuate.dashboard.DashboardDefinition functions

Function	Description
getDefaultActiveTab()	Returns the name of the default active tab for this dashboard definition
getTabs()	Returns an array of the tabs in this dashboard definition

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( )) ;
```

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 7-7 lists the dashboard event constants.

Table 7-7 Actuate JavaScript API dashboard event constants

Event	Description
ON_EXCEPTION	Event triggered when an exception occurs. The event handler takes an <code>actuate.Exception</code> object as an input parameter.
ON_SESSION_TIMEOUT	Session time-out event. 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. 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 7-8 lists the actuate.dashboard.GadgetScript functions.

Table 7-8 actuate.dashboard.GadgetScript functions

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

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.

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());
```

## 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());
```

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());
```

## 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());
```

## 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 7-9 lists the `actuate.dashboard.Tab` functions.

**Table 7-9** `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 |

### 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());
```

### 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());
```

## 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 7-10 lists the valid filter operators and the number of arguments to pass to the constructor or `setValues()`.

**Table 7-10** 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 7-10** Filter operators (continued)

| Operator    | Description                                   | Number of arguments |
|-------------|-----------------------------------------------|---------------------|
| MATCH       | Matches a pattern                             | 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   | Does not match a pattern                      | 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 7-11 lists actuate.data.Filter functions.

**Table 7-11** actuate.data.Filter functions

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

### getColumnName

**Syntax** `string Filter.getColumnName( )`

Returns the column name.

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

**Example** This example retrieves the name of the column:

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

### getOperator

**Syntax** `string Filter.getOperator( )`

Returns the filter operator.

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

**Example** This example retrieves the name of the filter operator:

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

## getValues

**Syntax** string Filter.getValues( )  
string[ ] Filter.getValues( )

Returns the evaluated results of this filter. When the filter is constructed or set with a single argument, the returned value corresponds to the single argument. When two arguments or an array are set in the filter, the return value is an array of values.

**Returns** String or array of strings. Returns the value or values from the filter.

**Example** This example retrieves the name of the filter operator:

```
function retrieveValues(myFilter) {
 var myVals = myFilter.getValues();
 return myVals;
}
```

## setColumnName

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

Sets the name of the column to filter.

**Parameter** **columnName**

String. The column name.

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

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

## setOperator

**Syntax** void Filter.setOperator(string operator)

Sets filter operator. The operator determines the comparison made between the data in the column and the value or values set in the filter.

**Parameter** **operator**

String. The operator can be any valid operator. Table 7-10 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(actuate.data.Filter.BOTTOM_N);
 myfilter.setValues("5");
}
```

## setValues

**Syntax**

```
void Filter.setValues(string value)
void Filter.setValues(string value1, string value2)
void Filter.setValues(string[] values)
```

Sets string values for the filter to compare to the data in the column according to the operator. Table 7-10 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(actuate.data.Filter.BETWEEN);
 myfilter.setValues("10", "35");
}
```

---

## Class actuate.data.ReportContent

**Description** The ReportContent class is a container for downloadable report content.

### Constructor

**Syntax** actuate.data.ReportContent(data)

Constructs a ReportContent object.

**Parameter** **data**

String. Content text.

### Function summary

Table 7-12 describes actuate.data.ReportContent functions.

**Table 7-12** actuate.data.ReportContent function

| Function         | Description                                |
|------------------|--------------------------------------------|
| gettextContent() | Returns the text in the downloaded content |

### gettextContent

**Syntax** string ReportContent.getTextContent( )

Returns the text in the downloaded content.

**Returns** String. The text in the downloaded content.

**Example** To make a callback function that prints back the first line of text from some downloaded content back onto the page, use code similar to the following:

```
function callback(data1){
 var rcontent = data1.ReportContent.getTextContent();
 var contentArray = rcontent.split("\n");
 var items = contentArray.length
 document.write("<P>\n")
 document.write(listItems.arguments[0] + "\n</P>")
}
```

---

## Class actuate.data.Request

**Description** Specifies a request for retrieving data and the conditions for that request. This class provides the scope for a request by defining a target element and a range of rows. The scope of the request determines what goes into a ResultSet. Functions that use request can only retrieve ResultSets from report elements that have an explicit bookmark.

### Constructor

**Syntax** `actuate.data.Request(string bookmark, integer startRow, integer maxRow)`

Constructs a request object that other classes use to retrieve data.

**Parameters**

- bookmark** String. A bookmark that identifies an Actuate report element. The `actuate.data.Request` object uses the bookmark to identify the report element to request information from. If null, Request uses the first bookmark. Functions that use request can only retrieve `actuate.data.ResultSet` objects from report elements that have an explicit bookmark.

**startRow**

Integer. The numerical index of the requested first row. The smallest value is 0.

**maxRow**

Integer. The numerical index of the requested last row. 0 indicates no limit.

### Function summary

Table 7-13 lists `actuate.data.Request` functions.

**Table 7-13** `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 7-13** actuate.data.Request functions (continued)

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

## getBookmark

**Syntax** string Request.getBookmarkName( )

Returns the bookmark name for this request.

**Returns** String. The bookmark used in the request object's constructor.

**Example** This example retrieves the bookmark set in the myRequest object:

```
return myRequest.getBookmarkName();
```

## getColumns

**Syntax** string[ ] Request.getColumns( )

Returns a list of column names that match the request.

**Returns** Array of strings. The column names.

**Example** This example retrieves the first, third, and fifth column names from the request object myRequest:

```
function get135Columns (myRequest) {
 var columns = myRequest.getColumns();
 return columns [0];
 return columns [2];
 return columns [4];
}
```

## getFilters

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

Returns filters set for this request.

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

## getMaxRows

**Syntax** integer Request.getMaxRows( )

Returns the maximum number of rows to retrieve.

**Returns** Integer. The index of the last row in the request. 0 means no limit.

## getSorters

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

Returns sorters assigned to this request.

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

## getStartRow

**Syntax** Integer Request.getStartRow( )

Returns the index of the starting row as an integer.

**Returns** Integer. The startRow value. The first row in a column has an index of 0.

## setBookmark

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

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

**Parameter** **bookmark**

String. A bookmark.

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

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

## setColumns

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

Sets the request column names.

**Parameter** **columns**

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

## setFilters

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

## actuate.data.Request

Adds filters to a request. Filters further refine the set of data provided by a request. Using setFilter removes the previous filters from the request object. All of the filters set in a request are applied when the request is used.

**Parameter** **filters**

An array of actuate.data.Filter objects or a single actuate.data.Filter object to refine the request. Use an array for this argument, even if there is only one value.

## setMaxRows

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

Sets the maximum number of rows to retrieve.

**Parameter** **maxrow**

Integer. The numerical value of the index for the last row to request. 0 indicates no limit.

**Example** This example sets the index of the last row for the myRequest request object to 50:

```
myRequest.setMaxRows(50);
```

## setSorters

**Syntax** void Request.setSorts(actuate.data.Sorter[ ] sorters)

Adds sorters to a request to sort the set of data that a request provides. Sorting the data increases the effectiveness of requests by providing the data in a relevant order. Using setSorters removes the previous sorter objects from the request object. All of the sorters set in a request are applied when the request is used.

Sorters are applied in the order that they occur in the array. For example, if the first sorter specifies sorting on a state column and the second sorter specifies sorting on a city column, the result set is sorted by city within each state.

**Parameter** **sorters**

An array of actuate.data.Sorter objects or a single actuate.data.Sorter object to sort the result of the request. Use an array for this argument, even if there is only one value.

**Example** This example sets the alphaNumericSorterSet array in myRequest:

```
myRequest.setSorters(alphaNumericSorterSet);
```

## setStartRow

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

Sets the requested first row.

**Parameter** **startrow**

Integer. The numerical value of the index for the first row to request. The first row in a column has an index of 0.

**Example** This example sets the index of the first row for the myRequest request object to 10:

```
myRequest.setStartRow(10);
```

---

## Class actuate.data.ResultSet

**Description** The actuate.data.ResultSet class represents the data retrieved from a report document. The functions in the actuate.data.ResultSet class access the data by row. The actuate.data.ResultSet class keeps an internal reference to the current row and increments the current row with next( ).

### Constructor

There is no public constructor for actuate.data.ResultSet. The actuate.DataService.downloadResultSet and actuate.Viewer.downloadResultSet functions instantiate the ResultSet object. Set the reference to the ResultSet object in the callback function. For example, when the result set is used as the input parameter for the callback function, result becomes the label for the ResultSet, as shown below:

```
viewer.downloadResultSet(request, parseRS)
function parseRS(resultset){
 // do something with resultset
}
```

### Function summary

Table 7-14 lists actuate.data.ResultSet functions.

**Table 7-14** actuate.data.ResultSet functions

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

### getColumnNameS

**Syntax** string[ ] Request.getColumnNames( )

Returns a list of column names.

**Returns** Array of strings. The column names.

**Example** This example retrieves the first, third, and fifth column names from the ResultSet object myResult:

```
function get135Columns (myResult) {
 var columns = myResult.getColumns();
 return columns[0];
 return columns[2];
```

```

 return columns[4];
 }
}

```

## getValue

**Syntax** string ResultSet.getValue(integer columnIndex)

Returns the value of the specified column from the current row. Specify the column by its numerical index. Use next( ) before using getValue( ) to set the cursor to the first record.

**Parameter** **columnIndex**

Integer. The numerical index of the column from which to retrieve data.

**Returns** String. The field value.

**Example** This example returns the value for the column with an index value of 4 from the current row in the ResultSet object myResult:

```
return myResult.getValue(4);
```

## next

**Syntax** boolean next( )

Increments the current row for the ResultSet. When no current row is set, next( ) sets the current row to the first row in the ResultSet. When no next row exists, next( ) returns false.

**Returns** Boolean. True indicates a successful row increment. False indicates that there are no further rows.

**Example** This example returns the value for the column with an index value of 4 from all of the rows in the ResultSet object myResult:

```

function getColumn4Rows(myResult) {
 var nextrow = myResult.next();
 while (nextrow) {
 return myResult.getValue(4);
 nextrow = myResult.next();
 }
}

```

---

## Class actuate.data.Sorter

**Description** Specifies the conditions for sorting data as it is returned by a request or stored temporarily in a local ResultSet object. The sort arranges rows based on the value of a specified column.

### Constructor

**Syntax** actuate.data.Sorter(string columnName, boolean ascending)

Constructs a sorter object.

**Parameters** **columnName**  
String. The name of the column to sort.

**ascending**

Boolean. True sets sorting to ascending. False sets sorting to descending.

### Function summary

Table 7-15 lists actuate.data.Sorter functions.

**Table 7-15** actuate.data.Sorter functions

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

### getColumnName

**Syntax** string Sorter.getColumnName( )

Returns the name of the column to sort on.

**Returns** String. The column name.

**Example** This example displays an alert box that contains the column name currently being sorted on:

```
function showMyColumnName (mySorter) {
 var sortColName = mySorter.getColumnName();
 alert(sortColName);
}
```

## isAscending

**Syntax** boolean Sorter.isAscending( )

Returns true if the current sort order is ascending. Returns false if the current order is descending.

**Returns** Boolean. True indicates ascending. False indicates descending.

**Example** This example checks if the current sort order is ascending. When the current sort order is descending, this code sets the order to ascending:

```
function makeAscending(mySort) {
 if (mySort.isAscending()) {
 return;
 } else {
 mySort.setAscending(true);
 }
}
```

## setAscending

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

Sets the sort order to ascending or descending.

**Parameter** **ascending**

Boolean. True sets the sort order to ascending. False sets the sort order to descending.

**Example** This example checks if the current sort order is descending. When the current sort order is ascending, this code sets the order to descending:

```
function makeAscending(mySort) {
 if (mySort.isAscending()) {
 return;
 } else {
 mySort.setAscending(true);
 }
}
```

## setColumnName

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

Sets the column to sort on.

**Parameter** **columnName**

String. The column name.

**Example** This example makes the current sorter arrange the result set ascending by the Sales column:

```
function makeAscendingOnSales(mySort) {
 mySort.setColumnName("Sales");
 if (mySort.isAscending()) {
 return;
 } else {
 mySort.setAscending(true);
 }
}
```

---

## Class actuate.DataService

**Description** Connects to an Actuate web application service to retrieve data from Actuate BIRT reports as a ResultSet.

### Constructor

**Syntax** `actuate.DataService(string iportalUrl, actuate.RequestOptions requestOptions)`  
Constructs a DataService object.

**Parameters**

|                   |                                                                                                                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>iportalUrl</b> | String. Optional. The URL of an Actuate web application service. The DataService uses the web application service set in <code>actuate.initialize</code> if one is not specified. |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                       |                                                                                                                                                                                                                              |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>requestOptions</b> | <code>actuate.RequestOptions</code> object. Optional. Specifies the request options for the iportal web service connection. The DataService uses the options set in <code>actuate.initialize</code> if one is not specified. |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Function summary

Table 7-16 lists `actuate.DataService` functions.

**Table 7-16** `actuate.DataService` functions

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

### downloadResultSet

**Syntax** `void DataService.downloadResultSet(string datasource, actuate.data.Request request, function callback, function errorCallback)`

Returns data from an Actuate BIRT report document managed by an Actuate web application. The `actuate.data.ResultSet` object that `downloadResultSet()` returns is used by the callback function.

**Parameters**

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| <b>datasource</b> | String. The repository path and name of the file from which to retrieve data. |
|-------------------|-------------------------------------------------------------------------------|

|                |                                                                                 |
|----------------|---------------------------------------------------------------------------------|
| <b>request</b> | <code>actuate.data.Request</code> object. Specifies the request for the report. |
|----------------|---------------------------------------------------------------------------------|

|                 |                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>callback</b> | Function. The callback function to use after the ResultSet finishes downloading. This function must take the returned ResultSet object as an input parameter. |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
actuate.DataService
```

#### **errorCallback**

Function. Optional. The function to call when an error occurs. The possible errors are actuate.Exception objects. The errorCallback( ) function must take an exception as an argument.

- Example** This example retrieves a result set as specified by the myRequest request object, and calls the makeAscendingSales function, which must take a actuate.data.ResultSet object as an input parameter:

```
var myRequest = new actuate.data.Request("Top_5_Customers", 1, 0);
var myDataService =
new actuate.DataService("http://127.0.0.1:8900/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 7-17 lists actuate.Exception functions.

**Table 7-17** actuate.Exception functions

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

### getDescription

**Syntax** string Exception.getDescription( )

Returns exception details as provided by the Server, Client, and User objects.

**Returns** String. A detailed description of the error. Information is provided according to the type of exception generated, as shown below:

- Server error: The SOAP string
- Client error: For the Firefox browser, a list comprised of fileName+number+stack
- Usage error: Any values set in the object generating the exception

```
actuate.Exception
```

**Example** This example displays the server error description in an alert box:

```
alert("Server error: " + Exception.getDescription());
```

## getErrCode

**Syntax** string Exception.getErrCode( )

Returns the error code for server exceptions.

**Returns** String. A server error code.

**Example** This example displays the server error code in an alert box:

```
alert("Server error: " + Exception.getErrCode());
```

## getMessage

**Syntax** string Exception.getMessage( )

Returns a short message about the exception. This message is set for an actuate.Exception object with the actuate.Exception.initJSEception( ) function.

**Returns** String. A server error code.

**Example** This example displays the error's short message code in an alert box:

```
alert("Error Message: " + Exception.getMessage());
```

## getType

**Syntax** string Exception.getType( )

Returns the type of the exception:

- ERR\_CLIENT: Exception type for a client-side error
- ERR\_SERVER: Exception type for a server error
- ERR\_USAGE: Exception type for a Actuate JavaScript API usage error

**Returns** String. A server error code.

**Example** This example displays the error type in an alert box:

```
alert("Error type: " + Exception.getType());
```

## isExceptionType

**Syntax** boolean Exception.isExceptionType(object exceptionType)

Compares the input object to the exception contained in this actuate.Exception object to the exceptionType object argument.

**Parameter** **exceptionType**

Object. Either an Exception object, such as an instance of actuate.Viewer.ViewerException, or the name of an Exception class as a string.

**Returns** Boolean. Returns true if the exception contained in this actuate.Exception object matches the exceptionType object argument.

**Example** To alert the user when the exception e is a usage error, use code similar to the following:

```
if (e.isExceptionType(actuate.exception.ERR_USAGE)) {
 alert('Usage error occurred!');
}
```

---

## Class actuate.Parameter

**Description** The actuate.Parameter class retrieves and displays Actuate BIRT report parameters in an HTML container. Users can interact with the parameters on the page and pass parameter values to an actuate.Viewer object, but not to the server directly.

The actuate.Parameter class displays the parameters by page. The actuate.parameters.navigate( ) function changes the page display or changes the current position on the page.

### Constructor

**Syntax** actuate.Parameter(string container)

Constructs a parameter object for a page, initializing the parameter component.

**Parameter** **container**

String. The name of the HTML element that displays the rendered parameter component or a container object. The constructor initializes the parameter component but does not render it.

### Function summary

Table 7-18 lists actuate.Parameter functions.

**Table 7-18** actuate.Parameter functions

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

**Table 7-18** actuate.Parameter functions

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

## downloadParameters

**Syntax** void Parameter.downloadParameters(function callback)

Retrieves an array of actuate.parameter.ParameterDefinition objects that contain the report parameters for the report and sends the array to the callback function, which must take the array as an input parameter.

**Parameter** **callback**  
Function. The function to execute after the report parameters finish downloading. Parameter.downloadParameters( ) sends an array of actuate.parameter.ParameterDefinition objects to the callback function as an input argument.

**Example** This example retrieves a set of report parameters and sends them to a callback function:

```
function getChartParams (myParameter) {
```

```
actuate.Parameter
```

```
 myParameter.downloadParameters(callback());
}
```

## downloadParameterValues

**Syntax** void Parameter.downloadParameterValues(function callback)

Returns an array of the actuate.parameterParameterValue objects for the parameter object. If no values have been set, the parameter object downloads the default values from the server.

**Parameter** **callback**

Function. The function to execute after the report parameters finish downloading. Parameter.downloadParameterValues( ) sends an array of actuate.parameterParameterValue objects to the callback function as an input argument.

**Example** To download the parameter values and add them to the viewer, the callback function must use the values as an input parameter, as shown in the following code:

```
paramObj.downloadParameterValues(runNext);
function runNext(values){
 viewer.setParameterValues(values);
}
```

## getLayout

**Syntax** string Parameter.getLayout( )

Returns the parameter layout type.

**Returns** String. The parameter layout, which will match one of the layout constants in actuate.parameter.Constants:

- actuate.parameter.Constants.LAYOUT\_NONE
- actuate.parameter.Constants.LAYOUT\_GROUP
- actuate.parameter.Constants.LAYOUT\_COLLAPSIBLE

**Example** This example calls getLayout( ) to display the parameter layout type in an alert box:

```
alert(paramObj.getLayout());
```

## getParameterGroupNames

**Syntax** string[ ] Parameter.getParameterGroupNames( )

Returns all the group names for the parameter page as an array of strings.

**Returns** Array of strings. Each string is a group name.

**Example** This example displays an alert box with the name of the first group for the parameter page:

```
var groupNames = paramObj.getParameterGroupNames();
alert("First Group Name: " + groupNames[0]);
```

## getReportName

**Syntax** string Parameter.getReportName( )

Returns the name of the report file currently referenced by this Parameter object.

**Returns** String. The report file name.

**Example** This example displays an alert box with the name of the report file:

```
alert("Report file: " + paramObj.getReportName());
```

## getTransientDocumentName

**Syntax** string Parameter.getTransientDocumentName( )

Returns the name of the transient document generated by running the report currently referenced by this Parameter object.

**Returns** String.

**Example** This example displays an alert box with the name of the transient document:

```
alert("Transient document: " + paramObj.getTransientDocumentName());
```

## hideNavBar

**Syntax** void Parameter.hideNavBar( )

Hides the navigation bar for the parameter component in the LAYOUT\_GROUP layout.

**Example** This example hides the navigation bar:

```
paramObj.hideNavBar();
alert("Navigation bar is hidden");
```

## hideParameterGroup

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

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

## actuate.Parameter

### Parameter **groupNames**

String or array of strings. Hides any groups listed.

**Example** This example hides the report parameters that belong to the groups that are listed in the myGroups string array:

```
var myGroups = ["Group1", "Group2", "Group3"];
paramObj.hideParameterGroup(myGroups);
alert("Groups are hidden");
```

### hideParameterName

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

Hides report parameters as specified by name.

### Parameter **parameterNames**

String or array of strings.

**Example** This example hides the parameters that are listed in the myParams string array:

```
var myParams = ["Parameter1", "Parameter2", "Parameter3"];
paramObj.hideParameterName(myParams);
alert("Parameters are hidden");
```

### navigate

#### Syntax void Parameter.navigate(string containerId, string navTarget)

Changes the current page of the parameter component. The navTarget determines the new location to display the parameter container.

### Parameters **containerId**

String. The value of the id parameter for the HTML <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);
}
```

### onUnload

#### Syntax void Parameter.onUnload( )

Performs garbage collection for the parameter object and unloads JavaScript variables that are no longer needed by Parameter.

**Example** This example unloads JavaScript variables and displays an alert box:

```
myParameter.onUnload();
alert("JS variables unloaded.");
```

## registerEventHandler

**Syntax** void Parameter.registerEventHandler(actuate.parameter.EventConstants event, function handler)

Registers an event handler to activate for parameter events. This function can assign several handlers to a single event.

**Parameters**

**event**

actuate.parameter.EventConstants. A constant corresponding to a supported event. actuate.Parameter supports the following two events:

- actuate.parameter.EventConstants.ON\_CHANGED
- actuate.parameter.EventConstants.ON\_SELECTION\_CHANGED

**handler**

Function. The function to execute when the event occurs. The handler must take two arguments: the parameter instance that fired the event and an event object specific to the event type.

**Example**

To register an event handler to catch exceptions, call actuate.Parameter.registerEventHandler using the ON\_CHANGED constant after creating the viewer object, as shown in the following example:

```
function initParameter() {
 parameter = new actuate.Parameter("acparameter");
 parameter.registerEventHandler(actuate.parameter.EventConstants
 .ON_CHANGED, errorHandler);
}
```

## removeEventHandler

**Syntax** void Parameter.removeEventHandler(actuate.viewer.EventConstants event, function handler)

Removes an event handler.

**Parameters**

**event**

actuate.parameter.EventConstants. A constant corresponding to a supported event. actuate.Parameter supports the following two events:

- actuate.parameter.EventConstants.ON\_CHANGED

## actuate.Parameter

- actuate.parameter.EventConstants.ON\_SELECTION\_CHANGED

### handler

Function. A handler function registered for the event.

**Example** To remove an event handler, call actuate.Parameter.removeEventHandler with a legal event constant, as shown in the following example:

```
function cleanupParameter(){
 parameter.removeEventHandler(actuate.parameter.EventConstants
 .ON_CHANGED, errorHandler);
}
```

## renderContent

**Syntax** void Parameter.renderContent(actuate.parameter.ParameterDefinition[ ] paramDefs, function callback)

Renders the parameter component to the container.

**Parameters** **paramDefs**  
Array of actuate.parameter.ParameterDefinition objects.

### callback

Function. The function to execute after the rendering is done.

**Example** This example calls renderContent( ) after hiding parameter groups:

```
function showNoGroups (myParameter) {
 myParameter.hideParameterGroup(zipcodes);
 myParameter.renderContent (myParameterArray,
 cleanupParameter(myParameter));
}
```

## setAutoSuggestDelay

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

Sets the auto suggest delay time.

**Parameter** **delay**  
Long. Interpreted as milliseconds.

**Example** This example implements a custom auto suggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest (myParameter) {
 myParameter.setAutoSuggestFetchSize(10);
 myParameter.setAutoSuggestListSize(3);
 myParameter.setAutoSuggestDelay(250);
}
```

## setAutoSuggestFetchSize

**Syntax** void Parameter.setAutoSuggestFetchSize(integer size)

Sets the fetch size of the auto suggestion list. AutoSuggest fetches all suggestions from the server when the fetch size is not set.

**Parameter** **size**

Integer. The number of suggestions to fetch at a time.

**Example**

This example implements a custom auto suggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest (myParameter) {
 myParameter.setAutoSuggestFetchSize(10);
 myParameter.setAutoSuggestListSize(3);
 myParameter.setAutoSuggestDelay(250);
}
```

## setAutoSuggestListSize

**Syntax** void Parameter.setAutoSuggestListSize(integer size)

Sets the length of the auto suggestion list. AutoSuggest shows all of the suggestions from the server when the list length is not set.

**Parameter** **size**

Integer. The number of suggestions to display.

**Example**

This example implements a custom auto suggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest (myParameter) {
 myParameter.setAutoSuggestFetchSize(10);
 myParameter.setAutoSuggestListSize(3);
 myParameter.setAutoSuggestDelay(250);
}
```

## setExpandedGroups

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

Defines a set of groups that are expanded by default.

**Parameter** **groupNames**

Array of strings. The group names to expand by default.

**Example**

This example sets the "Motorcycles", "Trucks", and "Airplanes" groups as expanded by default:

```
var myGroups = new Array["Motorcycles", "Trucks", "Airplanes"];
paramObj.setExpandedGroups(myGroups);
```

## setFont

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

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

**Parameter** **fontStyleString**

String. The name of a font.

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

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

## setGroupContainer

**Syntax** void Parameter.setGroupContainer(string[ ] groupNames, string containerId)

Sets the HTML element container for the provided group. All parameter objects listed in groupNames are assigned to the container.

**Parameters** **groupNames**

Array of strings. The group names to be assigned.

**containerID**

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

**Example** This example assigns the group names in the myGroups string array to the leftpane HTML element:

```
var myGroups = ["Group1", "Group2", "Group3"];
paramObj.setGroupContainer(myGroups, "leftpane");
```

## setLayout

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

Sets the parameter layout.

**Parameter** **layoutName**

String constant. Possible values are:

- actuate.parameter.Constants.LAYOUT\_GROUP
- actuate.parameter.Constants.LAYOUT\_NONE
- actuate.parameter.Constants.LAYOUT\_COLLAPSIBLE

**Example** This example sets the parameter object's layout type to LAYOUT\_COLLAPSIBLE:

```
paramObj.setLayout ("LAYOUT_COLLAPSIBLE");
```

## setReadOnly

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

Sets the parameters to read-only.

**Parameter** **readOnly**

Boolean. True indicates that the parameters are read-only.

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

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

## setReportName

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

Sets the report file from which to get report parameters.

**Parameter** **reportFile**

String. The report file path and name. The report file can be a report design file or a report document file.

**Example** To set the name using an HTML input tag with an ID of Selector, use the following code:

```
myViewer.setReportName(document.getElementById("Selector").value);
```

## setService

**Syntax** void Parameter.setService(string iPortalURL, actuate.RequestOptions requestOptions)

Sets the target service URL to which the Parameter object links. If the service URL is not set, this Parameter object links to the default service URL set on the actuate object.

**Parameters** **iPortalURL**

String. The target Actuate web application URL.

**requestOptions**

actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

**Example** This example sets the URL for the Actuate iPortal web application service:

```
paramObj.setService("http://127.0.0.1:8700
 /iportal", myRequestOptions);
```

## setShowDisplayType

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

Sets whether localized data is shown or not.

**Parameter** **showDisplayType**

Boolean. True indicates that localized data is shown.

**Example** This example hides localized data:

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

## submit

**Syntax** void Parameter.submit(function callback)

Submits requests to the server for the report parameters. When this function is called, an AJAX request is triggered to submit all the operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the parameter container.

**Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

**Example** This example calls submit( ) after hiding localized data:

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

---

## Class actuate.parameter.Constants

**Description** Global constants used for Parameter class. Table 7-19 lists the constants used for the parameter class.

**Table 7-19** 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 dynamic filter parameters, use the array of generic objects as the input parameter for actuate.Viewer.setParameterValues.

### Constructor

**Syntax** actuate.parameter.ConvertUtility(actuate.parameter.ParameterValue[ ] aParamVals)

Constructs a new ConvertUtility object.

**Parameter** **aParamVals**  
Array of actuate.parameter.ParameterValue objects to convert.

### Function summary

Table 7-20 lists actuate.parameter.ConvertUtility functions.

**Table 7-20** actuate.parameter.ConvertUtility functions

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

### convert

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

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

**Parameter** **callback**  
Function. The callback function to call after converting the results. The callback function must take the generic array of objects as an argument.

**Example** This example stores the name-value pair array for myParamValues in a variable called nameValueArray:

```
var nameValueArray = new Array();
var converter = new actuate.ConvertUtility(myParamValues)
```

```
converter.convert(callback);
function callback (values){
 nameValueArray = values;
}
```

## convertDate

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

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

**Parameter** **callback**

Function. An optional function to call when this function completes. The callback function receives an array of actuate.parameter.ParameterValue objects as a parameter.

**Example** This example converts the name-value pair array for myParamValues and stores the results in a variable called nameValueArray:

```
var nameValueArray = new Array();
var converter = new actuate.ConvertUtility(myParamValues)
converter.convertDate(callback);
function callback (values){
 nameValueArray = values;
}
```

## getParameterMap

**Syntax** object ConvertUtility.getParameterMap( )

Returns the parameters as an associative array. This function makes the name of each parameter an object property and sets the value of that property to the associated parameter value.

**Returns** Object.

**Example** This example stores the associative array for myParamValues in a variable called nameValueArray:

```
var paramMap = new Object();
var converter = new actuate.ConvertUtility(myParamValues)
paramMap = converter.getParameterMap();
```

## getParameterValues

**Syntax** actuate.parameter.ParameterValue[ ] ConvertUtility.getParameterValues( )

Returns the array of ParameterValue objects.

**Returns** Array of actuate.parameter.ParameterValue objects.

## actuate.parameter.ConvertUtility

**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 7-21 lists the parameter event constants.

**Table 7-21** Actuate JavaScript API parameter event constants

| Event                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ON_CHANGE_COMPLETED  | <p>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:</p> <ul style="list-style-type: none"> <li>■ actuate.Parameter: parameter component for which the event occurred</li> </ul>                                                                                                                         |
| ON_CHANGED           | <p>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:</p> <ul style="list-style-type: none"> <li>■ actuate.Parameter: parameter component for which the event occurred</li> </ul>                                                                                                                                                                                                                                                                     |
| ON_EXCEPTION         | <p>Event triggered when an exception occurs. The event handler must take an actuate.Exception object as an input argument. The Exception object contains the exception information.</p>                                                                                                                                                                                                                                                                                                                                                                                        |
| ON_SELECTION_CHANGED | <p>Event triggered when a selection change occurs. For example, this event triggers if the value of a parameter list control changes. The event handler must take an actuate.Parameter object as an input argument. This input argument is the parameter component for which the event occurred.</p>                                                                                                                                                                                                                                                                           |
| ON_SESSION_TIMEOUT   | <p>Session time-out event. Whenever a session time-out event occurs and the user tries to perform any operation on parameter component, a prompt dialog will be shown to ask whether the user wants to log in again or not. If the user clicks yes, the ON_SESSION_TIMEOUT event will be fired. If no handler has been registered for this event, a default built-in login dialog will be displayed.</p> <p>The event handler takes the following arguments:</p> <ul style="list-style-type: none"> <li>■ actuate.Parameter: 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 7-22 lists actuate.parameter.NameValuePair functions.

**Table 7-22** actuate.parameter.NameValuePair functions

| Function    | Description                           |
|-------------|---------------------------------------|
| getName( )  | Gets the name for this NameValuePair  |
| getValue( ) | Gets the value for this NameValuePair |
| setName( )  | Sets the name for this NameValuePair  |
| setValue( ) | Sets the value for this NameValuePair |

### getName

**Syntax** string NameValuePair.getName( )

Returns the name for this NameValuePair.

**Returns** String.

**Example** This sample code returns the name component of the myNVPair NameValuePair object:

```
alert("Name component is " + myNVPair.getName());
```

## getValue

**Syntax** `string NameValuePair.getValue( )`

Returns the value for this NameValuePair.

**Returns** String.

**Example** This sample code returns the value component of the myNVPair NameValuePair object:

```
alert("Value component is " + myNVPair.getValue());
```

## setName

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

Sets the name for the NameValuePair.

**Parameter** `name`  
String.

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

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

## setValue

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

Sets the value for the NameValuePair.

**Parameter** `value`  
String.

**Example** This sample code sets the value component of the myNVPair NameValuePair object to "Cleveland":

```
myNVPair.setValue ("Cleveland");
```

---

## Class actuate.parameter.ParameterData

**Description** The ParameterData class is a high-level wrapper for an actuate.parameter.ParameterDefinition object.

### Constructor

|                   |                                                                                                                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>     | string actuate.parameter.ParameterData(string reportName,<br>actuate.parameter.ParameterDefinition pd)                                                                                                                                |
|                   | Constructs a new ParameterData object.                                                                                                                                                                                                |
| <b>Parameters</b> |                                                                                                                                                                                                                                       |
|                   | <p><b>reportName</b><br/>String. The name of the report where the parameter definition originates.</p> <p><b>pd</b><br/>actuate.parameter.ParameterDefinition object. The parameter definition set for this ParameterData object.</p> |

### Function summary

Table 7-23 lists the actuate.parameter.ParameterData functions.

**Table 7-23** actuateparameter.ParameterData functions

| Function                     | Description                                                        |
|------------------------------|--------------------------------------------------------------------|
| get CascadingParentValues( ) | Returns the cascading parent value                                 |
| get ChildData( )             | Returns the child ParameterData object                             |
| get ControlType( )           | Returns the controlType UI value                                   |
| get CurrentValue( )          | Returns the current UI value set by the UI control                 |
| get DefaultValue( )          | Returns the default value for this ParameterData object            |
| get HelpText( )              | Returns the help text for this ParameterData object                |
| get NameValueList( )         | Returns the list of name-value pairs for this ParameterData object |
| get ParameterName( )         | Returns the parameter name for this ParameterData object           |
| get ParentData( )            | Returns the parent ParameterData object                            |
| get PickList( )              | Returns the pick list for the child ParameterData object           |

**Table 7-23** actuateparameter.ParameterData functions

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

## get Cascading Parent Values

**Syntax** `actuate.parameter.ParameterValue[ ]`

```
 ParameterData.get CascadingParentValues(
 actuate.parameter.ParameterValue[] parentValues)
```

Returns the cascading parent value.

**Parameter** `parentValues`

An array of `actuate.parameter.ParameterValue` objects. This array is the one to be populated.

**Returns** An array of `actuate.parameter.ParameterValue` objects. This is the input array populated with the cascading parent values.

**Example** This sample code returns a storage array of `actuate.parameter.ParameterValue` objects representing the cascading parent values:

```
var parentValues = new Array();
parentValues = myParamData.get CascadingParentValues(parentValues);
```

## getChildData

**Syntax** actuate.parameter.ParameterData ParameterData.getChildData( )

Returns the child ParameterData object.

**Returns** actuate.parameter.ParameterData object.

**Example** This example assigns the child ParameterData object to a myChildData variable:

```
var myChildData = myParameterData.getChildData();
```

## getControlType

**Syntax** string ParameterData.getControlType( )

Returns the controlType UI value for this ParameterData object.

**Returns** String. The controlType UI value. Legal controlType UI values are:

- null
- AutoSuggest
- ControlRadioButton
- ControlList
- ControlListAllowNew
- ControlCheckBox

**Example** This sample code displays the controlType UI value for the myParamData object in an alert box:

```
alert(myParamData.getControlType());
```

## getCurrentValue

**Syntax** actuate.parameter.ParameterValue ParameterData.getCurrentValue( )

Returns the current UI value set by the UI control.

**Returns** actuate.parameter.ParameterValue. Returns null when the UI control has not set a value.

**Example** This sample code assigns the current UI value to the myCurrVal variable:

```
var myCurrVal = myParameterData.getCurrentValue();
```

## getDefaultValue

**Syntax** string ParameterData.getDefaultValue( )

Returns the default value for this ParameterData object.

**Returns** String. The default value. Returns null when the default value is null.

**Example** This sample code displays the default value for myParamData in an alert box:

```
alert(myParamData.getDefaultValue());
```

## getHelpText

**Syntax** string ParameterData.getHelpText( )

Returns the help text for this ParameterData object.

**Returns** String. The help text.

**Example** This example displays the help text for the myParamData object in an alert box:

```
alert(myParamData.getHelpText());
```

## getNameValueList

**Syntax** actuate.parameter.NameValuePair[ ] ParameterData.getNameValueList( )

Returns the list of name-value pairs for this ParameterData object.

**Returns** Array of actuate.parameter.NameValuePair objects.

**Example** This example stores the array of NameValuePair objects for the myParamValues object in a variable called myNVList:

```
var myNVList = new Array();
myNVList = myParamValues.getNameValueList();
```

## getParameterName

**Syntax** string ParameterData.getParameterName( )

Returns the parameter name for this ParameterData object.

**Returns** String. The parameter name.

**Example** This sample code displays the parameter name for the myParamData object in an alert box:

```
alert(myParamData.getParameterName());
```

## getParentData

**Syntax** actuate.parameter.ParameterData ParameterData.getParentData( )

Returns the parent ParameterData object.

**Returns** actuate.parameter.ParameterData object.

## actuate.parameter.ParameterData

**Example** This sample code assigns this ParameterData object's parent ParameterData object to the myParentData variable:

```
var myParentData = myParameterData.getParentData();
```

### getPickList

**Syntax** actuate.parameterParameterValue[ ] ParameterData.getPickList(function callback)

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

**Parameter** **callback**

Function. An optional function to call when this function completes. This function receives the following parameters:

- An array of actuate.parameter.NameValuePair objects
- An integer that represents the pick list's total leftover count

**Returns** An array of actuate.parameterParameterValue objects.

**Example** This sample code uses the callback function runNext( ) to display the pick list's total leftover count in an alert box and assigns the array of NameValuePair objects to the pickListNVPairs variable:

```
paramObj.getPickList(runNext);
function runNext(pairs, leftover) {
 alert(leftover);
 var pickListNVPairs = new Array();
 pickListNVPairs = pairs;
}
```

### getPromptText

**Syntax** string ParameterData.getPromptText( )

Returns the prompt text for this ParameterData object.

**Returns** String. The prompt text.

**Example** This sample code displays the prompt text for the myParamData object in an alert box:

```
alert(myParamData.getPromptText());
```

### getSuggestionList

**Syntax** string[ ] ParameterData.getSuggestionList(function callback, string filter)

Returns the filter-based suggestion list for this ParameterData object.

- Parameters**
- callback**  
Function. An optional function to call when this function completes. This function receives an array of actuate.parameter.NameValuePair objects as a parameter.
  - filter**  
String. The filter for the suggestion list.
- Example** This sample code uses the string "Trucks" to call back function runNext( ) to filter the suggestion list and assigns the filtered NameValuePair objects to the mySuggestions variable:

```
paramObj.getSuggestionList(runNext, "Trucks");
function runNext(suggested) {
 var mySuggestions = new Array();
 mySuggestions = suggested;
}
```

## isAdhoc

- Syntax** boolean ParameterData.isAdhoc( )  
 Returns true when this parameter is a dynamic filter parameter.
- 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.isAdhoc());
```

## isCascadingParameter

- Syntax** boolean ParameterData.isCascadingParameter( )  
 Returns true when this parameter is a cascading parameter.
- Returns** Boolean. True when this parameter is a cascading parameter.
- Example** This example displays the cascading parameter status of a parameter in an alert box:

```
alert(paramData.isCascadingParameter());
```

## isDynamicFilter

- Syntax** boolean ParameterData.isDynamicFilter( )  
 Returns true when this parameter is a dynamic filter.
- Returns** Boolean. True when this parameter is a dynamic filter.

## actuate.parameter.ParameterData

**Example** This example displays the dynamic filter status of a parameter in an alert box:

```
alert (paramData.isDynamicFilter());
```

### isMultiList

**Syntax** boolean ParameterData.isMultiList( )

Returns true when this parameter is shown as a multi-list UI element.

**Returns** Boolean. True when this parameter is shown as a multi-list UI element.

**Example** This example displays the multi-list UI element status of a parameter in an alert box:

```
alert (paramData.isMultiList());
```

### isRequired

**Syntax** boolean ParameterData.isRequired( )

Returns true when this parameter is required.

**Returns** Boolean. True when this parameter is required.

**Example** This example displays the required status of a parameter in an alert box:

```
alert (paramData.isRequired());
```

### setChildData

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

Adds a child parameter to this parameter.

**Parameter** **childData**

An actuate.parameter.ParameterData object that contains the child for this ParameterData object.

**Example** This sample code sets the ParameterData object myChildData as the child of the ParameterData object myParamData:

```
myParamData.setChildData (myChildData) ;
```

### setCurrentValue

**Syntax** void ParameterData.setCurrentValue(actuate.parameter.ParameterValue value)

Sets the UI value of the UI control. When a UI value changes, UIControl calls this method to update the ParameterData object.

**Parameter** **value**

An actuate.parameter.ParameterValue object set by the UI.

- Example** This sample code sets the ParameterValue object myValue as the value of the ParameterData object myParamData:

```
myParamData.setCurrentValue(myValue);
```

## setParentData

- Syntax** void ParameterData.setParentData(actuate.parameter.ParameterData parentData)

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

- Parameter** parentData

An actuate.parameter.ParameterData object that contains the parent for this ParameterData object.

- Example** This sample code sets the ParameterData object myParentData as the parent of the ParameterData object myParamData:

```
myParamData.setParentData(myParentData);
```

## setWebService

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

Defines a web service to use to send SOAP messages.

- Parameter** webService

Object. A web service to send SOAP messages.

---

## Class actuate.parameter.ParameterDefinition

**Description** The ParameterDefinition object contains all of the qualities, values, names, and conditions for a parameter. A ParameterDefinition object can display options to the user and respond to user-generated events. The actuate.Parameter class downloads an array of ParameterDefinition objects with downloadParameters( ). The order of this array is also the order in which the parameters are displayed. Parameters can be grouped to divide the parameters on the page into logical sets under a heading.

This class requires significant memory and bandwidth resources. ParameterValue is much smaller than ParameterDefinition. ParameterValue is the more efficient way to communicate to the server that a parameter value has changed.

### Constructor

**Syntax** actuate.parameter.ParameterDefinition( )

Constructs a new ParameterDefinition object.

### Function summary

Table 7-24 lists actuate.parameter.ParameterDefinition functions.

**Table 7-24** actuate.parameter.ParameterDefinition functions

| Function                   | Description                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------|
| getAutoSuggestThreshold( ) | Gets the auto suggest threshold value for this ParameterDefinition                           |
| getCascadingParentName( )  | Gets the cascadingParentName value for this ParameterDefinition                              |
| getColumnName( )           | Gets the columnName value for this ParameterDefinition                                       |
| getColumnType( )           | Gets the columnType value for this ParameterDefinition                                       |
| getControlType( )          | Gets the controlType value for this ParameterDefinition                                      |
| getCurrentDisplayName( )   | Gets the auto suggest current display name for the current value of this ParameterDefinition |
| getDataType( )             | Gets the dataType value for this ParameterDefinition                                         |
| getDefaultValue( )         | Gets the defaultValue value for this ParameterDefinition                                     |
| getDefaultValueIsNull( )   | Gets a flag if the default value is null for this ParameterDefinition                        |
| getDisplayName( )          | Gets the displayName value for this ParameterDefinition                                      |
| getGroup( )                | Gets the group value for this ParameterDefinition                                            |

**Table 7-24** actuate.parameter.ParameterDefinition functions (continued)

| Function                  | Description                                                        |
|---------------------------|--------------------------------------------------------------------|
| getHelpText()             | Gets the helpText value for this ParameterDefinition               |
| getName()                 | Gets the name value for this ParameterDefinition                   |
| getOperatorList()         | Gets the list of valid operators                                   |
| getPosition()             | Gets the position value for this ParameterDefinition               |
| getSelectNameValueList()  | Gets the selectNameValueList value for this ParameterDefinition    |
| getSelectValueList()      | Gets the selectValueList value for this ParameterDefinition        |
| isAdHoc()                 | Returns whether the parameter is a dynamic filter                  |
| isHidden()                | Gets the isHidden value for this ParameterDefinition               |
| isPassword()              | Gets the isPassword value for this ParameterDefinition             |
| isRequired()              | Gets the isRequired value for this ParameterDefinition             |
| isViewParameter()         | Gets the isViewParameter value for this ParameterDefinition        |
| setAutoSuggestThreshold() | Sets the auto suggest threshold value for this ParameterDefinition |
| setCascadingParentName()  | Sets the cascadingParentName value for this ParameterDefinition    |
| setColumnName()           | Sets the columnName value for this ParameterDefinition             |
| setColumnType()           | Sets the columnType value for this ParameterDefinition             |
| setControlType()          | Sets the controlType value for this ParameterDefinition            |
| setCurrentDisplayName()   | Sets the current display name for this ParameterDefinition         |
| setDataType()             | Sets the dataType value for this ParameterDefinition               |
| setDefaultValue()         | Sets the defaultValue value for this ParameterDefinition           |
| setDefaultValueIsNull()   | Sets the defaultValue to null for this ParameterDefinition         |
| setDisplayName()          | Sets the displayName value for this ParameterDefinition            |
| setGroup()                | Sets the group value for this ParameterDefinition                  |
| setHelpText()             | Sets the helpText value for this ParameterDefinition               |
| setIsAdHoc()              | Sets whether the parameter is a dynamic filter                     |
| setIsHidden()             | Sets the isHidden value for this ParameterDefinition               |
| setIsMultiSelectControl() | Sets the isMultiSelectControl value for this ParameterDefinition   |
| setIsPassword()           | Sets the isPassword value for this ParameterDefinition             |

(continues)

**Table 7-24** actuate.parameter.ParameterDefinition functions (continued)

| Function                  | Description                                                     |
|---------------------------|-----------------------------------------------------------------|
| setIsRequired( )          | Sets theisRequired value for this ParameterDefinition           |
| setIsViewParameter( )     | Sets the isViewParameter value for this ParameterDefinition     |
| setName( )                | Sets the name value for this ParameterDefinition                |
| setPosition( )            | Sets the position value for this ParameterDefinition            |
| setSelectNameValueList( ) | Sets the selectNameValueList value for this ParameterDefinition |
| setSelectValueList( )     | Sets the selectValueList value for this ParameterDefinition     |

## getAutoSuggestThreshold

**Syntax** `integer ParameterDefinition.getAutoSuggestThreshold( )`

Gets the auto suggest threshold value for this ParameterDefinition. The auto suggest threshold determines the number of characters a user types in before they are given suggestions from auto suggest.

**Returns** Integer.

**Example** To store the auto suggest threshold of the parameter definition paramdef in a variable called threshold, use code similar to the following:

```
var threshold = paramdef.getAutoSuggestThreshold();
```

## get CascadingParentName

**Syntax** `string ParameterDefinition.getCascadingParentName( )`

Gets the cascadingParentName value for this ParameterDefinition. A cascading parent parameter is only used when one parameter depends upon another.

**Returns** String.

**Example** To store the cascading parent name of the parameter definition paramdef in a variable called parentname, use code similar to the following:

```
var parentname = paramdef.getCascadingParentName();
```

## getColumnName

**Syntax** `string ParameterDefinition.getColumnName( )`

Gets the columnName value for this ParameterDefinition. This setting sets the column to retrieve data from for a dynamic filter parameter that performs a query.

This setting has no effect on other types of parameters.

**Returns** String.

**Example** To store the column name of the parameter definition paramdef in a variable called columnname, use code similar to the following:

```
var columnname = paramdef.getColumnname();
```

## getColumnType

**Syntax** string ParameterDefinition.getColumnType()

Gets the columnType value for this ParameterDefinition. This setting sets the data type queried by an ad hoc parameter that performs a query.

This setting has no effect on other types parameters.

**Returns** String. Possible values are: null, "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

**Example** To store the column type of the parameter definition paramdef in a variable called columntype, use code similar to the following:

```
var columntype = paramdef.getColumnType();
```

## getControlType

**Syntax** string ParameterDefinition.getControlType()

Gets the controlType value for this ParameterDefinition. It determines the form element displayed for the user to set the parameter value.

**Returns** String. Possible values are: null, "", "ControlRadioButton", "ControlList", "ControlListAllowNew", and "ControlCheckBox".

**Example** To store the control type string for the parameter definition paramdef in a variable called controltype, use code similar to the following:

```
var columntype = paramdef.getColumnType();
```

## getCurrentDisplayName

**Syntax** string ParameterDefinition.getCurrentDisplayName()

Gets the current display name for this ParameterDefinition.

**Returns** String.

**Example** To store the current display name of the parameter definition paramdef in a variable called displayname, use code similar to the following:

```
var displayname = paramdef.getDisplayName();
```

## getDataType

**Syntax** string ParameterDefinition.getDataType( )

Gets the dataType value for this ParameterDefinition.

**Returns** String. Possible values are: "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

**Example** To store the data type of the parameter definition paramdef in a variable called type, use code similar to the following:

```
var type = paramdef.getDataType();
```

## getDefaultValue

**Syntax** string ParameterDefinition.getDefaultValue( )

Gets the defaultValue value for this ParameterDefinition, if applicable.

**Returns** String.

**Example** To store the default value as a string for the parameter definition paramdef in a variable called default, use code similar to the following:

```
var default = paramdef.getDefaultValue();
```

## 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.getDefaultValuesNull()) {
 alert('Default value is null!');
}
```

## getDisplayName

**Syntax** string ParameterDefinition.getDisplayName( )

Gets the displayName for this ParameterDefinition.

**Returns** String.

**Example** To store the displayed name for the parameter definition paramdef in a variable called displayname, use code similar to the following:

```
var displayname = paramdef.getDisplayName();
```

## getGroup

**Syntax** string ParameterDefinition.getGroup( )

Gets the group for this ParameterDefinition, indicating if it is a member of a group.

**Returns** String. A group name, or null if there is no group.

**Example** To print the group name for the parameter definition paramdef to the current document, use code similar to the following:

```
document.write(paramdef.getGroup());
```

## getHelpText

**Syntax** string ParameterDefinition.getHelpText( )

Gets the helpText for this ParameterDefinition.

**Returns** String. The help text.

**Example** To store the help text for the parameter definition paramdef in a variable called helptext, use code similar to the following:

```
var helptext = paramdef.getHelpText();
```

## getName

**Syntax** string ParameterDefinition.getName( )

Gets the name for this ParameterDefinition.

**Returns** String. The parameter name.

**Example** To store the name for the parameter definition paramdef in a variable called paramname, use code similar to the following:

```
var paramname = paramdef.getName();
```

## getOperatorList

**Syntax** string[ ] ParameterDefinition.getOperatorList( )

Gets the operator list for this ParameterDefinition.

**Returns** An array of strings containing the operator list.

**Example** To store the list of operators for the parameter definition paramdef in a variable called ops, use code similar to the following:

```
var ops = new Array();
ops = paramdef.getOperatorList();
```

## getPosition

**Syntax** Integer ParameterDefinition.getPosition( )

Gets the position in the array for this ParameterDefinition.

**Returns** Integer.

**Example** To store the position of the parameter definition paramdef in a variable called position, use code similar to the following:

```
var position = paramdef.getPosition();
```

## getSelectNameValueList

**Syntax** selectNameValueList[ ] ParameterDefinition.getSelectNameValueList( )

Gets the selectNameValueList for this ParameterDefinition. This list applies if the parameter is set with a selection list.

**Returns** Array of actuate.parameter.NameValuePair objects.

**Example** To retrieve the name-value pair list for the parameter definition paramdef and put it into a new array, use code similar to the following:

```
var namevalues = new array();
namevalues = paramdef.getSelectNameValueList().slice();
```

## getSelectValueList

**Syntax** string[ ] ParameterDefinition.getSelectValueList( )

Gets the selectValueList for this ParameterDefinition. This list applies when the parameter is set with a selection list.

**Returns** An array of strings containing the select value list.

**Example** To retrieve the list of values selectable for the parameter definition paramdef and put it into a new array, use code similar to the following:

```
var selectvalues = new array();
selectvalues = paramdef.getSelectValueList().slice();
```

## isAdHoc

**Syntax** boolean ParameterDefinition.isAdHoc( )

Returns true when this parameter is a dynamic filter parameter.

**Returns** Boolean. True when this parameter is a dynamic filter.

**Example** To set the default value to null for the parameter definition paramdef if it is a dynamic filter parameter, use code similar to the following:

```
if (paramdef.isAdHoc()) {
 paramdef.setDefaultvalueIsNull(true) ;
}
```

## isHidden

**Syntax** boolean ParameterDefinition.isHidden( )

Gets the isHidden value for this ParameterDefinition.

**Returns** Boolean. True indicates that this parameter is hidden.

**Example** To reveal a parameter with the parameter definition paramdef if it is hidden, use code similar to the following:

```
if (paramdef.isHidden()) {
 paramdef.setIsHidden(false) ;
}
```

## isPassword

**Syntax** boolean ParameterDefinition.isPassword( )

Gets the isPassword value for this ParameterDefinition.

**Returns** Boolean. True indicates that the parameter is a password.

**Example** To set the parameter definition paramdef as required if it is a password parameter, use code similar to the following:

```
if (paramdef.isPassword()) {
 paramdef.setIsRequired(true) ;
}
```

## isRequired

**Syntax** boolean ParameterDefinition.isRequired( )

Gets the isRequired value for this ParameterDefinition.

**Returns** Boolean. True indicates that the parameter is required.

**Example** To set specific help text for the parameter definition paramdef if it is a required parameter, use code similar to the following:

```
if (paramdef.isRequired()) {
 paramdef.setHelpText("This parameter is required.");
}
```

## isViewParameter

**Syntax** boolean ParameterDefinition.isViewParameter( )

## actuate.parameter.ParameterDefinition

Gets the isViewParameter value for this ParameterDefinition.

**Returns** Boolean. True indicates that the parameter is a view-time parameter. False indicates that the parameter is a run-time parameter.

**Example** To set specific help text for the parameter definition paramdef if it is a view-time parameter, use code similar to the following:

```
if (paramdef.isViewParameter()){
 paramdef.setHelpText("This is a view-time parameter.");
}
```

## setAutoSuggestThreshold

**Syntax** void ParameterDefinition.setAutoSuggestThreshold(integer threshold)

Sets the auto suggest threshold for this ParameterDefinition. The auto suggest threshold determines the number of characters a user types in before they are given suggestions from auto suggest.

**Parameter** **threshold**  
Integer.

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

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

## setCascadingParentName

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

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

**Parameter** **cascadingParentName**  
String.

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

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

## setColumnName

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

Sets the columnName for this ParameterDefinition. Used for queries.

**Parameter** **columnName**  
String.

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

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

## setColumnType

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

Sets the columnType for this ParameterDefinition. Used for queries.

**Parameter** **columnType**

String. Possible values are null, "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

- Example** To allow the parameter definition paramdef to interpret a column as untyped data, use code similar to the following:

```
paramdef.setColumnType ("Unknown") ;
```

## setControlType

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

Sets the control type of this ParameterDefinition.

**Parameter** **controlType**

String. Possible values are null, "", "AutoSuggest", "ControlRadioButton", "ControlList", "ControlListAllowNew", and "ControlCheckBox".

- Example** To set the parameter definition paramdef to use a control list, use code similar to the following:

```
paramdef.setControlType ("ControlList") ;
```

## setCurrentDisplayName

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

Sets the displayed name for this parameter.

**Parameter** **currentDisplayName**

String.

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

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

## setDataType

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

## actuate.parameter.ParameterDefinition

Sets the dataType for this ParameterDefinition.

**Parameter** **dataType**

String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

**Example** To set the parameter definition paramdef data type to date, use code similar to the following:

```
paramdef.setDataType("Date");
```

## setDefaultValue

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

Sets the default value for this ParameterDefinition.

**Parameter** **defaultValue**

String.

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

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

## setDefaultValuesNull

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

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

**Parameter** **value**

Boolean.

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

```
paramdef.setDefaultValueIsNull(true);
```

## setDisplayName

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

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

**Parameter** **displayName**

String.

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

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

## setGroup

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

Sets the group value for this ParameterDefinition.

**Parameter** **group**  
String.

**Example** To assign the parameter definition paramdef to the "Customer Details" parameter group, use code similar to the following:

```
paramdef.setGroup("Customer Details");
```

## setHelpText

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

Sets the helpText value for this ParameterDefinition.

**Parameter** **helpText**  
String.

**Example** To set specific help text for the parameter definition paramdef if it is a required parameter, use code similar to the following:

```
if (paramdef.isRequired()){
 paramdef.setHelpText("This parameter is required.");
}
```

## setIsAdHoc

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

Sets this parameter as a dynamic filter parameter.

**Parameter** **isAdHoc**  
Boolean. True makes this parameter into a dynamic filter.

**Example** To enable the parameter definition paramdef to accept dynamic filter values, use code similar to the following:

```
paramdef.setIsAdHoc(true);
```

## setIsHidden

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

Sets the parameter to hidden.

**Parameter** **isHidden**  
Boolean. True hides the parameter.

**Example** To hide a parameter defined by a parameter definition called paramdef, use code similar to the following:

```
paramdef.setIsHidden(true);
```

## setIsMultiSelectControl

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

Sets the parameter to accept multiple selected values.

**Parameter** **isMultiSelect**

Boolean. True allows multiple selected values to be set for this parameter.

**Example** To allow a parameter defined by a parameter definition called paramdef to accept multiple selected values, use code similar to the following:

```
paramdef.setIsMultiSelectControl(true);
```

## setIsPassword

**Syntax** void ParameterDefinition.setIsPassword(boolean isPassword)

Sets this parameter to treat its value as a password, which hides the input on the page and encrypts the value.

**Parameter** **isPassword**

Boolean. True indicates a password value.

**Example** To set the parameter type accepted by the parameter definition paramdef to password, use code similar to the following:

```
paramdef.setIsPassword(true);
```

## setIsRequired

**Syntax** void ParameterDefinition.setIsRequired(booleanisRequired)

Sets the parameter to required.

**Parameter** **isRequired**

Boolean. True indicates a mandatory parameter.

**Example** To make the parameter defined by the parameter definition paramdef mandatory, use code similar to the following:

```
paramdef.setIsRequired(true);
```

## setIsViewParameter

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

Sets the isViewParameter value for this ParameterDefinition.

**Parameter** **isViewParameter**  
Boolean.

**Example** To make the parameter defined by the parameter definition paramdef a view-time parameter, use code similar to the following:

```
paramdef.setIsViewParameter(true);
```

## setName

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

Sets the name to use internally for this ParameterDefinition.

**Parameter** **name**  
String.

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

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

## setPosition

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

Sets the position value for this ParameterDefinition. The index indicates the position in the array of the ParameterDefinition.

**Parameter** **position**  
Integer.

**Example** To shift the parameter definition paramdef down on position in the parameter array, use code similar to the following:

```
paramdef.setPosition(++paramdef.getPosition());
```

## setSelectNameValueList

**Syntax** void ParameterDefinition.setSelectNameValueList  
(actuate.parameter.NameValuePair[ ] selectNameValueList)

Sets the selectNameValueList value for this ParameterDefinition.

**Parameter** **selectNameValueList**  
Array of actuate.parameter.NameValuePair objects.

**Example** To set the parameter definition paramdef to select the same name-value list as the parameter definition nparm, use code similar to the following:

```
paramdef.setSelectNameValueList(nparam.getSelectNameValueList());
```

## setSelectValueList

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

Sets the selectValueList value for this ParameterDefinition.

**Parameter** **selectValueList**

Array.

**Example** To set the parameter definition paramdef to select the values 2007-2009, use code similar to the following:

```
var values = new Array("2007", "2008", "2009");
paramdef.setSelectValueList(values);
```

---

## Class actuate.parameter.ParameterValue

**Description** ParameterValue is a container for the value of Parameter to be passed to a report for processing. When a user sets a value in the interface, the corresponding ParameterValue must change.

Because ParameterValue is much smaller 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 7-25 lists actuate.parameter.ParameterValue functions.

**Table 7-25** actuate.parameter.ParameterValue functions

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

*(continues)*

**Table 7-25** actuate.parameter.ParameterValue functions (continued)

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

## getColumnName

**Syntax** string ParameterValue.getColumnName( )

Gets the column name value for this ParameterValue. Columns are supported as part of ad hoc parameters.

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

**Example** To store the column name for the parameter value pvalue in a variable called columnname, use code similar to the following:

```
var columnname = pvalue.getColumnName();
```

## getColumnType

**Syntax** string ParameterValue.getColumnType( )

Gets the data type value of the column for this ParameterValue. Columns are supported as part of ad hoc parameters.

**Returns** String. Possible values are null, "", "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

- Example** To store the column type for the parameter value pvalue in a variable called columntype, use code similar to the following:

```
var columntype = pvalue.getColumnType();
```

## getDataType

**Syntax** string ParameterValue.getDataType( )

Gets the dataType value for this ParameterValue.

**Returns** String. Possible values are null, "", "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

- Example** To store the data type for the parameter value pvalue in a variable called type, use code similar to the following:

```
var type = pvalue.getDataType();
```

## getDisplayName

**Syntax** string ParameterValue.getDisplayName( )

Gets the displayed name for this ParameterValue.

**Returns** String. The displayed name.

- Example** To store the displayed name of the parameter value pvalue in a variable called displayName, use code similar to the following:

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

## getGroup

**Syntax** string ParameterValue.getGroup( )

Gets the group value for this ParameterValue.

**Returns** String.

- Example** To store the group that the parameter value pvalue belongs to in a variable called group, use code similar to the following:

```
var group = pvalue.getGroup();
```

## getName

**Syntax** string ParameterValue.getName( )

Gets the name value for this ParameterValue.

**Returns** String.

## actuate.parameter.ParameterValue

**Example** To store the name of the parameter value pvalue in a variable called name, use code similar to the following:

```
var name = pvalue.getName();
```

### getPosition

**Syntax** integer ParameterValue.getPosition()

Gets the position value for this ParameterValue.

**Returns** Integer.

**Example** To save the position of the parameter value pvalue in the parameter list to a variable called pos, use code similar to the following:

```
var pos = pvalue.getPosition();
```

### getPromptParameter

**Syntax** boolean ParameterValue.getPromptParameter()

Gets the promptParameter value for this ParameterValue.

**Returns** Boolean.

**Example** To store the prompt parameter of the parameter value pvalue in a variable called prompt, use code similar to the following:

```
var prompt = pvalue.getPromptParameter();
```

### getValue

**Syntax** string[ ] ParameterValue.getValue()

Gets the value values for this ParameterValue.

**Returns** String or array of strings. The value or values of this ParameterValue object.

**Example** To store the value of the parameter value pvalue in a variable called value, use code similar to the following:

```
var value = pvalue.getValue();
```

### getValuesNull

**Syntax** boolean ParameterValue.getValuesNull()

Gets the valueIsNull value for this ParameterValue.

**Returns** Boolean. True indicates that this ParameterValue is null.

- Example** To alert the user that the value of the parameter value pvalue is null, use code similar to the following:

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

## isViewParameter

**Syntax** boolean ParameterValue.isViewParameter()

Gets the isViewParameter value for this ParameterValue.

**Returns** Boolean. True indicates that this ParameterValue is visible.

- Example** To set specific help text for the parameter value pvalue if it is a view-time parameter, use code similar to the following:

```
if (pvalue.isViewParameter()) {
 pvalue.setHelpText("This is a view-time parameter.");
}
```

## setColumnName

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

Sets the column name value for this ParameterValue.

**Parameter** **columnName**

String. The name of the column.

- Example** To set the column name for the parameter value pvalue to Year, use code similar to the following:

```
pvalue.setColumnName ("Year");
```

## setColumnType

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

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

**Parameter** **columnType**

String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

- Example** To set the column type for the parameter value pvalue to Date, use code similar to the following:

```
pvalue.setColumnTpe ("Date");
```

## setDataType

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

Sets the dataType value for this ParameterValue.

**Parameter** **dataType**

String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

**Example** To set the data type for the parameter value pvalue to Date, use code similar to the following:

```
pvalue.setDataType("Date");
```

## setDisplayName

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

Sets the displayed name value for this ParameterValue.

**Parameter** **name**

String. A displayed parameter name.

**Example** To set the display name of the parameter value pvalue to Year, use code similar to the following:

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

## setGroup

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

Sets the group value for this ParameterValue.

**Parameter** **group**

String. The name of the group.

**Example** To set the group for the parameter value pvalue to Customer Details, use code similar to the following:

```
pvalue.setGroup("Customer Details");
```

## setIsViewParameter

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

Sets the isViewParameter value for this ParameterValue.

**Parameter** **isViewParameter**

Boolean. True indicates a view-time parameter.

- Example** To make the parameter value pvalue into a view-time parameter, use code similar to the following:

```
pvalue.setIsViewParameter(true);
```

## setName

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

Sets the name value for this ParameterValue.

**Parameter** **name**

String. A parameter name.

- Example** To set the name of the parameter value pvalue to Year, use code similar to the following:

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

## setPosition

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

Sets the position value for this ParameterValue.

**Parameter** **position**

Integer. The position from the top of the parameter list.

- Example** To move the parameter value pvalue one place farther down in the parameter list, use code similar to the following:

```
pvalue.setPosition(++pvalue.getPosition());
```

## setPromptParameter

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

Sets the promptParameter value for this ParameterValue.

**Parameter** **promptParameter**

Boolean. True indicates that this parameter prompts the user.

- Example** To set the parameter value pvalue to not prompt the user, use code similar to the following:

```
pvalue.setPromptParameter(false);
```

## setValue

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

Sets the value or values for this ParameterValue.

## actuate.parameter.ParameterValue

### Parameter **value**

String or array of strings. The value or values of this ParameterValue object.

**Example** To set the value of the parameter value pvalue to 2010, use code similar to the following:

```
pvalue.setValue("2010");
```

To set the values of the ParameterValue object pvalues to 2008, 2009, and 2010, use code similar to the following:

```
pvalue.setValue({ "2008", "2009", "2010" });
```

## setValuesNull

### Syntax void ParameterValue.setValuesNull(boolean valueIsNull)

Sets the valueIsNull value for this ParameterValue.

### Parameter **valueIsNull**

Boolean. True indicates that this ParameterValue is null.

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

```
pvalue.setValueIsNull(true);
```

---

## Class actuate.report.Chart

**Description** Provides functions to operate on a chart element, such as changing its format or retrieving data from specific elements.

### Constructor

The actuate.report.Chart object is created when `actuate.viewer.PageContent.getChartByBookmark()` is called.

### Function summary

Table 7-26 lists actuate.report.Chart functions.

**Table 7-26** 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                                                                                                             |
| <code>submit()</code>            | Submits all the asynchronous operations that the user has requested on this report and renders the chart component on the page |

## clearFilters

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

Clears the filters for a given column.

**Parameter** **columnName**

String. The name of the column.

**Example** This example clears existing filters from the PRODUCTLINE column of a chart and changes the chart title:

```
function resetFilter(bchart) {
 bchart.clearFilters("PRODUCTLINE");
 bchart.setChartTitle("Orders By Country");
 bchart.submit();
}
```

## drillDownCategory

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

Drills down into a chart by category.

**Parameter** **categoryData**

String. The name of the data category to drill down to.

## drillDownSeries

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

Drills down into a chart by series.

**Parameter** **seriesName**

String. The name of the data series to drill down to.

## drillUpCategory

**Syntax** void Chart.drillUpCategory()

Drills up into a chart by one data category level.

## drillUpSeries

**Syntax** void Chart.drillUpSeries()

Drills up into a chart by one series level.

## getBookmark

**Syntax** string Chart.getBookmark( )

Returns the chart's bookmark name.

**Returns** String. The chart's bookmark name.

**Example** This example sets the chart's title to the bookmark name:

```
function titleBookmark(bchart) {
 bchart.setChartTitle(bchart.getBookmark());
 bchart.submit();
}
```

## getClientChart

**Syntax** actuate.report.HTML5Chart.ClientChart Chart.getClientChart( )

Returns the HTML5 Chart instance if this chart has an HTML5 Chart output format, otherwise returns null.

**Returns** actuate.report.HTML5Chart.ClientChart. The HTML5 formatted chart or null.

**Example** This example displays the chart ID of the HTML5 chart in an alert box:

```
function showHTML5ChartID(myChart) {
 var myHTML5Chart = myChart.getClientChart();
 var HTML5ChartID = myHTML5Chart.getViewerId();
 alert (HTML5ChartID);
}
```

## getHtmlDom

**Syntax** HTMLElement Chart.getHtmlDom( )

Returns the HTML element for this chart.

**Returns** HTMLElement. The HTML DOM element.

**Example** This example displays the HTML DOM element for this chart inside a red border:

```
function showHtmlDom(myChart) {
 var domNode = myChart.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceld

**Syntax** string Chart.getInstanceld( )

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```
function showID(myChart){
 var elementID = myChart.getInstanceId();
 alert (elementID);
}
```

## getPageContent

**Syntax** actuate.viewer.PageContent Chart.getPageContent( )

Returns the content of the page to which this chart belongs.

**Returns** actuate.report.PageContent. The report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewerID(myChart){
 var pageContent = myChart.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getType

**Syntax** string Chart.getType( )

Returns the chart's report element type.

**Returns** String. This method returns the string "Chart" when the type is actuate.report.Chart.CHART and the string "Flash Chart" when the type is actuate.report.Chart.FLASH\_CHART.

**Example** This example displays the chart type in an alert box:

```
alert ("Chart is of type " + myChart.getType());
```

## hide

**Syntax** void Chart.hide( )

Hides this element.

**Example** To hide the chart bchart, use code similar to the following:

```
alert("Hiding chart" + bchart.getBookmark());
bchart.hide();
bchart.submit();
```

## setChartTitle

**Syntax** void Chart.setChartTitle(string title)

Sets the title for this chart element.

**Parameter** **title**

String. The title for the chart.

**Example** This example sets the chart's title to the bookmark name:

```
function titleBookmark(bchart) {
 bchart.setChartTitle(bchart.getBookmark());
 bchart.submit();
}
```

## setDimension

**Syntax** void Chart.setDimension(actuate.report.Chart dimension)

Sets the number of dimensions for the chart element. The chart dimension only works if supported by the chart's type. A 3D chart does not support multiple value axes. Remove all of the *y*-axes after the first before converting a chart to 3D.

**Parameter** **dimension**

actuate.report.Chart. The number of dimensions in which to display the chart element. Supported values are 2D and 2D with depth. The constants defined for this argument are:

- actuate.report.Chart.CHART\_DIMENSION\_2D
- actuate.report.Chart.CHART\_DIMENSION\_2D\_WITH\_DEPTH

**Example** This example changes the chart bchart's dimension to 2D with depth:

```
bchart.setChartTitle(bchart.getBookmark() + ": 2D with Depth");
bchart.setDimension(actuate.report.Chart
 .CHART_DIMENSION_2D_WITH_DEPTH);
bchart.submit();
```

## setFilters

**Syntax** void Chart.setFilters(actuate.data.Filter filter)

void Chart.setFilters(actuate.data.Filter[ ] filters)

Applies filters to this chart element. To apply more than one filter to a chart element, call this function multiple times, once for each filter object.

## actuate.report.Chart

|                   |                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <b>filter</b><br>An actuate.data.Filter object. A single filter condition to apply to this chart element.<br><br><b>filters</b><br>An array of actuate.data.Filter objects. Filter conditions to apply to this chart element. |
| <b>Example</b>    | This example applies a filter to the chart and changes the chart's title to reflect the filter:                                                                                                                               |

```
function chartFilter(bchart){
 var filter = new actuate.data.Filter("PRODUCTLINE", "=",
 "Trucks and Buses");
 var filters = new Array();
 filters.push(filter);
 bchart.setFilters(filters);
 bchart.setChartTitle("Orders By Country (Trucks and Buses)");
 bchart.submit();
}
```

## setSize

|                   |                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>     | void Chart.setSize(integer width, integer height)<br><br>Sets the width and height of the chart element displayed.                                                                                                                  |
| <b>Parameters</b> | <b>width</b><br>Integer. The width in pixels.<br><br><b>height</b><br>Integer. The height in pixels.                                                                                                                                |
| <b>Example</b>    | To set the chart bchart to be 600 pixels wide by 800 pixels high, use code similar to the following:<br><br><pre>alert("Resizing " + bchart.getBookmark() + " to 600x800");<br/>bchart.setSize(600,800);<br/>bchart.submit();</pre> |
| <b>setSubType</b> |                                                                                                                                                                                                                                     |

- CHART\_SUBTYPE\_PERCENTSTACKED
- CHART\_SUBTYPE\_SIDEBYSIDE
- CHART\_SUBTYPE\_STACKED

**Example** To change the subtype of the chart bchart to side-by-side, use code similar to the following:

```
bchart.setChartTitle("Side by Side Chart");
bchart.setSubType(actuate.report.Chart.CHART_SUBTYPE_SIDEBYSIDE);
bchart.submit();
```

## show

**Syntax** void Chart.show( )

Shows this element.

**Example** To reveal the hidden chart bchart, use code similar to the following:

```
alert("Showing chart" + bchart.getBookmark());
bchart.show();
bchart.submit();
```

## submit

**Syntax** void Chart.submit(function callback)

Submits all the asynchronous operations for this chart. The submit( ) function triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the chart container.

**Parameter** **callback**

Function. Optional. A function to execute after the asynchronous call processing is done. Submit passes the current actuate.Viewer object to the callback as an input parameter.

**Example** This example sets the chart's title to the bookmark name and pops up an alert box after calling submit( ):

```
function titleBookmark(bchart) {
 bchart.setChartTitle(bchart.getBookmark());
 bchart.submit(alert("Title Changed"));
}
```

---

## Class actuate.report.DataItem

**Description** A container for a data element in a report. DataItem provides functions to operate on a data element, such as retrieving the data value and getting the HTML DOM element from the report data element.

### Constructor

The DataItem object is constructed by  
actuate.viewer.PageContent.getDataItemByBookmark( ).

### Function summary

Table 7-27 lists actuate.report.DataItem functions.

**Table 7-27** actuate.report.DataItem functions

| Function         | Description                                            |
|------------------|--------------------------------------------------------|
| getBookmark()    | Returns the bookmark name for this data item           |
| getData()        | Returns the data value on this data element            |
| getHtmlDom()     | Returns the HTML element for this data item            |
| getInstanceId()  | Returns the instance id of this report element.        |
| getPageContent() | Returns the page content to which this element belongs |
| getType()        | Returns the report element type                        |
| hide()           | Hides this element                                     |
| show()           | Shows this element                                     |
| submit()         | Applies the changes made to this DataItem              |

### getBookmark

**Syntax** string DataItem.getBookmark( )

Returns the bookmark name for this data item.

**Returns** String.

**Example** This example displays the data item's bookmark in an alert box:

```
alert(myDataItem.getBookmark());
```

## getData

**Syntax** string DataItem.getData( )

Returns the data value of this data element.

**Returns** String. The data value.

**Example** This example displays the data element's data value in an alert box:

```
alert(myDataItem.getData());
```

## getHtmlDom

**Syntax** HTMLElement DataItem.getHtmlDom( )

Returns the HTML element for this data item.

**Returns** HTMLElement.

**Example** This example displays the HTML DOM element for this data item inside a red border:

```
function showHtmlDom(myDataItem) {
 var domNode = myDataItem.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** string DataItem.getInstanceId( )

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```
function showID(myDataItem) {
 var elementID = myDataItem.getInstanceId();
 alert(elementID);
}
```

## getPageContent

**Syntax** actuate.viewer.PageContent DataItem.getPageContent( )

actuate.report.DataItem

Returns the page content to which this data item belongs.

**Returns** actuate.report.PageContent. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myDataItem) {
 var pageContent = myDataItem.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getType

**Syntax** string DataItem.getType( )

Returns the report element type of this object, which is data.

**Returns** String. "Data".

**Example** This example checks the report element type and displays an alert if the type is not "Data":

```
if (myDataItem.getType() != "Data") {
 alert("Type mismatch, report element type is not data");
}
```

## hide

**Syntax** void DataItem.hide( )

Hides this element.

**Example** Use hide( ) to hide a data item object, as shown in the following code:

```
myDataItem.hide();
```

## show

**Syntax** void DataItem.show( )

Shows this element.

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

```
myDataItem.show();
```

## submit

**Syntax** void DataItem.submit(function callback)

Submits all the asynchronous operations for this DataItem. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the

processing, it returns a response and the results are rendered on the page in the DataItem container.

**Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

**Example** Use submit( ) to execute changes on a data item object, as shown in the following code:

```
myDataItem.submit();
```

---

## Class actuate.report.FlashObject

**Description** A container for a Flash object in a report. FlashObject provides functions to operate on a Flash object, such as retrieving content and getting the HTML DOM element from the report Flash element.

### Constructor

The FlashObject object is constructed by  
actuate.viewer.PageContent.getFlashObjectByBookmark( ).

### Function summary

Table 7-28 lists actuate.report.FlashObject functions.

**Table 7-28** actuate.report.FlashObject functions

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

### clearFilters

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

Clears the filters of a given column.

**Parameter** **columnName**

String. The name of the column from which to clear the filters.

**Example** This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(flashobj) {
 flashobj.clearFilters("PRODUCTLINE");
```

```

 flashobj.submit();
 }
}
```

## getBookmark

**Syntax** string FlashObject.getBookmark( )

Returns the bookmark of this FlashObject element.

**Returns** String.

**Example** This example displays the Flash object's bookmark in an alert box:

```

function alertBookmark(myFlashobj) {
 alert(myFlashobj.getBookmark());
}
```

## getHtmlDom

**Syntax** HTMLElement FlashObject.getHtmlDom( )

Returns the HTML element for this FlashObject.

**Returns** HTMLElement.

**Example** This example displays the HTML DOM element for this data item inside a red border:

```

function showHtmlDom(myFlashobj) {
 var domNode = myFlashobj.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** string FlashObject.getInstanceId( )

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```

function showID(myFlashObject) {
 var elementID = myFlashObject.getInstanceId();
```

```
actuate.report.FlashObject
```

```
 alert (elementID) ;
}
```

## getPageContent

**Syntax** `actuate.viewer.PageContent FlashObject.getPageContent( )`

Returns the page content to which this FlashObject belongs.

**Returns** `actuate.viewer.PageContent`. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myFlashobj) {
 var pageContent = myFlashobj.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getType

**Syntax** `string FlashObject.getType( )`

Returns the report element type of this object, which is FlashObject.

**Returns** String. "FlashObject".

**Example** This example checks the report element type and displays an alert if the type is not "FlashObject":

```
if (myFlashObject.getType() != "FlashObject") {
 alert("Type mismatch, report element type is not FlashObject");
}
```

## hide

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

Hides this element.

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

```
myFlashobj.hide();
```

## setFilters

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

Sets the given filters.

**Parameter filters**

An array of actuate.data.Filter objects. The filter conditions to apply to this chart element.

**Example** This example applies a filter to the Flash object:

```
function newFilter(myFlashobj) {
 var filter = new
 actuate.data.Filter("PRODUCTLINE", "=", "Trucks and Buses");
 var filters = new Array();
 filters.push(filter);
 myFlashobj.setFilters(filters);
}
```

## show

**Syntax** void FlashObject.show( )

Shows this element.

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

```
myFlashobj.show();
```

## submit

**Syntax** void FlashObject.submit(function callback)

Submits all the asynchronous operations for this FlashObject. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the FlashObject container.

**Parameter callback**

Function. The function to execute after the asynchronous call processing is done.

**Example** This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(flashobj){
 flashobj.clearFilters("PRODUCTLINE");
 flashobj.submit(alert("Filters Cleared"));
}
```

---

## Class actuate.report.Gadget

**Description** A container for a Flash gadget object in a report. The Gadget class provides functions to operate on a Flash gadget object, such as retrieving content and getting the HTML DOM element from the report Flash element.

### Constructor

The Gadget object is constructed by  
viewer.PageContent.getGadgetByBookmark( ).

### Function summary

Table 7-29 lists actuate.report.Gadget functions.

**Table 7-29** actuate.report.Gadget functions

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

### clearFilters

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

Clears the filters of a given column.

**Parameter** **columnName**

String. The name of the column from which to clear the filters.

**Example** This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(myGadget) {
 myGadget.clearFilters("PRODUCTLINE");
 myGadget.submit();
}
```

## getBookmark

**Syntax** string Gadget.getBookmark( )

Returns the bookmark of this Gadget element.

**Returns** String. The gadget's bookmark.

**Example** This example displays the gadget's bookmark in an alert box:

```
function alertBookmark(myGadget) {
 alert(myGadget.getBookmark());
}
```

## getHtmlDom

**Syntax** HTMLElement Gadget.getHtmlDom( )

Returns the HTML element for this gadget.

**Returns** HTMLElement.

**Example** This example displays the HTML DOM element for this gadget inside a red border:

```
function showHtmlDom(myGadget) {
 var domNode = myGadget.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** string Gadget.getInstanceId( )

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

actuate.report.Gadget

```
function showID(myGadget) {
 var elementID = myGadget.getInstanceId();
 alert (elementID);
}
```

## getPageContent

**Syntax** actuate.viewer.PageContent Gadget.getPageContent( )

Returns the page content to which this gadget belongs.

**Returns** actuate.viewer.PageContent. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myGadget) {
 var pageContent = myGadget.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getType

**Syntax** string Gadget.getType( )

Returns the report element type of this object, which is Gadget.

**Returns** String. "Gadget".

**Example** This example checks the report element type and displays an alert if the type is not "Gadget":

```
if (myGadget.getType() != "Gadget") {
 alert("Type mismatch, report element type is not Gadget");
}
```

## hide

**Syntax** void Gadget.hide( )

Hides this element.

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

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

## setFilters

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

Sets the given filters.

**Parameter** **filters**

An array of actuate.data.Filter objects. The filter conditions to apply to this chart element.

**Example** This example applies a filter to the gadget:

```
function newFilter(myGadget) {
 var filter = new
 actuate.data.Filter("PRODUCTLINE", "=", "Trucks and Buses");
 var filters = new Array();
 filters.push(filter);
 myGadget.setFilters(filters);
}
```

## setGadgetType

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

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

**Parameter** **chartType**

String. The possible values are constants as listed below:

- GADGET\_TYPE\_BULLET: Bullet gadget subtype
- GADGET\_TYPE\_CYLINDER: Cylinder gadget subtype
- GADGET\_TYPE\_LINEARGAUGE: LinearGauge gadget subtype
- GADGET\_TYPE\_METER: Meter gadget subtype
- GADGET\_TYPE\_SPARK: Spark gadget subtype
- GADGET\_TYPE\_THERMOMETER: Thermometer gadget subtype

**Example** To change the gadget type to a meter, use code similar to the following:

```
myGadget.setGadgetType(actuate.report.Gadget.GADGET_TYPE_METER);
```

## setSize

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

Specifies the width and height of a gadget in pixels.

**Parameters** **width**

Integer. The width in pixels.

**height**

Integer. The height in pixels.

## actuate.report.Gadget

- Example** To set the gadget to a 300-by-300-pixel square area, use code similar to the following:

```
myGadget.setSize(300, 300);
```

## show

- Syntax** void Gadget.show( )

Shows this element.

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

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

## submit

- Syntax** void Gadget.submit(function callback)

Submits all the asynchronous operations for this gadget. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the gadget container.

- Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

- Example** This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(myGadget) {
 myGadget.clearFilters("PRODUCTLINE");
 myGadget.submit(alert("Filters Cleared"));
}
```

---

## Class actuate.report.HTML5Chart.ClientChart

**Description** A container for an HTML5-enabled chart element in a report. ClientChart provides functions to operate on a ClientChart element on the client side only, such as retrieving the chart size or setting the title and series values for the currently displayed chart.

### Constructor

The ClientChart object is constructed by `actuate.report.Chart.getClientChart()`.

### Function summary

Table 7-30 lists `actuate.report.HTML5Chart.ClientChart` functions.

**Table 7-30** `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>getAxisMax()</code>       | Returns the maximum value of X-axis series                                       |
| <code>getAxisMin()</code>       | Returns the minimum value of X-axis series                                       |
| <code>getAxisMax()</code>       | Returns the maximum value of Y-axis series                                       |
| <code>getAxisMin()</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 |

*(continues)*

**Table 7-30** actuate.report.HTML5Chart.ClientChart functions (continued)

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

## addSeries

**Syntax** void ClientChart.addSeries(string seriesName, Array values)

Adds a data series to this ClientChart.

**Parameters** **seriesName**

String. A name for the series.

**values**

Array. The values for the series, defining X and Y value pairs.

**Example** This example adds the monthly revenue series as an array of numbers:

```
myClientChart.addSeries('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
 4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

## getCategoryCount

**Syntax** integer ClientChart.getCategoryCount( )

Returns the number of categories in this ClientChart.

**Returns** Integer. The number of categories.

**Example** This example displays the number of categories in myClientChart as an alert:

```
alert("This HTML5 client chart has" +
 myClientChart.getCategoryCount() + "categories.");
```

## getChartHeight

**Syntax** integer ClientChart.getChartHeight( )

Returns the height of this ClientChart in pixels.

**Returns** Integer. The height of the chart in pixels.

**Example** This example displays the height of myClientChart as an alert:

```
alert("Height: " + myClientChart.getHeight());
```

## getChartWidth

**Syntax** integer ClientChart.getChartWidth( )

Returns the width of this ClientChart in pixels.

**Returns** Integer. The width of the chart in pixels.

**Example** This example displays the width of myClientChart as an alert:

```
alert("Width: " + myClientChart.getChartWidth());
```

## getClientOptions

**Syntax** actuate.report.HTML5Chart.ClientOption ClientChart.getClientOptions( )

Returns the ClientOptions set for this ClientChart.

**Returns** actuate.report.HTML5Chart.ClientOption object. The client options.

**Example** This example retrieves the client options for myClientChart and stores them in the myClientOptions variable:

```
var myClientOptions = myClientChart.getClientOptions();
```

## getCore

**Syntax** actuate.report.HTML5Chart.Highcharts ClientChart.getCore( )

Returns the Highcharts object contained in this ClientChart.

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

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

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

## getSeriesCount

**Syntax** integer ClientChart.getSeriesCount( )

Returns the number of run-time series in this ClientChart.

**Returns** Integer. The number of series.

**Example** This example displays the number of run-time series in myClientChart as an alert:

```
alert("Runtime Series: " + myClientChart.getSeriesCount());
```

## getXAxisMax

**Syntax** float ClientChart.getXAxisMax( )

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

## actuate.report.HTML5Chart.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.getAxisMax());
```

## getAxisMin

**Syntax** float ClientChart.getAxisMin( )

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.getAxisMin());
```

## getAxisMax

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

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

**Parameter** **axisIndex**

Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

**Returns** Float. The axis series' maximum.

**Example** This example displays the maximum value of the series associated with the Y-axis in myClientChart as an alert:

```
alert("Max for Y-axis series: " + myClientChart.getAxisMax());
```

## getAxisMin

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

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

**Parameter** **axisIndex**

Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

**Returns** Float. The axis series' minimum.

**Example** This example displays the minimum value of the series associated with the Y-axis in myClientChart as an alert:

```
alert("Min for Y-axis series: " + myClientChart.getYAxisMin());
```

## isChartWithAxes

**Syntax** boolean ClientChart.isChartWithAxes()

Returns whether this chart has axes.

**Returns** Boolean. True indicates axes, false otherwise.

**Example** This example displays whether myClientChart has axes:

```
alert("Chart has axes: " + myClientChart.isChartWithAxes());
```

## redraw

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

Redraws this ClientChart with options.

**Parameter** **chartOptions**

actuate.report.HTML5Chart.ClientOption object. Optional. The chart options.

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

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

## removeSeries

**Syntax** void ClientChart.removeSeries(string seriesName, boolean redraw)

Removes a series by name.

**Parameters** **seriesName**

String. The name of the series to remove.

**redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**Example** This example removes the series monthly revenue from myClientChart and redraws the chart:

```
myClientChart.removeSeries('monthly revenue', true);
```

## setSeriesVisible

**Syntax** void ClientChart.setSeriesVisible(string seriesName, boolean visible)

Makes a series visible.

## actuate.report.HTML5Chart.ClientChart

### Parameters **seriesName**

String. The name of the series to change.

### **visible**

Boolean. Optional. True indicates visible. Default is true.

**Example** This example sets the series monthly revenue as visible for myClientChart:

```
myClientChart.setSeriesVisible('monthly revenue', true);
```

## **setTitle**

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

Sets the title of this ClientChart.

### Parameter **title**

String. Chart title text.

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

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

## **setValues**

### **Syntax** void ClientChart.setValues(string seriesName, float[ ] values, boolean redraw)

Sets the value for a series.

### Parameters **seriesName**

String. Name of the series to change.

### **values**

Array of float. The values for the series, defining X and Y value pairs.

### **redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**Example** This example adds the monthly revenue series as an array of numbers:

```
myClientChart.setValues('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

## **setXAxisRange**

### **Syntax** void ClientChart.setXAxisRange(float min, float max, boolean redraw)

Sets the value range for the X-axis.

### Parameters **min**

Float. A new minimum value.

**max**

Float. A new maximum value.

**redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**Example** This example sets the X-axis range to 1 through 3 and redraws the chart:

```
myClientChart.setXAxisRange(1, 3);
```

## setYAxisRange

**Syntax** `void ClientChart.setYAxisRange(float min, float max, boolean redraw, integer axisIndex)`

Sets the value range for the Y-axis.

**Parameters** **min**

Float. A new minimum value.

**max**

Float. A new maximum value.

**redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**axisIndex**

Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

**Example** This example sets the Y-axis range to 0 through 15 and redraws the chart:

```
myClientChart.setYAxisRange(0, 15);
```

---

## Class actuate.report.HTML5Chart.ClientOption

**Description** A container for a ClientOption element in a report. ClientOption provides functions to change ClientChart features, such as orientation, type, and title.

### Constructor

**Syntax** void actuate.report.HTML5Chart.ClientOption( )

Generates a new ClientOption object to manage the chart options for a ClientChart.

### Function summary

Table 7-31 lists actuate.report.HTML5Chart.ClientOption functions.

**Table 7-31** actuate.report.HTML5Chart.ClientOption functions

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

### addSeries

**Syntax** void ClientOption.addSeries(string seriesName, float[ ] values)

Adds a data series to this ClientOption.

**Parameters** **seriesName**

String. A name for the series.

**values**

Array of float. The values for the series, defining X and Y value pairs.

**Example** This example adds the monthly revenue series as an array of numbers:

```
myClientOption.addSeries('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
 4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

## explodePieSlice

**Syntax** void ClientOption.explodePieSlice(string categoryName, boolean sliced)

Explodes the specified pie chart's slice.

**Parameters** **categoryName**

String. The name of a category.

**sliced**

Boolean. Optional. True means the chart is sliced. Default is true.

**Example** This example explodes the Q1 category from a chart with myClientOption:

```
myClientOption.explodePieSlice('Q1');
```

## isChartWithAxes

**Syntax** boolean ClientChart.isChartWithAxes( )

Returns whether this chart has axes.

**Returns** Boolean.

**Example** This example displays whether myClientOption has axes:

```
alert("Options has axes: " + myClientOption.isChartWithAxes());
```

## pivotChart

**Syntax** void ClientChart.pivotChart( )

Switches the axes of the chart, if the chart has axes.

**Example** This example switches the axes in myClientOption and then redraws myClientChart with the switched axes:

```
var myClientOption = myClientChart.getClientOption();
myClientOption.pivotChart();
myClientChart.redraw(myClientOption);
```

## setChartType

**Syntax** void ClientOption.setChartType(string chartType, boolean isCurve)

Sets the chart type in this ClientOption.

**Parameters** **chartType**

String. The chart type. Valid values are line, area, bar, scatter, and pie.

**isCurve**

Boolean. Optional. Indicates if line or area chart is curve. Default value is false.

**Example** This example changes the chart type to pie in myClientOption:

```
myClientOption.setChartType('pie');
```

## setSeriesVisible

**Syntax** void ClientOption.setSeriesVisible(string seriesName, boolean visible)

Makes a series visible.

**Parameters** **seriesName**

String. The name of the series to change.

**visible**

Boolean. Optional. Default is true.

**Example** This example sets the series months as visible for myClientOption:

```
myClientOption.setSeriesVisible('monthly revenue', true);
```

## setTitle

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

Sets the title of this ClientOption.

**Parameter** **title**

String. Chart title text.

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

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

## setXAxisTitle

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

Sets the X-axis title of this ClientOption.

**Parameter** **title**

String. X-axis title text.

**Example** This example sets the title of the X-axis in myClientOption to 'Month':

```
myClientOption.setXAxisTitle('Month');
```

## setYAxisTitle

**Syntax** void ClientOption.setTitle(string title, integer ChartOptions)

Sets the Y-axis title of this ClientOption.

**Parameters** **title**

String. Y-axis title text.

**chartOptions**

Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

**Example** This example sets the title of the Y-axis in myClientOption to 'Dollars, in millions':

```
myClientOption.setYAxisTitle('Dollars, in millions');
```

---

## Class actuate.report.HTML5Chart.ClientPoint

**Description** Represents a data point in a chart. ClientPoint provides functions to manage a point in a series on an individual basis, including selections, options, and events. The options for ClientPoint are defined in the Highcharts point class, which is documented at the following URL:

<http://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 7-32 lists actuate.report.HTML5Chart.ClientPoint functions.

**Table 7-32** actuate.report.HTML5Chart.ClientPoint functions

| Function        | Description                         |
|-----------------|-------------------------------------|
| applyOptions( ) | Changes the point values or options |
| destroy( )      | Destroys a point to clear memory    |
| remove( )       | Removes a point                     |
| select( )       | Toggles the selection of a point    |
| remove( )       | Updates the point with new options  |

### applyOptions

**Syntax** void ClientPoint.applyOptions({float | object} options)

Applies the options containing the x and y data and possibly some extra properties. This is called on point initialization or from point.update.

**Parameter** **options**

Float, array of float, or object. The point options. If options is a single number, the point gets that number as the Y value. If options is an array, the point gets the first two numbers as an X and Y value pair. If options is an object, advanced options as outlined in the Highcharts options.point are applied. The fields include color, events, id, marker, legend, Index (pie chart only), name, sliced (pie chart only), x, and y.

**Example** This example changes the Y value of myClientPoint to 12:

```
myClientPoint.applyOptions(12);
```

## destroy

**Syntax** void ClientPoint.destroy( )

Destroys a point to clear memory. Its reference still stays in series.data.

**Example** This example destroys the options and values for myClientPoint:

```
myClientPoint.destroy();
```

## remove

**Syntax** void ClientPoint.remove(boolean redraw, {boolean | object} animation)

Removes this point and optionally redraws the series and axes.

**Parameters** **redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**animation**

Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

**Example** This example removes myClientPoint from a series, and redraws the chart with animation to display the changed series:

```
myClientPoint.remove();
```

## select

**Syntax** void ClientPoint.select(boolean selected, boolean accumulate)

Selects this point.

**Parameters** **selected**

Boolean. Specifies whether to select or deselect the point.

**accumulate**

Boolean. Whether to add this point to the previous selection. By default, this is true when the Ctrl (PC) or Cmd (Macintosh) key is held during selection.

**Example** This example selects MyClientPoint and deselects all other points:

```
myClientPoint.select(true, false);
```

## remove

**Syntax** void ClientPoint.remove(boolean redraw, {boolean | object} animation)

Updates this point and optionally redraws the series and axes.

**Parameters** **redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

## actuate.report.HTML5Chart.ClientPoint

### **animation**

Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

**Example** This example removes myClientPoint and redraws the chart:

```
myClientPoint.remove();
```

## **update**

**Syntax** void ClientPoint.update({float|float[ ]|object} options, boolean redraw, {boolean | object} animation)

Updates this point and optionally redraws the series and axes.

**Parameters** **options**

Float, array of float, or object. The point options. If options is a single number, the point gets that number as the Y value. If options is an array, the point gets the first two numbers as an X and Y value pair. If options is an object, advanced options as outlined in the Highcharts options.point are applied. The fields include color, events, id, marker, legend, Index (pie chart only), name, sliced (pie chart only), x, and y.

### **redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

### **animation**

Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

**Example** This example updates myClientPoint with an X value of 1 and a Y value of 12, then redraws the point:

```
myClientPoint.update([1,12]);
```

---

## Class actuate.report.HTML5Chart.ClientSeries

**Description** A container for a ClientSeries in a ClientChart. ClientSeries provides functions to manage a series and the graph of that series. In the ClientSeries object, all the points are accessible from the ClientSeries.data array.

### Constructor

**Syntax** void actuate.report.HTML5Chart.ClientSeries( )

Generates a new ClientSeries object to manage a series for a ClientChart.

### Function summary

Table 7-33 lists actuate.report.HTML5Chart.ClientSeries functions.

**Table 7-33** 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                            |

### 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

## actuate.report.HTML5Chart.ClientSeries

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);
```

## **cleanData**

**Syntax** void ClientSeries.cleanData()

Sorts the series and removes duplicate points or values.

**Example** This example sorts myClientSeries and removes its duplicate points and values:

```
myClientSeries.cleanData();
```

## **destroy**

**Syntax** void ClientSeries.destroy()

Clears DOM series objects and frees memory.

**Example** This example clears the memory of myClientSeries and its member objects:

```
myClientSeries.destroy();
```

## **hide**

**Syntax** void ClientSeries.hide()

Hides the graph of this series.

**Example** This example hides myClientSeries graph from the chart:

```
myClientSeries.hide();
```

## **redraw**

**Syntax** void ClientSeries.redraw()

Redraws the graph of this series after updating the data and axes.

**Example** This example redraws the graph of myClientSeries:

```
myClientSeries.redraw();
```

## remove

**Syntax** void ClientSeries.remove(boolean redraw, {boolean | object} animation)

Removes this series and optionally redraws the chart.

**Parameters**

### redraw

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

### animation

Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

**Example** This example removes the graph of myClientSeries from the chart:

```
myClientSeries.remove();
```

## render

**Syntax** void ClientSeries.render()

Renders the graph of this series and its markers.

**Example** This example renders the graph of myClientSeries to the chart:

```
myClientSeries.render();
```

## select

**Syntax** void ClientSeries.select(boolean selected)

Selects this series.

**Parameter**

### selected

Boolean. Optional. Specifies whether to select or deselect the series. If undefined, toggles selection.

**Example** This example selects myClientSeries:

```
myClientSeries.select(true);
```

## setData

**Syntax** void ClientSeries.setData({float | object}[ ] data, boolean redraw)

Replaces the series data with a new set of data.

**Parameters** **data**

Array of float and/or object. An array of data points for the series. The points can be given in three ways:

- 1 A list of numerical values, which are assigned as Y values, paired with X values starting with 0 and incrementing by 1 for each additional number.  
For example:  
`[0, 5, 3, 5]`
- 2 A list of arrays with two values, which are assigned as X and Y value pairs. If the first value is a string, it is applied as the name of the point, and the x value is incremented following the above rules. For example:  
`[[4, 2], [6, 3], [8, 2]]`
- 3 A list of objects with named values, which are assigned to points using the Highcharts point configuration specification options.point. For example:

```
[{name: 'Point 1',
 color: '#00FF00',
 y: 0
},
{name: 'Point 2',
 color: '#FF00FF',
 y: 5
}]
```

**redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**Example** This example replaces the points in myClientSeries with three new points:

```
myClientSeries.setData([[4, 2], [6, 3], [8, 2]]);
```

## setVisible

**Syntax** `void ClientSeries.setVisible(boolean vis, boolean redraw)`

Sets the visibility of this series.

**Parameters** **vis**

Boolean. Optional. Specifies whether to display the series. True displays the series, false hides it. If no value is provided, the visibility changes to false if visibility is true, and true if visibility is false.

**redraw**

Boolean. Optional. Specifies whether to redraw the chart. Default is true.

**Example** This example sets myClientSeries to visible and redraws it:

```
myClientSeries.setVisible(true);
```

## show

**Syntax** void ClientSeries.show( )

Displays the graph of this series.

**Example** This example displays the graph of myClientSeries:

```
myClientSeries.show();
```

## Class actuate.report.HTML5Chart.Highcharts

**Description** A container for a Highcharts element in a ClientChart. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

[http://www.actuate.com/documentation/R11SP4/actuatebirt  
/highcharts/Highcharts-Options-Reference.htm](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 Highcharts for a ClientChart.

---

## Class actuate.report.HTML5Chart.Renderer

**Description** A container for a Highcharts renderer object. Directly accesses the Highcharts rendering layer to draw primitive shapes like circles, rectangles, paths or text directly. The renderer represents a wrapper object for SVG in modern browsers and VML in older versions of Microsoft Internet Explorer.

### Constructor

**Syntax** `void actuate.report.HTML5Chart.Renderer()`

Generates a new Renderer object to manage the Highcharts rendering options for a ClientChart.

### Function summary

Table 7-34 lists actuate.report.HTML5Chart.Renderer functions.

**Table 7-34** actuate.report.HTML5Chart.Renderer functions

| Function                | Description                                        |
|-------------------------|----------------------------------------------------|
| <code>arc()</code>      | Draws and returns an arc                           |
| <code>circle()</code>   | Draws a Scalable Vector Graphic circle             |
| <code>clipRect()</code> | Defines a clipping rectangle                       |
| <code>destroy()</code>  | Destroys the renderer and its allocated members    |
| <code>g()</code>        | Creates a group                                    |
| <code>image()</code>    | Displays an image                                  |
| <code>path()</code>     | Draws a path                                       |
| <code>rect()</code>     | Draws and returns a rectangle                      |
| <code>setSize()</code>  | Resizes the box and re-aligns all aligned elements |
| <code>text()</code>     | Adds text to the Scalable Vector Graphic object    |

### arc

**Syntax** `object Renderer.arc(integer x, integer y, integer r, integer innerR, float start, float end)`

Generates and draws an arc on the chart.

**Parameters** `x`

Integer. The X position of the arc's center, measured in pixels from the left edge of the rendering area.

**y**

Integer. The Y position of the arc's center, measured in pixels from the top edge of the rendering area.

**r**

Integer. The outer radius, measured in pixels.

**innerR**

Integer. The inner radius, measure in pixels.

**start**

Float. The starting angle of the arc, measured in radians, where 0 is directly right and -Math.PI/2 is directly upward. The arc is drawn clockwise from start to end.

**end**

Float. The ending angle of the arc, measured in radians, where 0 is directly right and -Math.PI/2 is directly upward.

**Returns** Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

<http://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);
```

**circle**

**Syntax** object Renderer.circle(integer x, integer y, integer r)

Generates and draws a Scalable Vector Graphic circle on the chart.

**Parameters****x**

Integer. The X position of the circle's center, measured in pixels from the left edge of the rendering area.

**y**

Integer. The Y position of the circle's center, measured in pixels from the top edge of the rendering area.

**r**

Integer. The radius, measured in pixels.

**Returns** Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

- Example** This example draws a circle with a center 200 pixels from the left edge and 150 pixels from the top edge of the chart area:

```
myRenderer.circle(200, 150, 100);
```

## clipRect

**Syntax** object Renderer.clipRect(string id, integer x, integer y, integer width, integer height)

Generates and draws a clipping rectangle on the chart.

**Parameters**

- id**  
String. A string to identify the element.

### x

Integer. The X position of the rectangle's upper left corner, measured in pixels from the left edge of the rendering area.

### y

Integer. The Y position of the rectangle's upper left corner, measured in pixels from the top edge of the rendering area.

### width

Integer. The width, in pixels.

### height

Integer. The height, in pixels.

**Returns** Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

- Example** This example draws a 100-pixel-by-100-pixel rectangle 100 pixels from the left and top edges of chart area:

```
myRenderer.cliprect('myClipRect', 100, 100, 100, 100);
```

## destroy

**Syntax** void Renderer.destroy( )

Destroys this renderer and its allocated elements.

- Example** This example destroys the myRenderer object and frees its memory:

```
myRenderer.destroy();
```

## g

**Syntax** object Renderer.g(string name)

Adds an SVG/VML group to the Renderer object.

**Parameter** **name**

String. The name of the group. Used in the class name, which will be "highcharts-"+ name. Other Element objects are added to the group by using this group as the first parameter in .add( ) for the element wrappers.

**Returns**

Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

**Example** This example creates a new group called myGroup:

```
myRenderer.g('myGroup');
```

## image

**Syntax** `object Renderer.image(string src, integer x, integer y, integer width, integer height)`

Generates and draws a image on the chart.

**Parameters** **src**

String. A URL for the image.

**x**

Integer. The X position of the image's upper left corner, measured in pixels from the left edge of the rendering area.

**y**

Integer. The Y position of the image's upper left corner, measured in pixels from the top edge of the rendering area.

**width**

Integer. The width, in pixels.

**height**

integer. The height, in pixels.

**Returns**

Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

**Example**

This example adds the sun.png image to the chart 100 pixels from the left and top of the edge of the chart:

```
myRenderer.image('http://highcharts.com/demo/gfx/sun.png', 100, 100, 30, 30);
```

## path

**Syntax** `object Renderer.path(object[ ] path)`

Adds a path to the renderer based on SVG's path commands. In SVG-capable browsers, all path commands are supported, but in VML only a subset is supported, including the moveTo, lineTo, and curve commands.

- Parameter** **path**  
Array of string and integer objects. An SVG path with attributes split up in array form.
- Returns** Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.
- Example** This example draws a path from the upper left corner of the rendering area (0, 0) to the points (100, 100), (200, 50), and (300, 100), where the first number represents the distance from the left edge of the rendering area and the second number represents the distance from the top edge of the rendering area:

```
myRenderer.path(['M', 0, 0, 'L', 100, 100, 200, 50, 300, 100]);
```

## rect

**Syntax** object Renderer.rect(integer x, integer y, integer width, integer height, integer r, integer strokeWidth)

Generates and draws a rectangle on the chart.

- Parameters**
- x**  
Integer. The X position of the rectangle's upper left corner, measured in pixels from the left edge of the rendering area.
  - y**  
Integer. The Y position of the rectangle's upper left corner, measured in pixels from the top edge of the rendering area.
  - width**  
Integer. The width, in pixels.
  - height**  
Integer. The height, in pixels.
  - r**  
Integer. The corner radius, measured in pixels.
  - strokeWidth**  
Integer. Stroke measurement to support crisp drawing.
- Returns** Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

- Example** This example draws a 100-pixel-by-100-pixel rectangle 100 pixels from the left and top edges of chart area with 5-pixel-radius quarter-circles as edges:

```
myRenderer.rect(100, 100, 100, 100, 5);
```

## setSize

- Syntax** void Renderer.setSize(integer width, integer height, boolean animate)  
Resizes the rendering area and re-aligns all aligned elements.

- Parameters**
- width**  
Integer. The width, in pixels.

- height**  
Integer. The height, in pixels.

- animate**  
Boolean. Optional. Whether to animated the resize. Default is true.

- Example** This example resizes the renderer area to 500 pixels by 500 pixels:

```
myRenderer.setSize(500, 500);
```

## text

- Syntax** object Renderer.text(string str, integer x, integer y, boolean useHTML)  
Adds text to the Scalable Vector Graphic object.

- Parameters**
- str**  
String. The text in this text element.

- x**  
Integer. The X position of the text's lower left corner, measured in pixels from the left edge of the rendering area.

- y**  
Integer. The Y position of the text's lower left corner, measured in pixels from the top edge of the rendering area.

- useHTML**  
Boolean. Specifies whether to use HTML to render the text.

- Returns** Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help.

- 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 7-35 lists actuate.report.Label functions.

**Table 7-35** actuate.report.Label functions

| Function         | Description                                            |
|------------------|--------------------------------------------------------|
| getBookmark()    | Returns the bookmark name for this Label               |
| getHtmlDom()     | Returns the HTML element for this Label                |
| getInstanceId()  | Returns the report element instance id                 |
| getLabel()       | Returns the text of this Label element                 |
| getPageContent() | Returns the page content to which this element belongs |
| getType()        | Returns the Label's element type                       |
| hide()           | Hides this element                                     |
| show()           | Shows this element                                     |
| submit()         | Applies changes made to this gadget                    |

### getBookmark

**Syntax** string Label.getBookmark( )

Returns the bookmark name for this Label.

**Returns** String. The Label's bookmark.

**Example** This example displays the Label's bookmark in an alert box:

```
alert(myLabel.getBookmark());
```

### getHtmlDom

**Syntax** HTMLElement Label.getHtmlDom( )

## actuate.report.Label

Returns the HTML element for this Label.

**Returns** HTMLElement.

**Example** This example displays the HTML DOM element for this Label inside a red border

```
function showHtmlDom(myLabel) {
 var domNode = myLabel.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** string Label.getInstanceId()

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```
function showID(myLabel) {
 var elementID = myLabel.getInstanceId();
 alert (elementID);
}
```

## getLabel

**Syntax** string Label.getLabel()

Returns the text of this Label element.

**Returns** String. The Label text.

**Example** This example displays the text of the myLabel object in an alert box:

```
alert("Label element text is " + myLabel.getLabel());
```

## getPageContent

**Syntax** actuate.viewer.PageContent Label.getPageContent()

Returns the page content to which this Label belongs.

**Returns** actuate.viewer.PageContent. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```

function showViewID(myLabel) {
 var pageContent = myLabel.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}

```

## getType

**Syntax** string Label.getType( )

Returns the report element type of this object, which is Label.

**Returns** String. "Label".

**Example** This example checks the report element type and displays an alert if the type is not "Label":

```

if (myElement.getType() != "Label") {
 alert("Type mismatch, report element type is not Label")
}

```

## hide

**Syntax** void Label.hide( )

Hides this element.

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

```
myLabel.hide();
```

## show

**Syntax** void Label.show( )

Shows this element.

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

```
myLabel.show();
```

## submit

**Syntax** void Label.submit(function callback)

Submits all the asynchronous operations for this Label. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the label container.

**Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

`actuate.report.Label`

**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 7-36 lists actuate.report.Table functions.

**Table 7-36** actuate.report.Table functions

| Function         | Description                                                                                                                    |
|------------------|--------------------------------------------------------------------------------------------------------------------------------|
| clearFilters()   | Clears the filters from the given column                                                                                       |
| getBookmark()    | Returns the bookmark name for this Table                                                                                       |
| getColumn()      | Gets the Table data by column index and returns only the data from the current visible page                                    |
| getHtmlDom()     | Returns the HTML element for this Table                                                                                        |
| getInstanceId()  | Returns the report element instance id                                                                                         |
| getPageContent() | Returns the page content to which this element belongs                                                                         |
| getRow()         | Gets the Table data by row index                                                                                               |
| getType()        | Returns the report element type                                                                                                |
| groupBy()        | Adds an inner group to this Table                                                                                              |
| hide()           | Hides this element                                                                                                             |
| hideColumn()     | Hides a Table column by specifying the column name                                                                             |
| hideDetail()     | Hides detailed information for displayed groups                                                                                |
| removeGroup()    | Removes an inner group                                                                                                         |
| setFilters()     | Applies filters to this Table                                                                                                  |
| setSorters()     | Adds sorters to this Table                                                                                                     |
| show()           | Shows this element                                                                                                             |
| showColumn()     | Shows a Table column by specifying the column name                                                                             |
| showDetail()     | Shows detailed information for displayed groups                                                                                |
| submit()         | Submits all the asynchronous operations that the user has requested on this report and renders the Table component on the page |
| swapColumns()    | Swaps two columns, reordering the columns                                                                                      |

## clearFilters

**Syntax** void Table.clearFilters(string columnName)

Clears the filters of a given column.

**Parameter** **columnName**

String. The name of the column.

**Example** This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(myTable) {
 myTable.clearFilters("PRODUCTLINE");
 myTable.submit();
}
```

## getBookmark

**Syntax** string Table.getBookmark( )

Returns the Table's name.

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

**Example** This example displays the Table's bookmark in an alert box:

```
function alertBookmark(myTable) {
 alert(myTable.getBookmark());
}
```

## getColumn

**Syntax** array[ ] Table.getColumn(integer columnIndex)

Gets the Table data by column index. Returns the data from the current visible page.

**Parameter** **columnIndex**

Integer. Optional. The numerical index of the column from which to retrieve data. The getColumn() function returns the values for the first column when no value is provided for columnIndex.

**Returns** Array. A list of data in the format of the column.

**Example** This example returns the first column in myTable:

```
function getMyColumn(myTable) {
 return myTable.getColumn();
}
```

## getHtmlDom

**Syntax** `HTMLElement Table.getHtmlDom( )`

Returns the Table's name.

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

**Example** This example displays the HTML DOM element for this Table inside a red border:

```
function showHtmlDom(myTable) {
 var domNode = myTable.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** `string Table.getInstanceId( )`

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```
function showID(myTable) {
 var elementID = myTable.getInstanceId();
 alert (elementID);
}
```

## getPageContent

**Syntax** `actuate.viewer.PageContent Table.getPageContent( )`

Returns the page content to which this Table belongs.

**Returns** `actuate.viewer.PageContent`. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myTable) {
 var pageContent = myTable.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getRow

**Syntax** array[ ] Table.getRow(integer rowIndex)

Gets the Table data by row index. Returns the data from the current visible page.

**Parameter** **rowIndex**

Integer. Optional. The numerical index of the row from which to retrieve data. The `getRow()` function returns the values for the first row when no value for `rowIndex` is provided.

**Returns** Array. A list of data in the format of the columns that cross the row.

**Example** This example retrieves the first row in `myTable`:

```
function getMyRow(myTable) {
 return myTable.getRow();
}
```

## getType

**Syntax** string Table.getType( )

Returns the report element type of this object, which is `Table`.

**Returns** String. "Table".

**Example** This example returns the report element type of this object in an alert box:

```
function getTableType(myTable) {
 alert("Element type is: " + myTable.getType());
}
```

## groupBy

**Syntax** void Table.groupBy(string columnName)

Groups the data in a table by the values in a given column. If there is an existing group, this operation will add the new group after the existing group.

**Parameter** **columnName**

String. The name of the column to use for the innermost group to the Table.

**Example** This example groups the data in `myTable` by the values in the `TOTAL` column:

```
function groupByColumn(myTable) {
 myTable.groupBy("TOTAL");
}
```

## hide

**Syntax** void Table.hide( )

Hides this element.

**Example** This example hides myTable:

```
myTable.hide();
```

## hideColumn

**Syntax** void Table.hideColumn(string columnName)

Hides a table column by specifying the column name.

**Parameter** **columnName**

String. The data binding name for the column to hide.

**Example** This example hides the TOTAL column from myTable:

```
function myHiddenColumn(myTable) {
 myTable.hideColumn("TOTAL");
 myTable.submit();
}
```

## hideDetail

**Syntax** void Table.hideDetail(string columnName)

Hides information for a column from the grouped data displayed on the page. If every column is hidden, only the group name is visible.

**Parameter** **columnName**

String. The data binding name for the column to hide.

**Example** This example hides the TOTAL column from the grouped data visible for myTable:

```
function hideMyDetail(myTable) {
 myTable.hideDetail("TOTAL");
 myTable.submit();
}
```

## removeGroup

**Syntax** void Table.removeGroup()

Removes the innermost group.

**Example** This example removes the innermost group from myTable and displays an alert box after calling submit( ):

```
function removeMyGroup(myTable) {
 myTable.removeGroup();
 myTable.submit(alert("Group removed"));
}
```

## setFilters

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

Applies a filter or filters to this Table element.

|                   |                                                                                                                                                                                                            |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <b>filter</b><br>actuate.data.Filter object. A single filter condition to apply to this Table.<br><br><b>filters</b><br>An array of actuate.data.Filter objects. Filter conditions to apply to this Table. |
| <b>Example</b>    | To add a filter to the Table to display only entries with a CITY value of NYC, use the following code:                                                                                                     |

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

## setSorters

**Syntax** void Table.setSorters(actuate.data.Sorter sorter)  
void Table.setSorters(actuate.data.Sorter[ ] sorters)

Applies a sorter or sorters to this Table.

|                   |                                                                                                                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <b>sorter</b><br>actuate.data.Sorter object. A single sort condition to apply to this Table.<br><br><b>sorters</b><br>An array of actuate.data.Sorter objects. Sort conditions to apply to this Table. |
| <b>Example</b>    | This example adds the myStateSorter and myCitySorter sorters to myTable:                                                                                                                               |

```
function setAllMySorters(myTable) {
 myTable.setSorters(["myStateSorter", "myCitySorter"]);
}
```

## show

**Syntax** void Table.show( )

Shows this element.

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

```
myTable.show();
```

## showColumn

**Syntax** void Table.showColumn(string columnName)

Shows the Table column by specifying the column name.

**Parameter** **enabled**

String. The data binding name for the column to display.

**Example** This example shows the PRODUCTLINE column in myTable:

```
function showMyColumn(myTable) {
 myTable.showColumn("PRODUCTLINE");
 myTable.submit();
}
```

## showDetail

**Syntax** void Table.showDetail(string columnName)

Displays information for a column from the grouped data displayed on the page. If every column is hidden, only the group name is visible.

**Parameter** **columnName**

String. The data binding name for the column to display.

**Example** This example shows the information from the PRODUCTLINE column in the grouped data that is displayed for myTable:

```
function showMyDetail(myTable) {
 myTable.showDetail("PRODUCTLINE");
 myTable.submit();
}
```

## submit

**Syntax** void Table.submit(function callback)

Submits all the asynchronous operations for this Table element. The submit() function triggers an AJAX request to submit all the asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the table container.

**Parameter** **callback**

Function. The function called after the asynchronous call processing finishes.

**Example** This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(myTable) {
 myTable.clearFilters("PRODUCTLINE");
```

```
actuate.report.Table
```

```
 myTable.submit(alert("Filters Cleared"));
}
```

## swapColumns

**Syntax** void Table.swapColumns(string columnName1, string columnName2)

Swaps the columns to reorder to column sequence of the Table.

**Parameters** **columnName1**

String. The first column to swap in the column order.

**columnName2**

String. The second column to swap in the column order.

**Example** This example swaps the TOTAL and PRODUCTLINE columns in myTable:

```
function swapMyColumns (myTable) {
 myTable.swapColumns ("TOTAL", "PRODUCTLINE");
 myTable.submit ();
}
```

---

## Class actuate.report.TextItem

**Description** A container for a Text element in a report. TextItem provides functions to operate on a Text element, such as retrieving the text value and getting the HTML DOM element from the report Text element.

### Constructor

The TextItem object is constructed by viewer.PageContent.getTextByBookmark( ).

### Function summary

Table 7-37 lists actuate.report.TextItem functions.

**Table 7-37** actuate.report.TextItem functions

| Function         | Description                                            |
|------------------|--------------------------------------------------------|
| getBookmark()    | Returns the bookmark name for this Text                |
| getHtmlDom()     | Returns the HTML element for this Text                 |
| getInstanceId()  | Returns the report element instance id                 |
| getPageContent() | Returns the page content to which this element belongs |
| getText()        | Returns the text in this Text element                  |
| getType()        | Returns the Text element's type                        |
| hide()           | Hides this element                                     |
| show()           | Shows this element                                     |
| submit()         | Applies changes made to this element                   |

### getBookmark

**Syntax** string TextItem.getBookmark( )

Returns the bookmark name for this Text item.

**Returns** String.

**Example** This example displays the table's bookmark in an alert box:

```
function alertBookmark(myTextItem) {
 alert(myTextItem.getBookmark());
}
```

## getHtmlDom

**Syntax** `HTMLElement TextItem.getHtmlDom( )`

Returns the HTML element for this Text.

**Returns** `HTMLElement`.

**Example** This example displays the HTML DOM element for this Text item inside a red border:

```
function showHtmlDom(myTextItem) {
 var domNode = myTextItem.getHtmlDom();
 var box = document.createElement('div');
 box.style.border = '2px solid red';
 var label = document.createElement('h2');
 label.innerHTML = 'The HTML DOM:';
 box.appendChild(label);
 box.appendChild(domNode);
 document.body.appendChild(box);
}
```

## getInstanceId

**Syntax** `string TextItem.getInstanceId( )`

Returns the instance id of this report element.

**Returns** String. The instance id.

**Example** This example displays the instance ID of the report element in an alert box:

```
function showID(myTextItem) {
 var elementID = myTextItem.getInstanceId();
 alert (elementID);
}
```

## getPageContent

**Syntax** `actuate.viewer.PageContent TextItem.getPageContent( )`

Returns the page content to which this Text belongs.

**Returns** `actuate.viewer.PageContent`. report content.

**Example** This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myTextItem) {
 var pageContent = myTextItem.getPageContent();
 var pageViewerID = pageContent.getViewerId();
 alert (pageViewerID);
}
```

## getText

**Syntax** `string TextItem.getText( )`

Returns the text of this Text element.

**Returns** String. The content text.

**Example** This example displays the text of the myTextItem object in an alert box:

```
alert("Text content for myTextItem is " + myTextItem.getText());
```

## getType

**Syntax** `string TextItem.getType( )`

Returns the report element type of this object, which is Text.

**Returns** String. "Text".

**Example** This example checks the report element type and displays an alert if the type is not "Text":

```
if (myTextItem.getType() != "Text") {
 alert("Type mismatch, report element type is not Text");
}
```

## hide

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

Hides this element.

**Example** This example hides myTextItem:

```
myTextItem.hide();
myTextItem.submit();
```

## show

**Syntax** `void TextItem.show( )`

Shows this element.

**Example** This example shows myTextItem:

```
myTextItem.show();
myTextItem.submit();
```

## submit

**Syntax** `void TextItem.submit(function callback)`

## actuate.report.TextItem

Submits all the asynchronous operations for this TextItem. The submit( ) function triggers an AJAX request for all asynchronous operations. The server returns a response after processing. The results render on the page in the TextItem container.

**Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

**Example** This example uses submit( ) after calling show( ) to show myTextItem:

```
myTextItem.show();
myTextItem.submit();
```

---

## Class actuate.ReportExplorer

**Description** The actuate.ReportExplorer class retrieves and displays a navigable repository or file system interface that enables users to navigate folders and select files. This generic user interface enables the user to browse and select repository contents.

### Constructor

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

Constructs a ReportExplorer object, initializing the ReportExplorer component.

**Parameter** **container**  
String. The name of the HTML element that displays the rendered ReportExplorer component or a container object. The constructor initializes the ReportExplorer component but does not render it.

### Function summary

Table 7-38 lists actuate.ReportExplorer functions.

**Table 7-38** actuate.ReportExplorer functions

| Function                                | Description                                         |
|-----------------------------------------|-----------------------------------------------------|
| <code>getFolderName()</code>            | Gets the root folder name                           |
| <code>getLatestVersionOnly()</code>     | Gets the latestVersionOnly flag                     |
| <code>getResultDef()</code>             | Gets the resultDef value for this GetFolderItems    |
| <code>getSearch()</code>                | Gets the search value for this GetFolderItems       |
| <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 latestVersionOnly flag                     |
| <code> setResultDef()</code>            | Sets the resultDef value for this GetFolderItems    |
| <code>setSearch()</code>                | Sets the search value for this GetFolderItems       |
| <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 |

(continues)

**Table 7-38** actuate.ReportExplorer functions (continued)

| Function           | Description                           |
|--------------------|---------------------------------------|
| showFoldersOnly( ) | Sets the flag to only display folders |
| submit( )          | Applies changes made to this element  |

## getFolderName

**Syntax** string ReportExplorer.getFolderName( )

Returns the name of the root folder for this ReportExplorer.

**Returns** String. The folder name.

**Example** This example displays the root folder's name in an alert box:

```
function alertRootFolder(myReportExplorer) {
 alert(myReportExplorer.getFolderName());
}
```

## getLatestVersionOnly

**Syntax** boolean ReportExplorer.getLatestVersionOnly( )

Returns the latest version only flag for this ReportExplorer.

**Returns** Boolean. True indicates that ReportExplorer displays only the latest version of each report.

**Example** This example displays the latest version only flag in an alert box:

```
function alertLatestVersionFlag(myReportExplorer) {
 alert(myReportExplorer.getLatestVersionOnly());
}
```

## getResultDef

**Syntax** string[ ] ReportExplorer.getResultDef( )

Returns the results definition.

**Returns** Array of strings. Valid values are: "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount".

**Example** This example displays the results definition an alert box:

```
function alertResultsDefinition(myReportExplorer) {
 alert(myReportExplorer.getResultDef());
}
```

## getSearch

**Syntax** `actuate.ReportExplorer.FileSearch ReportExplorer.getSearch( )`

Returns the FileSearch object assigned to this ReportExplorer.

**Returns** `actuate.reportexplorer.FileSearch` object. The file search settings.

**Example** This example sets the FileSearch setting for reportexplorer1 to the FileSearch settings of reportexplorer2:

```
reportexplorer1.setSearch(reportexplorer2.getSearch());
```

## onUnload

**Syntax** `void ReportExplorer.onUnload( )`

Unloads JavaScript variables that are no longer needed by ReportExplorer.

**Example** This example cleans up unused JavaScript variables for myReportExplorer:

```
myReportExplorer.onUnload();
```

## registerEventHandler

**Syntax** `void ReportExplorer.registerEventHandler(string eventName, function handler)`

Registers an event handler to activate for parameter `eventName`. This function can assign several handlers to a single event.

**Parameters** **eventName**

String. Event name to capture.

**handler**

Function. The function to execute when the event occurs. The handler must take two arguments: the ReportExplorer instance that fired the event and an event object specific to the event type.

**Example** This example registers the `errorHandler()` function to respond to the `ON_EXCEPTION` event:

```
myReportExplorer.registerEventHandler(actuate.ReportExplorer
 .EventConstants.ON_EXCEPTION, errorHandler);
```

## removeEventHandler

**Syntax** `void ReportExplorer.removeEventHandler(string eventName, function handler)`

Removes an event handler to activate for parameter `eventName`.

**Parameters** **eventName**

String. Event name to remove from the internal list of registered events.

## actuate.ReportExplorer

### handler

Function. The function to disable.

- Example** This example removes the errorHandler( ) function from responding to the ON\_EXCEPTION event:

```
myReportExplorer.removeEventHandler(actuate.ReportExplorer
 .EventConstants.ON_EXCEPTION, errorHandler);
```

## setContainer

### Syntax

```
void ReportExplorer.setContainer(string containerId)
```

Sets the HTML element container for the ReportExplorer content.

### Parameter

**containerID**

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

- Example** This example sets MyReportExplorer to render the <div> element labeled "History":

```
myReportExplorer.setContainer("History");
```

## setFolderName

### Syntax

```
void ReportExplorer.setFolderName(string folderName)
```

Sets the name of the root folder for this ReportExplorer.

### Parameter

**folderName**

String. The name of the repository folder to use as the root folder. Use a repository path to use subfolders for the root folder. The string '~/' maps to the current user's home folder.

- Example** This example sets the report explorer root folder to /Public:

```
myReportExplorer.setFolderName("/Public");
```

## setLatestVersionOnly

### Syntax

```
void ReportExplorer.setLatestVersionOnly(boolean latestVersionOnly)
```

Sets the latest version only flag for this ReportExplorer.

### Parameter

**latestVersionOnly**

Boolean. True removes all but the latest versions from the report explorer.

- Example** This example sets ReportExplorer to display only the latest versions of all files:

```
myReportExplorer.setLatestVersionOnly(true);
```

## setResultDef

**Syntax** void ReportExplorer.setResultDef(string[ ] resultDef)

Sets the results definition.

**Parameter** **resultDef**

Array of strings. Valid values are: "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount". iHub requires the Name, FileType, and Version fields in the results definition array to identify all files.

**Example** This example sets the result set to five columns of data including name, file type, version, version name, and description:

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

## setSearch

**Syntax** void ReportExplorer.setSearch(actuate.ReportExplorer.FileSearch search)

Assigns a FileSearch object to this ReportExplorer.

**Parameter** **search**

actuate.reportexplorer.FileSearch object. The file search settings.

**Example** This example sets the FileSearch setting for reportexplorer1 to the FileSearch settings of reportexplorer2:

```
reportexplorer1.setSearch(reportexplorer2.getSearch());
```

## setService

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

Sets the target service URL to which this explorer links. When the service URL is not set, this viewer links to the default service URL which is set on the actuate object.

**Parameters** **iPortalURL**

String. The target Actuate web application URL, either a Java Component or iHub Visualization Platform client.

**requestOptions**

actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

**Example** This example sets the URL for the Actuate iPortal web application service:

```
myExplorer.setService("http://127.0.0.1:8700
/iportal", myRequestOptions);
```

## setStartingFolder

**Syntax** void ReportExplorer.setStartingFolder(string strfoldername)

Sets the fully qualified path of the initially selected folder in the explorer tree.

**Parameter** **strfoldername**

String. The fully qualified path of a folder.

**Example** This example sets the initially selected folder to Public in the local repository:

```
myExplorer.setStartingFolder("C:\Program Files\Actuate11\iHub2
\servletcontainer\iportal\WEB-INF\repository\Public");
```

## setUseDescriptionAsLabel

**Syntax** void ReportExplorer.setUseDescriptionAsLabel(boolean useDescription)

Sets the explorer to display the folder description as the folder label instead of the folder name.

**Parameter** **useDescription**

Boolean. True displays descriptions for folders instead of folder names.

**Example** This example displays descriptions for folders instead of folder names:

```
myExplorer.setUseDescriptionAsLabel(true);
```

## showFoldersOnly

**Syntax** void ReportExplorer.showFoldersOnly(boolean flag)

Sets ReportExplorer to display folders but not files.

**Parameter** **flag**

Boolean. True displays folders but not files.

**Example** This example displays folders in ReportExplorer but not files:

```
myExplorer.showFoldersOnly(true);
```

## submit

**Syntax** void ReportExplorer.submit(function callback)

Submits requests to the server for ReportExplorer. When this function is called, an AJAX request is triggered to submit all the operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the ReportExplorer container.

**Parameter** **callback**

Function. The function to execute after the asynchronous call processing is done.

- Example** This example submits ReportExplorer with a root folder that set with setStartingFolder( ) and result definition set with setResultDef( ):

```
myExplorer.setStartingFolder("/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 7-39 lists the constants used for the ReportExplorer class.

**Table 7-39** 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 7-40 lists the ReportExplorer event constants.

**Table 7-40** 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 7-41 lists actuate.reportexplorer.File functions.

**Table 7-41** actuate.reportexplorer.File functions

| Function              | Description                                  |
|-----------------------|----------------------------------------------|
| getAccessType( )      | Gets the accessType value for this File      |
| getDescription( )     | Gets the description value for this File     |
| getFileType( )        | Gets the fileType value for this File        |
| getId( )              | Gets the id value for this File              |
| getName( )            | Gets the name value for this File            |
| getOwner( )           | Gets the owner value for this File           |
| getPageCount( )       | Gets the pageCount value for this File       |
| getSize( )            | Gets the size value for this File            |
| getTimeStamp( )       | Gets the timeStamp value for this File       |
| getUserPermissions( ) | Gets the userPermissions value for this File |
| getVersion( )         | Gets the version value for this File         |
| getVersionName( )     | Gets the versionName value for this File     |
| setAccessType( )      | Sets the accessType value for this File      |
| setDescription( )     | Sets the description value for this File     |
| setFileType( )        | Sets the fileType value for this File        |
| setId( )              | Sets the id value for this File              |
| setName( )            | Sets the name value for this File            |
| setOwner( )           | Sets the owner value for this File           |
| setPageCount( )       | Sets the pageCount value for this File       |
| setSize( )            | Sets the size value for this File            |
| setTimeStamp( )       | Sets the timeStamp value for this File       |

**Table 7-41** actuate.reportexplorer.File functions

| Function              | Description                                  |
|-----------------------|----------------------------------------------|
| setUserPermissions( ) | Sets the userPermissions value for this File |
| setVersion( )         | Sets the version value for this File         |
| setVersionName( )     | Sets the versionName value for this File     |

## getAccessType

**Syntax** string File.getAccessType( )

Gets the access type.

**Returns** String. Either "private" or "shared" according to whether the file has been shared or not.

**Example** To stop a script from running if a file is private, use code similar to the following:

```
if(file.getAccessType() == "private"){ return; }
```

## getDescription

**Syntax** string File.getDescription( )

Gets the description from the file.

**Returns** String. The description.

**Example** To stop a script from running if a file does not have a description, use code similar to the following:

```
if(file.getDescription() == (null || "")){ return; }
```

## getFileType

**Syntax** string File.getFileType( )

Gets the file extension for this File.

**Returns** String. The file type.

**Example** To store the file extension of the File object file in a variable called type, use code similar to the following:

```
var type = file.getFileType();
```

## getId

**Syntax** integer File.getId( )

Gets the file ID value.

## actuate.reportexplorer.File

**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.getId();
```

## getName

**Syntax** string File.getName( )

Gets the name of the file.

**Returns** String. The file name.

**Example** To store the name of the File object file in a variable called name, use code similar to the following:

```
var name = file.getName();
```

## getOwner

**Syntax** string File.getOwner( )

Gets the name of the File's owner.

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

**Example** To store the name of the owner of the File object file in a variable called owner, use code similar to the following:

```
var owner = file.getOwner();
```

## getPageCount

**Syntax** integer File.getPageCount( )

Gets the number pages in the file, if applicable.

**Returns** Integer. The number of pages.

**Example** To halt a script if the number of pages exceeds 100 in the file referenced by the File object largefile, use code similar to the following:

```
if (largefile.getPageCount() > 100) {return;}
```

## getSize

**Syntax** integer File.getSize( )

Gets the size value for this File.

**Returns** Integer.

- Example** To store a File object size in a variable called size, use code similar to the following:

```
var size = file.getSize();
```

## getTimeStamp

- Syntax** string File.getTimeStamp( )

Gets the time stamp for this file.

- Returns** String. A date and time of the file's creation or last modification.

- Example** To store the time stamp for the file referenced by the File object oldfile in a variable called timestamp, use code similar to the following:

```
var timestamp = oldfile.getTimeStamp();
```

## getUserPermissions

- Syntax** string File.getUserPermissions( )

Gets the user permissions.

- Returns** String. The current user's permissions for this file.

- Example** To store a file's permissions in the permissions variable, use code similar to the following:

```
var permissions = file.getUserPermissions();
```

## getVersion

- Syntax** integer File.getVersion( )

Gets the version of the file.

- Returns** Integer. The version.

- Example** To store the file version in the version variable, use code similar to the following:

```
var version = file.getVersion();
```

## getVersionName

- Syntax** string File.getVersionName( )

Gets the version name.

- Returns** String. The version name.

- Example** To store a version name in the version variable, use code similar to the following:

```
var version = file.getVersionName();
```

## setAccessType

**Syntax** void File.setAccessType(string accessType)

Sets the access type.

**Parameter** **accessType**

String. "private" or "shared" indicating whether the file has been shared or not.

**Example** To make a file private, use code similar to the following:

```
file.setAccessType("private")
```

## setDescription

**Syntax** void File.setDescription(string description)

Sets the description from the file.

**Parameter** **description**

String. The description.

**Example** To clear a file's description, use code similar to the following:

```
file.setDescription("") ;
```

## setFileType

**Syntax** void File.setFileType(string fileType)

Sets the file type for this file.

**Parameter** **fileType**

String. The file type, which is a file extension.

**Example** To assign a file's type if none is assigned, use code similar to the following:

```
if (file.getFileType == null) {file.setFileType("txt");}
```

## setId

**Syntax** void File.setId(integer id)

Sets the file ID value.

**Parameter** **id**

Integer. A file ID number.

**Example** To set a file's ID to 42, use code similar to the following:

```
file.setId("42") ;
```

## setName

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

Sets the name of the file.

**Parameter** **name**  
String. The name.

**Example** To set a file's name to releasedates, use code similar to the following:

```
file.setName ("releasedates");
```

## setOwner

**Syntax** void File.setOwner(string owner)

Sets the name of the owner.

**Parameter** **owner**  
String. A user name.

**Example** To set a file's owner to Administrator, use code similar to the following:

```
file.setOwner ("Administrator");
```

## setPageCount

**Syntax** void File.setPageCount(integer pageCount)

Sets the number pages in the file.

**Parameter** **pageCount**  
Integer. The number of pages, which must be less than the current number of pages.

**Example** To set a File object's page to 100 if available, use code similar to the following:

```
if(file.getPageCount () > 100) {file.setPageCount (100);}
```

## setSize

**Syntax** void File.setSize(integer size)

Sets the size of the file.

**Parameter** **size**  
Integer. File size in bytes.

**Example** To set a file's size to 0, use code similar to the following:

```
file.setSize(0);
```

## setTimeStamp

**Syntax** void File.setTimeStamp(string timeStamp)

Sets the time stamp.

**Parameter** **timeStamp**

String. A date and time of the file's creation or last modification.

**Example** To set a file's time stamp to the current time, use code similar to the following:

```
var curruntime = new Date();
file.setTimeStamp(curruntime.toLocaleString());
```

## setUserPermissions

**Syntax** void File.setUserPermissions(string userPermissions)

Sets the user permissions.

**Parameter** **userPermissions**

String. The current user's permissions for this file.

**Example** To apply the user permissions for file1 to file2, use code similar to the following:

```
file2.setUserPermissions(file1.getUserPermissions());
```

## setVersion

**Syntax** void File.setVersion(integer version)

Sets the version of the file.

**Parameter** **version**

Integer. The version.

**Example** To set the file's version to 1 for the first version, use code similar to the following:

```
file.setVersion(1);
```

## setVersionName

**Syntax** void File.setVersionName(string versionName)

Sets the version name.

**Parameter** **versionName**

String. A version name.

**Example** To set a file's version name to 2004, use code similar to the following:

```
file.setVersionName("2004");
```

---

## Class actuate.reportexplorer.FileCondition

**Description** Used 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 7-42 lists actuate.reportexplorer.FileCondition functions.

**Table 7-42** actuate.reportexplorer.FileCondition functions

| Function    | Description                                 |
|-------------|---------------------------------------------|
| getField( ) | Gets the field for this FileCondition       |
| getMatch( ) | Gets the match value for this FileCondition |
| setField( ) | Sets the field for this FileCondition       |
| setMatch( ) | Sets the match value for this FileCondition |

### getField

**Syntax** string FileCondition.getField( )

Returns the field for this FileCondition.

**Returns** String. Possible values are "Name", "FileType", "Description", "PageCount", "Size", "TimeStamp", "Version", "VersionName", and "Owner".

**Example** To store the display field of fcondition in a variable called field, use code similar to the following:

```
var field = fcondition.getField();
```

### getMatch

**Syntax** string FileCondition.getMatch( )

Returns the match value for this FileCondition.

**Returns** String. A string for comparison.

## actuate.reportexplorer.FileCondition

- Example** To store the matching condition of fcondition in a variable called match, use code similar to the following:

```
var match = fcondition.getMatch();
```

### setField

**Syntax** void FileCondition.setField(string field)

Sets the field for the FileCondition.

**Parameter** **field**

String. Possible values are "Name", "FileType", "Description", "PageCount", "Size", "TimeStamp", "Version", "VersionName", and "Owner".

- Example** To set the display field to FileType for fcondition, use code similar to the following:

```
fcondition.setField("FileType");
```

### setMatch

**Syntax** void FileCondition.setMatch(string match)

Sets the match value for the FileCondition.

**Parameter** **match**

String. A string for comparison.

- Example** To set the match value for fcondition to rptdesign, use code similar to the following:

```
fcondition.setField("rptdesign");
```

---

## Class actuate.reportexplorer.FileSearch

**Description** Searches the contents of files according to one or more file conditions. FileSearch represents a JavaScript version of com.actuate.schemas.FileSearch.

### Constructor

**Syntax** `actuate.reportexplorer.FileSearch( )`

Constructs a new FileSearch object.

### Function summary

Table 7-43 lists actuate.reportexplorer.FileSearch functions.

**Table 7-43** actuate.reportexplorer.FileSearch functions

| Function                              | Condition                                              |
|---------------------------------------|--------------------------------------------------------|
| <code>getAccessType()</code>          | Gets the accessType value for this FileSearch          |
| <code>getCondition()</code>           | Gets the condition value for this FileSearch           |
| <code>getConditionArray()</code>      | Gets the ConditionArray value for this FileSearch      |
| <code>getCountLimit()</code>          | Gets the countLimit value for this FileSearch          |
| <code>getDependentFileId()</code>     | Gets the id value for this FileSearch                  |
| <code>getDependentFileName()</code>   | Gets the file name value for this FileSearch           |
| <code>getFetchDirection()</code>      | Gets the fetch direction for this FileSearch           |
| <code>getFetchHandle()</code>         | Gets the fetchHandle value for this FileSearch         |
| <code>getFetchSize()</code>           | Gets the fetchSize value for this FileSearch           |
| <code>getIncludeHiddenObject()</code> | Gets the includeHiddenObject value for this FileSearch |
| <code>getOwner()</code>               | Gets the owner                                         |
| <code>getPrivilegeFilter()</code>     | Gets the privilegeFilter value for this FileSearch     |
| <code>getRequiredFileId()</code>      | Gets the requiredFileId for this FileSearch            |
| <code>getRequiredFileName()</code>    | Gets the requiredFileName value for this FileSearch    |
| <code>setAccessType()</code>          | Sets the accessType value for this FileSearch          |
| <code>setCondition()</code>           | Sets the condition value for this FileSearch           |
| <code>setConditionArray()</code>      | Sets the ConditionArray value for this FileSearch      |
| <code>setCountLimit()</code>          | Sets the id value for this FileSearch                  |

(continues)

**Table 7-43** actuate.reportexplorer.FileSearch functions (continued)

| Function                  | Condition                                              |
|---------------------------|--------------------------------------------------------|
| setDependentFileId( )     | Sets the id value for this FileSearch                  |
| setDependentFileName( )   | Sets the file name value for this FileSearch           |
| setFetchDirection( )      | Sets the owner value for this FileSearch               |
| setFetchHandle( )         | Sets the fetchHandle value for this FileSearch         |
| setFetchSize( )           | Sets the fetchSize value for this FileSearch           |
| setIncludeHiddenObject( ) | Sets the includeHiddenObject value for this FileSearch |
| setOwner( )               | Sets the Owner                                         |
| setPrivilegeFilter( )     | Sets the PrivilegeFilter value for this FileSearch     |
| setRequiredFileId( )      | Sets the requiredFileId for this FileSearch            |
| setRequiredFileName( )    | Sets the requiredFileName value for this FileSearch    |

## getAccessType

**Syntax** string FileSearch.getAccessType( )

Gets the access type.

**Returns** String. Either "private" or "shared" according to whether the FileSearch has been shared or not.

**Example** To halt a script if a FileSearch is private, use code similar to the following:

```
if(fsearch.getAccessType() == "private"){ return; }
```

## getCondition

**Syntax** actuate.reportexplorer.FileCondition FileSearch.getCondition( )

Gets the condition from the FileSearch.

**Returns** actuate.reportexplorer.FileCondition object. A condition to apply in a search.

**Example** To halt a script if a FileSearch does not have a condition, use code similar to the following:

```
if(fsearch.getCondition() == null){ return; }
```

## getConditionArray

**Syntax** actuate.reportexplorer.FileCondition[ ] FileSearch.getConditionArray( )

Gets the file condition array for this FileSearch.

**Returns** Array of actuate.reportexplorer.FileCondition objects. Multiple conditions to apply in a search.

**Example** To retrieve the array of file conditions from the FileSearch object fsearch, use code similar to the following:

```
var conditions = new Array();
conditions = fsearch.getConditionArray();
```

## getCountLimit

**Syntax** integer FileSearch.getCountLimit( )

Gets the maximum number of match results to display set for this file search.

**Returns** Integer. The maximum number of match results to display. 0 indicates unlimited.

**Example** To retrieve the count limit from the FileSearch object fsearch, use code similar to the following:

```
var limit = fsearch.getCountLimit();
```

## getDependentFileId

**Syntax** string FileSearch.getDependentFileId( )

Gets the file ID of the FileSearch, identifying the file it is set to search.

**Returns** String. The file ID.

**Example** To retrieve the file Id from the FileSearch object fsearch, use code similar to the following:

```
var id = fsearch.getDependantFileId();
```

## getDependentFileName

**Syntax** string FileSearch.getDependentFileName( )

Gets the file name of the FileSearch.

**Returns** String. The file name.

**Example** To retrieve the file name from the FileSearch object fsearch, use code similar to the following:

```
var name = fsearch.getDependantFileName();
```

## getFetchDirection

**Syntax** boolean FileSearch.getFetchDirection( )

Gets the fetch direction of the FileSearch.

## actuate.reportexplorer.FileSearch

**Returns** Boolean. True indicates ascending order.

**Example** To switch the fetch direction for the FileSearch object fsearch, use code similar to the following:

```
fsearch.setFetchDirection(!fsearch.getFetchDirection());
```

## getFetchHandle

**Syntax** string FileSearch.getFetchHandle( )

Gets the fetch handle.

**Returns** String. The fetch handle.

**Example** To retrieve the fetch handle from the FileSearch object fsearch, use code similar to the following:

```
var handle = fsearch.getFetchHandle();
```

## getFetchSize

**Syntax** integer FileSearch.getFetchSize( )

Gets the fetch size.

**Returns** Integer. The fetch size.

**Example** To halt a script if a FileSearch has a fetch size of 0, use code similar to the following:

```
if(fsearch.getFetchSize() == 0){ return; }
```

## getIncludeHiddenObject

**Syntax** boolean FileSearch.getIncludeHiddenObject( )

Gets the includeHiddenObject value for this FileSearch.

**Returns** Boolean. True includes hidden object.

**Example** To alert the user that hidden objects are enabled for a FileSearch, use code similar to the following:

```
if(fsearch.getIncludeHiddenObject()){
 alert("Hidden objects are enabled.");
}
```

## getOwner

**Syntax** string FileSearch.getOwner( )

Gets the owner's name.

**Returns** String. The owner's user name.

**Example** To retrieve the owner of fsearch, use code similar to the following:

```
var owner = fsearch.getOwner();
```

## getPrivilegeFilter

**Syntax** actuate.reportexplorer.PrivilegeFilter FileSearch.getPrivilegeFilter( )

Gets the privilege filter.

**Returns** actuate.reportexplorer.PrivilegeFilter object. A privilege filter.

**Example** To retrieve the privilege filter for fsearch, use code similar to the following:

```
var privileges = fsearch.getPrivilegeFilter();
```

## getRequiredFileId

**Syntax** integer FileSearch.getRequiredFileId( )

Gets the requiredFileId of FileSearch.

**Returns** Integer. A field ID.

**Example** To retrieve the required field ID assigned to fsearch, use code similar to the following:

```
var id = fsearch.getRequiredFileId();
```

## getRequiredFileName

**Syntax** string FileSearch.getRequiredFileName( )

Gets the requiredFileName name.

**Returns** String. A file name.

**Example** To retrieve the file name assigned to fsearch, use code similar to the following:

```
var id = fsearch.getRequiredFileName();
```

## setAccessType

**Syntax** void FileSearch.setAccessType(string accessType)

Sets the access type.

**Parameter** **accessType**

String. Either "private" or "shared" according to whether FileSearch has been shared or not.

## actuate.reportexplorer.FileSearch

**Example** To make a FileSearch fsearch private, use code similar to the following:

```
fsearch.setAccessType("private");
```

### setCondition

**Syntax** void FileSearch.setCondition(actuate.reportExplorer.FileCondition condition)

Sets a search condition for this FileSearch.

**Parameter** **condition**

actuate.reportexplorer.FileCondition object. A condition to apply to this search.

**Example** To clear FileSearch fsearch's condition, use code similar to the following:

```
fsearch.setCondition(null);
```

### setConditionArray

**Syntax** void FileSearch.setConditionArray(actuate.reportExplorer.FileCondition[] ConditionArray)

Sets multiple search conditions for this FileSearch.

**Parameter** **ConditionArray**

Array of actuate.reportexplorer.FileCondition objects. Conditions to apply to this search.

**Example** To clear FileSearch fsearch's condition array, use code similar to the following:

```
fsearch.setConditionArray(null);
```

### setCountLimit

**Syntax** void FileSearch.setCountLimit(integer countlimit)

Sets the maximum number of match results to display.

**Parameter** **countlimit**

Integer. The maximum number of match results to display. 0 indicates unlimited.

**Example** To set FileSearch fsearch to stop searching after finding 100 matches, use code similar to the following:

```
fsearch.setCountLimit(100);
```

### setDependentFileId

**Syntax** void FileSearch.setDependentFileId(string dependentFileId)

Sets the file ID of the FileSearch.

**Parameter** **dependent fileId**  
String. A file ID.

**Example** To set FileSearch fsearch's File ID to current, use code similar to the following:  
`fsearch.setDependent fileId("current");`

## setDependentFileName

**Syntax** `void FileSearch.setDependentFileName(string dependentFileName)`  
Sets the file name of FileSearch.

**Parameter** **dependentFileName**  
String. A file name.

**Example** To set FileSearch fsearch's file name to current, use code similar to the following:  
`fsearch.setDependentFileName("current");`

## setFetchDirection

**Syntax** `void FileSearch.setFetchDirection(boolean fetchDirection)`  
Sets the fetch direction for this FileSearch.

**Parameter** **fetchDirection**  
Boolean. True indicates ascending order.

**Example** To switch the fetch direction for the FileSearch object fsearch, use code similar to the following:

```
fsearch.setFetchDirection(!fsearch.getFetchDirection());
```

## setFetchHandle

**Syntax** `void FileSearch.setFetchHandle(string fetchHandle)`  
Sets the fetch handle for FileSearch.

**Parameter** **fetchHandle**  
String. A fetch handle.

**Example** To set FileSearch fsearch's fetch handle to ezsearch, use code similar to the following:

```
fsearch.setFetchHandle("ezsearch");
```

## setFetchSize

**Syntax** `void FileSearch.setFetchSize(integer fetchSize)`  
Sets the fetch size.

**Parameter** **fetchSize**

Integer. The fetch size.

**Example** To set FileSearch fsearch's fetch size to 12, use code similar to the following:

```
fsearch.setFetchSize(12);
```

## setIncludeHiddenObject

**Syntax** void FileSearch.setIncludeHiddenObject(boolean includeHiddenObject)

Sets the includeHiddenObject value for this FileSearch.

**Parameter** **includeHiddenObject**

Boolean. True includes hidden object.

**Example** To prohibit FileSearch fsearch from including hidden objects, use code similar to the following:

```
fsearch.setIncludeHiddenObject(false);
```

## setOwner

**Syntax** void FileSearch.setOwner(string owner)

Sets the owner for this FileSearch.

**Parameter** **owner**

String. The owner's user name.

**Example** To set the FileSearch fsearch owner to administrator, use code similar to the following:

```
fsearch.setOwner("administrator");
```

## setPrivilegeFilter

**Syntax** void FileSearch.setPrivilegeFilter(actuate.reportexplorer.PrivilegeFilter privilegeFilter)

Sets the privilege filter.

**Parameter** **privilegeFilter**

actuate.reportexplorer.PrivilegeFilter object. The privilege filter.

**Example** To assign the privilege filter pfilter to the FileSearch fsearch, use code similar to the following:

```
fsearch.setPrivilegeFilter(pfilter);
```

## setRequiredFileId

**Syntax** void FileSearch.setRequiredFileId(string requiredFileId)

Sets the requiredFileId for this FileSearch.

**Parameter** **requiredFileId**

String. A file ID.

**Example** To set the FileSearch fsearch file ID to permanent, use code similar to the following:

```
fsearch.setRequiredFileId("permanent");
```

## setRequiredFileName

**Syntax** void FileSearch.setRequiredFileName(string requiredFileName)

Sets the required file name.

**Parameter** **requiredFileName**

String. A file name.

**Example** To set the FileSearch fsearch 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 7-44 lists actuate.reportexplorer.FolderItems functions.

**Table 7-44** actuate.reportexplorer.FolderItems functions

| Function                      | Description                                           |
|-------------------------------|-------------------------------------------------------|
| <code>getFetchHandle()</code> | Gets the fetchHandle value for GetFolderItemsResponse |
| <code>getItemList()</code>    | Gets the itemList value for GetFolderItemsResponse    |
| <code>getTotalCount()</code>  | Gets the totalCount value for GetFolderItemsResponse  |
| <code>setFetchHandle()</code> | Sets the fetchHandle value for GetFolderItemsResponse |
| <code>setItemList()</code>    | Sets the itemList value for GetFolderItemsResponse    |
| <code>setTotalCount()</code>  | Sets the totalCount value for GetFolderItemsResponse  |

### getFetchHandle

**Syntax** `string FolderItems.getFetchHandle( )`

Retrieves the fetch handle for this folder's contents.

**Returns** String. The fetch handle.

**Example** To retrieve the fetch handle from items, use code similar to the following:

```
var handle = fitems.getFetchHandle();
```

### getItemList

**Syntax** `actuate.reportexplorer.File[ ] FolderItems.getItemList( )`

Gets the list of file contents for the folder.

**Returns** Array of actuate.reportexplorer.File objects. A list of the folder contents.

**Example** To store the items item list in the files variable, use code similar to the following:

```
files = fitems.getItemList();
```

## getTotalCount

**Syntax** string FolderItems.getTotalCount( )

Returns the maximum number of list items to retrieve from this folder.

**Returns** String. The total count.

**Example** To retrieve the total count from items, use code similar to the following:

```
var count = fitems.getTotalCount();
```

## setFetchHandle

**Syntax** void FolderItems.setFetchHandle(string fetchHandle)

Sets the fetch handle value for this FolderItems object.

**Parameter** **fetchHandle**

String. The fetch handle.

**Example** To set the FolderItems items fetch handle to dir, use code similar to the following:

```
fitems.setFetchHandle("dir");
```

## setItemList

**Syntax** void FolderItems.setItemList(actuate.reportexplorer.File[ ] itemList)

Sets the list of contents for this folder.

**Parameter** **itemList**

Array of actuate.reportexplorer.File objects. A list of the folder contents.

**Example** To assign the item list from items1 to items2, use code similar to the following:

```
items2.setItemList(items1.getItemList());
```

## setTotalCount

**Syntax** void FolderItems.setTotalCount(string totalCount)

Sets the maximum number of list items to retrieve from this folder.

**Parameter** **totalCount**

String. The total count.

**Example** To reset the count total for items, 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 7-45 lists actuate.reportexplorer.PrivilegeFilter functions.

**Table 7-45** actuate.reportexplorer.PrivilegeFilter functions

| Function              | Description                                             |
|-----------------------|---------------------------------------------------------|
| getAccessRights( )    | Gets the accessRights value for this PrivilegeFilter    |
| getGrantedRoleId( )   | Gets the grantedRoleId value for this PrivilegeFilter   |
| getGrantedRoleName( ) | Gets the grantedRoleName value for this PrivilegeFilter |
| getGrantedUserId( )   | Gets the grantedUserId value for this PrivilegeFilter   |
| getGrantedUserName( ) | Gets the grantedUserName value for this PrivilegeFilter |
| setAccessRights( )    | Sets the accessRights value for this PrivilegeFilter    |
| setGrantedRoleId( )   | Sets the grantedRoleId value for this PrivilegeFilter   |
| setGrantedRoleName( ) | Sets the grantedRoleName value for this PrivilegeFilter |
| setGrantedUserId( )   | Sets the grantedUserId value for this PrivilegeFilter   |
| setGrantedUserName( ) | Sets the grantedUserName value for this PrivilegeFilter |

## getAccessRights

**Syntax** string privilegeFilter.getAccessRights( )

Gets the repository access rights value for this PrivilegeFilter.

**Returns** String. Repository access rights.

**Example** To halt a script if the access rights of a PrivilegeFilter pfilter are null, use code similar to the following:

```
if (pfilter.getAccessRights() == null) { return; }
```

## getGrantedRoleId

**Syntax** string PrivilegeFilter.getGrantedRoleId( )

Gets the grantedRoleId value for this PrivilegeFilter.

**Returns** String. A role ID.

**Example** To retrieve the granted role ID for a PrivilegeFilter pfilter, use code similar to the following:

```
var roleid = pfilter.getGrantedRoleId();
```

## getGrantedRoleName

**Syntax** string PrivilegeFilter.getGrantedRoleName( )

Gets the grantedRoleName value for this PrivilegeFilter.

**Returns** String. A role name.

**Example** To retrieve the granted role name for a PrivilegeFilter pfilter, use code similar to the following:

```
var rolename = pfilter.getGrantedRoleName();
```

## getGrantedUserId

**Syntax** string PrivilegeFilter.getGrantedUserId( )

Gets the grantedUserId value for this PrivilegeFilter.

**Returns** String. A user ID.

**Example** To retrieve the granted user ID for a PrivilegeFilter pfilter, use code similar to the following:

```
var userid = pfilter.getGrantedUserId();
```

## getGrantedUserName

**Syntax** string PrivilegeFilter.getGrantedUserName( )

Gets the grantedUserName value for this PrivilegeFilter.

**Returns** String. A user name.

**Example** To retrieve the granted user name for a PrivilegeFilter pfilter, use code similar to the following:

```
var username = pfilter.getGrantedUserName();
```

## setAccessRights

**Syntax** void PrivilegeFilter.setAccessRights(string accessRights)

Sets the repository access rights value for this PrivilegeFilter.

**Parameter** **accessRights**

String. The access rights.

**Example** To copy the set of access rights from PrivilegeFilter pfilter1 to PrivilegeFilter pfilter2, use code similar to the following:

```
pfilter2.setAccessRights(pfilter1.getAccessRights());
```

## setGrantedRoleId

**Syntax** void PrivilegeFilter.setGrantedRoleId(string grantedRoleId)

Sets the grantedRoleId of the column for this PrivilegeFilter.

**Parameter** **grantedRoleId**

String. A role ID.

**Example** To set the granted role ID of the PrivilegeFilter pfilter to All, use code similar to the following:

```
pfilter.setGrantedRoleId("All");
```

## setGrantedRoleName

**Syntax** void PrivilegeFilter.setGrantedRoleName(string grantedRoleName)

Sets the grantedRoleName value for this PrivilegeFilter.

**Parameter** **grantedRoleName**

String. A role name.

**Example** To set the granted role name of the PrivilegeFilter pfilter to Everyone, use code similar to the following:

```
pfilter.setGrantedRoleName("Everyone");
```

## setGrantedUserId

**Syntax** void PrivilegeFilter.setGrantedUserId(string grantedUserId)

Sets the grantedUserId value for this PrivilegeFilter.

**Parameter** **grantedUserId**

String. A user ID.

**Example** To set the granted user ID of the PrivilegeFilter pfilter to administrator, use code similar to the following:

```
pfilter.setGrantedRoleId("Administrator");
```

## setGrantedUserName

**Syntax** void PrivilegeFilter.setGrantedUserName(string grantedUserName)

Sets the grantedUserName value for this PrivilegeFilter.

**Parameter** **grantedUserName**

String. A user name.

**Example** To set the granted user name of the PrivilegeFilter pfilter to administrator, use code similar to the following:

```
pfilter.setGrantedRoleId("Administrator");
```

---

## Class actuate.RequestOptions

**Description** The request options that loginServlet requires to authenticate requests. RequestOptions is used by other classes to provide authentication information. It also adds any customized options to the request URL.

### Constructor

**Syntax** actuate.RequestOptions( actuate.RequestOptions requestOptions)  
Constructs a new RequestOptions object.

**Parameter** **requestOptions** actuate.RequestOptions object. Optional. Provides request option settings to copy into this RequestOptions object.

### Function summary

Table 7-46 lists actuate.RequestOptions functions.

**Table 7-46** actuate.RequestOptions functions

| Function               | Description                                       |
|------------------------|---------------------------------------------------|
| getIserverUrl( )       | Returns the BIRT iHub URL value                   |
| getLocale( )           | Returns the current locale                        |
| getRepositoryType( )   | Returns the repository type                       |
| getVolume( )           | Returns the volume                                |
| getVolumeProfile( )    | Returns the volume profile                        |
| setCustomParameters( ) | Appends custom parameters to the request URL      |
| setIserverUrl( )       | Sets the BIRT iHub URL value                      |
| setLocale( )           | Sets the locale                                   |
| setRepositoryType( )   | Sets the repository type: enterprise or workgroup |
| setVolume( )           | Sets the volume                                   |
| setVolumeProfile( )    | Sets the volume profile                           |

### getIserverUrl

**Syntax** string RequestOptions.getIserverurl( )

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();
```

## 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");
}
```

## 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();
```

## getVolume

**Syntax** string RequestOptions.getVolume( )

Returns the volume.

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

- Example** To retrieve the volume for the RequestOptions object reqOpts, use code similar to the following:

```
var encyVol = reqOpts.getVolume();
```

## 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.

## actuate.RequestOptions

**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();
```

## setCustomParameters

**Syntax** void RequestOptions.setCustomParameters(object parameters)

Returns a custom parameter in the request URL.

**Parameter** **parameters**

Object. An associative array of name:value pairs for URL parameters.

**Example** To add "&myParam=myValue" in a request URL derived from RequestOptions object, use code similar to the following:

```
MyRequestOptions.setCustomParameters ({myParam: "myValue" });
```

## setServerUrl

**Syntax** void RequestOptions.setServerUrl(string iServerUrl)

Sets the BIRT iHub URL.

**Parameter** **iServerUrl**

String. The BIRT iHub URL value.

**Example** This example sets the BIRT iHub URL for the reqOpts RequestOptions object:

```
reqOpts.setServerUrl("http://127.0.0.1:8700");
```

## setLocale

**Syntax** void RequestOptions.setLocale(string Locale)

Sets the locale.

**Parameter** **Locale**

String. Optional. The locale value. Null indicates the default locale.

**Example** This example resets the locale for the reqOpts RequestOptions object to the default value provided by the actuate web service to which the JSAPI connects:

```
reqOpts.setLocale();
```

This example resets the locale for the reqOpts RequestOptions object to Spain using the Spanish locale code listed in <context root>\WEB-INF\localemap.xml:

```
reqOpts.setLocale("es_ES");
```

## setRepositoryType

**Syntax** void RequestOptions.setRepositoryType(string repositoryType)

Sets the repository type, either enterprise or workgroup.

**Parameter** **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);
```

## setVolume

**Syntax** void RequestOptions.setVolume(string volume)

Sets the volume.

**Parameter** **volume**

String. The volume.

**Example** To set the volume to marcom if the RequestOptions object reqOpts volume is null, use code similar to the following:

```
if(reqOpts.getVolume() == null){
 reqOpts.setVolume("marcom");
}
```

## 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.

**Parameter** **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.setVolumeProfile("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 7-47 lists actuate.Viewer functions.

**Table 7-47** 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 7-47** actuate.Viewer functions (continued)

| Function                | Description                                                     |
|-------------------------|-----------------------------------------------------------------|
| getContentMargin()      | Gets the margin dimensions of the content in pixels             |
| getCurrentPageContent() | Returns the report content displayed in the viewer              |
| getCurrentPageNum()     | Returns the current page number                                 |
| getDataItem()           | Retrieves a data item by bookmark                               |
| getFlashObject()        | Retrieves a Flash object by bookmark                            |
| getGadget()             | Retrieves a gadget by bookmark                                  |
| getHeight()             | Returns the viewer height setting                               |
| getLabel()              | Retrieves a label by bookmark                                   |
| getReportletBookmark()  | Returns the bookmark of a Reportlet displayed in the viewer     |
| getReportName()         | Returns the report file displayed in the viewer                 |
| getTable()              | Retrieves a table by bookmark                                   |
| getText()               | Retrieves a text element by bookmark                            |
| getTotalPageCount()     | Returns the total number of pages                               |
| getUIConfig()           | Gets the UIConfig object assigned to the viewer                 |
| getUIOptions()          | Returns the UIOptions object                                    |
| getViewer()             | Returns a viewer object containing the given bookmarked element |
| getWidth()              | Returns the viewer width setting                                |
| gotoBookmark()          | Goes to the position in the report specified by the bookmark    |
| gotoPage()              | Goes to the specified page                                      |
| isInteractive()         | Returns whether interactive viewing features are enabled        |
| saveReportDesign()      | Saves a report design to the repository                         |
| saveReportDocument()    | Saves a report document to the repository                       |
| setContentMarg()        | Sets the viewer content margin                                  |
| setFocus()              | Sets the focus element of the viewer                            |
| setHeight()             | Sets the viewer height                                          |

*(continues)*

**Table 7-47** actuate.Viewer functions (continued)

| Function                       | Description                                                                                                                     |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| setParameters( )               | Sets the parameters to run a report using a list of literal string pairs                                                        |
| setParameterValues( )          | Sets the parameters to run a report using a generated object                                                                    |
| setReportDocument( )           | Sets the report document to render within this Viewer                                                                           |
| setReportletBookmark( )        | Sets bookmark name for a Reportlet                                                                                              |
| setReportName( )               | Sets the report file to render within this Viewer                                                                               |
| setService( )                  | Sets the target service URL                                                                                                     |
| setSize( )                     | Sets the size of the viewer                                                                                                     |
| setSupportSVG( )               | Sets the Scalable Vector Graphic support flag to enable Scalable Vector Graphics content                                        |
| setUIOptions( )                | Sets UIOptions using a UIOptions object                                                                                         |
| setViewingMode( )              | Sets the dashboard viewing mode                                                                                                 |
| setWidth( )                    | Sets the width of the viewer                                                                                                    |
| showDownloadReportDialog( )    | Enables the export report dialog window                                                                                         |
| showDownloadResultSetDialog( ) | Enables the download data dialog window                                                                                         |
| showFacebookCommentPanel( )    | Shows the Facebook comments panel                                                                                               |
| showParameterPanel( )          | Shows the parameter panel                                                                                                       |
| showPrintDialog( )             | Enables the print dialog window                                                                                                 |
| showTocPanel( )                | Shows the table of contents panel                                                                                               |
| submit( )                      | Submits all the asynchronous operations that the user has requested on this Viewer and renders the viewer component on the page |

## disableIV

**Syntax** void Viewer.disableIV(function callback)

Disables the Interactive Viewer features of this viewer object. This is an asynchronous setting committed by submit( ).

**Parameter** **callback**

Function. The callback function to call after the Interactive Viewer is disabled.

**Example** To disable the Interactive Viewer option for myViewer, use code similar to the following:

```
myViewer.disableIV(function alertUser(){alert("IV disabled");});
```

## downloadReport

**Syntax** void Viewer.downloadReport(string format, string pages, actuate.viewer.RenderOptions renderoption)

Exports the report with a specified format. The downloadReport function does not return any object. The report is exported to the client side. Then the browser opens a download window for the user to specify a location for the report.

**Parameters**

**format**

String. The format in which to export the report. Valid values and their corresponding formats are:

- doc: Word
- docx: Word 2007
- html: HTML-encoded web page
- ppt: PowerPoint
- ptx: PowerPoint 2007
- pdf: Adobe PDF
- ps: PostScript
- xls: Excel
- xlsx: Excel 2007

**pages**

String. The pages to retrieve. Indicate page ranges by using the first page number of the range and the last page number separated by a dash. To use more than one value, separate individual page numbers or page ranges with commas.

**renderoption**

actuate.viewer.RenderOptions object. Optional. Sets the rendering options for the download, which currently only applies to multisheet xls format reports.

**Example** To download the first five pages of the report in the viewer, use the following code:

```
viewer.downloadReport("pdf", "1-5", null);
```

## downloadResultSet

**Syntax** void Viewer.downloadResultSet(actuate.data.Request request, function callback)

## actuate.Viewer

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.

|                   |                                                                                                                                                                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <b>request</b><br>actuate.data.Request object. The request to generate the result set.<br><br><b>callback</b><br>Function. The callback function to call after retrieving the results. The callback function must take an actuate.data.ResultSet object as an argument. |
| <b>Example</b>    | This example creates an actuate.data.ResultSet object from the report in myViewer as specified by myRequest and passes it to a callback function:<br><br><code>myViewer.downloadResultSet(myRequest, callback);</code>                                                  |

## 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);});
}
```

|                  |                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------|
| <b>Parameter</b> | <b>callback</b><br>Function. The callback function to call after enabling the Interactive Viewer features. |
|------------------|------------------------------------------------------------------------------------------------------------|

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <b>Example</b> | This function must be used in the callback of viewer.submit( ) as shown in the following example: |
|----------------|---------------------------------------------------------------------------------------------------|

```
function runInteractive(){
myviewer.setReportName("/Public/BIRT and BIRT Studio Examples
/Sales by Customer.rptdesign");
myviewer.submit(function() {myviewer.enableIV(callback);});
}
```

## getChart

### Syntax

```
actuate.report.Chart Viewer.getChart(string bookmark)
```

Returns an instance of the chart referenced by a bookmark.

|                  |                                               |
|------------------|-----------------------------------------------|
| <b>Parameter</b> | <b>bookmark</b><br>String. The bookmark name. |
|------------------|-----------------------------------------------|

**Returns** actuate.report.Chart object.

**Example** This example returns the chart with the bookmark ChartBookmark:

```
function getMyChartByBookmark (myReport) {
 var bviewer = myReport.getViewer("Chart");
 var bpagecontents = bviewer.getCurrentPageContent();
 return bpagecontents.getChart("ChartBookmark");
}
```

## getClientHeight

**Syntax** integer Viewer.getClientHeight( )

Gets the browser window's height.

**Returns** Integer. Height in pixels.

**Example** To reset the viewer height to 20 pixels less than the browser window if it is larger than the browser window, use code similar to the following:

```
if (myViewer.getClientHeight() < myViewer.getHeight()) {
 myViewer.setHeight(myViewer.getClientHeight() - 20);
}
```

## getClientWidth

**Syntax** integer Viewer.getClientWidth( )

Gets the browser window's width.

**Returns** Integer. Width in pixels.

**Example** To reset the viewer width to 20 pixels less than the browser window if it is larger than the browser window, use code similar to the following:

```
if (myViewer.getClientWidth() < myViewer.getWidth()) {
 myViewer.setWidth(myViewer.getClientWidth() - 20);
}
```

## getContentByBookmark

**Syntax** void Viewer.getContentByBookmark(string bookmark, string format, function callback)

Gets the report content by bookmark and passes the content as data to a callback.

**Parameters** **bookmark**

String. The bookmark of a report element to retrieve.

**format**

String. The output format, which is either html or xhtml.

actuate.Viewer

#### **callback**

Function. Callback to be called once the operation is finished. The callback must take actuate.data.ReportContent object as an argument.

- Example** To retrieve the content with the bookmark FirstChart as html, use code similar to the following:

```
myViewer.getContentByBookmark("FirstChart", "html", processChart);
```

## **getContentByPageRange**

**Syntax** void Viewer.getContentByPageRange(string PageRange, string format, function callback)

Gets the report content by page range and passes the content as data to a callback.

**Parameters** **PageRange**

String. Page range to retrieve the report content, separated by a dash.

**format**

String. The output format, which is either html or xhtml.

**callback**

Function. Callback to be called once the operation is finished. The callback must take actuate.data.ReportContent object as an argument.

- Example** To retrieve the content from pages 3 through 5 as html, use code similar to the following:

```
myViewer.getContentByPageRange("3-5", "html", processPages);
```

## **getContentMargin**

**Syntax** integer | object Viewer.getContentMargin( )

Gets the viewer content margin.

**Returns** Integer or Object. An integer indicates the same margin on all sides, in pixels. The object contains the pixel values for the top, bottom, left, and right margins of the viewer in an array. For example, a 25-pixel top content margin and no margin in the other directions would be the object array {top:25, left:0, right:0, bottom:0}.

- Example** To set the margin of the viewer newViewer to match the margin of myViewer, use code similar to the following:

```
newViewer.setContentMargin(myViewer.getContentMargin());
```

## **getCurrentPageContent**

**Syntax** actuate.viewer.Content Viewer.getCurrentPageContent( )

Returns the report content displayed in the viewer. This function is the entry point for retrieving the report elements from this viewer object.

**Returns** actuate.viewer.PageContent object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the table "mytable" on the page loaded in the myViewer viewer object, use the following code:

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

## getCurrentPageNum

**Syntax** integer Viewer.getCurrentPageNum()

Returns the page number for the page currently being displayed.

**Returns** Integer. The current page number.

**Example** This function is useful to move to another page relative to the current page. To go to the next page in a document, use the following code:

```
viewer.gotoPage(viewer.getCurrentPageNum() + 1);
```

## getDataItem

**Syntax** actuate.report.DataItem Viewer.getDataItem(string bookmark)

Returns an instance of report data referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

**Returns** actuate.report.DataItem object.

**Example** To get the report data with the bookmark FirstDataItem and store it in the variable myDataItem, use code similar to the following:

```
var myDataItem = myViewer.getDataItem("FirstDataItem");
```

## getFlashObject

**Syntax** actuate.report.FlashObject Viewer.getFlashObject(string bookmark)

Returns an instance of the Flash object referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

actuate.Viewer

**Returns** actuate.report.FlashObject object.

**Example** To get the Flash object with the bookmark FirstFlashObject and store it in the variable myFlashObject, use code similar to the following:

```
var myFlashObject = myViewer.getFlashObject("FirstFlashObject");
```

## getGadget

**Syntax** actuate.report.Gadget Viewer.getGadget(string bookmark)

Returns an instance of the gadget referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

**Returns** actuate.report.Gadget object.

**Example** To get the gadget with the bookmark FirstGadget and store it in the variable myGadget, use code similar to the following:

```
var myGadget = myViewer.getGadget("FirstGadget");
```

## getHeight

**Syntax** string Viewer.getHeight( )

Returns the height value of the viewer.

**Returns** String.

**Example** This example decreases the viewer's height by 10:

```
var height = myViewer.getHeight();
myViewer.setHeight(height - 10);
```

## getLabel

**Syntax** actuate.report.Label Viewer.getLabel(string bookmark)

Returns an instance of the label referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

**Returns** actuate.report.Label object.

**Example** To get the label with the bookmark FirstLabel and store it in the variable myLabel, use code similar to the following:

```
var myLabel = myViewer.getLabel("FirstLabel");
```

## getReportletBookmark

**Syntax** `string Viewer.getReportletBookmark( )`

Returns the bookmark of the current report page or element.

**Returns** String. Bookmark.

**Example** This example displays the bookmark of the current report page in an alert box:

```
alert ("Report bookmark is " + myViewer.getReportletBookmark());
```

## getReportName

**Syntax** `string Viewer.getReportName( )`

Returns the name of the report file, either a report design file or report document file, that is currently displayed in this Viewer.

**Returns** String.

**Example** This example displays the currently displayed report file name in an alert box:

```
alert ("Currently displaying " + myViewer.getReportName());
```

## getTable

**Syntax** `actuate.report.Table Viewer.getTable(string bookmark)`

Returns an instance of the table referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

**Returns** `actuate.report.Table` object.

**Example** To get the table with the bookmark FirstTable and store it in the variable myTable, use code similar to the following:

```
var myTable = myViewer.getTable("FirstTable");
```

## getText

**Syntax** `actuate.report.Text Viewer.getText(string bookmark)`

Returns an instance of the Text object referenced by a bookmark.

**Parameter** **bookmark**

String. The bookmark name.

**Returns** `actuate.report.Text` object.

actuate.Viewer

**Example** To get the Text object with the bookmark Title and store it in the variable myText, use code similar to the following:

```
var myText = myViewer.getText("Title");
```

## getTotalPageCount

**Syntax** integer Viewer.getTotalPageCount( )

Returns the total number of pages in the report being viewed.

**Returns** Integer.

**Example** This function is useful to move to the last page of a document. To go to the last page in a document, use the following code:

```
viewer.gotoPage(viewer.getTotalPageCount());
```

## getUIConfig

**Syntax** actuate.viewer.UIConfig Viewer.getUIConfig( )

Returns the current UI configuration.

**Returns** actuate.viewer.UIConfig object. This function returns null when no UIConfig object is set.

**Example** To retrieve and store the content pane from the viewer, use the following code:

```
var contentpane = viewer.getUIConfig().getContentPane();
```

## getUIOptions

**Syntax** actuate.viewer.UIOptions Viewer.getUIOptions( )

Returns the UIOptions set in this viewer object.

**Returns** actuate.viewer.UIOptions object. This function returns null when no UIOptions object is set.

**Example** To retrieve and store the uiOptions for the viewer, use the following code:

```
var options = myViewer.getUIOptions();
```

## getViewer

**Syntax** actuate.Viewer Viewer.getViewer(string bookmark)

actuate.Viewer Viewer.getViewer(object elementID)

Returns a viewer object containing the report element that is associated with a bookmark or contained in an HTML element.

- Parameters**
- bookmark**  
String. The bookmark of the report element to view.
  - elementID**  
Object. An HTML element that contains the viewer.
- Returns** actuate.Viewer object or null if the viewer is not found.
- Example** This example uses getViewer( ) to retrieve a report element and return the bookmark of the chart in that report:

```
function chartBookmark (myReport) {
 var bviewer = myReport.getViewer("Chart");
 var bpagecontents = bviewer.getCurrentPageContent();
 return bpagecontents.getChartByBookmark("ChartBookmark");
}
```

## getWidth

- Syntax** string Viewer.getWidth( )
- Returns the width value of the viewer.
- Returns** String.
- Example** This example decreases the viewer's width by 10:

```
var width = myViewer.getWidth();
myViewer.setWidth(width - 10);
```

## gotoBookmark

- Syntax** void Viewer.gotoBookmark(string bookmark)
- Goes to the page position by the specified bookmark. The viewer displays to the first page when the bookmark is not found.
- Parameter**
- bookmark**  
String. The bookmark of a report element.
- Example** To move the viewer to the page position specified by the value of the 'bookmark' parameter, use this code:

```
viewer.gotoBookmark(document.getElementById('bookmark').value);
```

## gotoPage

- Syntax** void Viewer.gotoPage(integer pageNumber)
- Goes to the specified page. The viewer throws an exception when the page is not found.

actuate.Viewer

**Parameter** **pageNumber**

Integer. A page number in the report.

**Example** To go to the first page of a report, use the following code:

```
viewer.gotoPage(1);
```

## isInteractive

**Syntax** boolean Viewer.isInteractive()

Returns the interactive viewing status of the viewer. Enables or disables the interactive viewing features with actuate.Viewer.enableIV().

**Returns** Boolean. True when interactive viewing features are enabled.

**Example** This example displays an alert box with the interactive status of the viewer:

```
alert("Interactivity of this viewer is set to " +
 myViewer.isInteractive());
```

## saveReportDesign

**Syntax** void Viewer.saveReportDesign(string filename, function callback)

Saves the current viewer content as a report design. The viewer must enable interactive viewing with enableIV() prior to saving a report design.

**Parameters** **filename**

String. Sets the name of the saved file. The current file name is used if null. The file name must be a path relative to the viewer's repository.

**callback**

Function. Optional. The function to execute after the asynchronous call processing is done. The callback takes the current actuate.Viewer object as an input parameter.

**Example** To save the content of the viewer as the report design called NewDesign, use the following code:

```
myViewer.saveReportDesign("NewDesign");
```

## saveReportDocument

**Syntax** void Viewer.saveReportDocument(string filename, function callback)

Saves the current viewer content as a report document. The viewer must enable interactive viewing with enableIV() prior to saving a report design.

**Parameters** **filename**

String. Sets the name of the saved file. The current file name is used if null. The file name must be a path relative to the viewer's repository.

**callback**

Function. Optional. The function to execute after the asynchronous call processing is done. The callback takes the current `actuate.Viewer` object as an input parameter.

- Example** To save the content of the viewer as the report document called `NewDocument`, use the following code:

```
myViewer.saveReportDocument ("NewDocument") ;
```

**setContentMarg**

**Syntax** `void Viewer.setContentMargin(string[ ] margin)`

`void Viewer.setContentMargin(int margin)`

Sets the viewer content margin.

**Parameter** **margin**

Array of strings or integer. Each member of the array is the margin for the top, left, right, and bottom internal margins for the viewer. An integer sets all margins to that value.

- Example** To set the internal margin of the viewer to a 10-pixel buffer, use the following code:

```
myViewer.setContentMargin(10) ;
```

**setFocus**

**Syntax** `void setFocus(boolean focus)`

Sets the focus for the viewer.

**Parameter** **focus**

Boolean. The viewer's context menu is in focus when this parameter is set to true.

- Example** This example blurs the context menu for the viewer:

```
viewer.setFocus (false) ;
```

**setHeight**

**Syntax** `void Viewer.setHeight(integer height)`

Sets the viewer height.

**Parameter** **height**

Integer. The height in pixels.

- Example** To set the height of the viewer to 600 pixels, use the following code:

```
viewer.setHeight (600) ;
```

## setParameters

**Syntax** void Viewer.setParameters(string[ ] params)

Sets parameters for executing report using literal string pairs.

**Parameter** **params**

Array of strings. Each string in the array is constructed of name:"value" pairs. Use a literal list, such as {param1:"value1", param2:"value2", ... }.

**Example** To set the value of a parameter, city, to the value, New York, use the following object literal:

```
viewer.setParameters({ city:"New York"});
```

## setParameterValues

**Syntax** void Viewer.setParameterValues(actuate.parameter.ParameterValue[ ] parameters)

Sets parameter values for executing a report using ParameterValue objects.

**Parameter** **parameters**

Array of actuate.parameter.ParameterValue objects. An array of this kind is returned by actuate.Parameter.downloadParameterValues( ) and is the recommended function for creating the parameters input.

**Example** To set the parameter values for a report to the parameters in the pvs array, use this code:

```
viewer.setParameterValues(pvs);
```

## setReportDocument

**Syntax** void Viewer.setReportName(string reportFile, string connectionHandle)

Sets the report document to render in this Viewer.

**Parameters** **reportFile**

String. The report file path for a report document file. To set the version for the report, add a semicolon and the version number. For example, "/Public/BIRT and BIRT Studio Examples/Customer Dashboard.rptdocument;1" retrieves version 1 of Customer Dashboard.rptdocument.

**connectionHandle**

String. Optional. The unique identifier generated by iHub for a temporary report.

**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.setReportDocument("/Public/BIRT and BIRT Studio Examples
/Top 5 Sales Performers.rptdocument");
viewer.submit();
```

## setReportletBookmark

**Syntax** void Viewer.setReportletBookmark(string bookmark)

Sets the bookmark for the Reportlet rendered.

**Parameter** **bookmark**

String. The bookmark ID used to render the Reportlet. Viewer requires a bookmark to render a Reportlet. Viewer does not support automatically generated generic bookmarks from a BIRT report.

**Example** To open the Top 5 Customers Reportlet of the Customer Dashboard, set the Reportlet bookmark by name and then call viewer.submit, as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples
/Customer Dashboard.rptdocument");
viewer.setReportletBookmark("Top 5 Customers");
viewer.submit();
```

## setReportName

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

Sets the report file, either a report design or report document, to render in this Viewer.

**Parameter** **reportFile**

String. The report file path for a report design file or report document file. To set the version for the report, add a semicolon and the version number. For example, "/Public/BIRT and BIRT Studio Examples/Customer Dashboard.rptdesign;1" retrieves version 1 of Customer Dashboard.rptdesign.

**Example** To open the Top 5 Sales Performers report, set the report by name and then call submit( ), as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples
/Top 5 Sales Performers.rptdesign");
viewer.submit();
```

## setService

**Syntax** void Viewer.setService(string iPortalURL, actuate.RequestOptions  
requestOptions)

Sets the target service URL to which this Viewer links. When the service URL is not set, this Viewer links to the default service URL, which is set on the actuate object.

`actuate.Viewer`

**Parameters** **iPortalURL**

String. The target Actuate web application URL, either a Java Component or iHub Visualization Platform client.

**requestOptions**

`actuate.RequestOptions` object. Optional. `requestOptions` defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

**Example** This example sets the URL for the Actuate iPortal web application service:

```
myViewer.setService("http://127.0.0.1:8700/iportal",
myRequestOptions);
```

## **setSize**

**Syntax** `void Viewer.setSize(integer width, integer height)`

Resizes the viewer's width and height.

**Parameters** **width**

Integer. The new width is specified in pixels.

**height**

Integer. The new height is specified in pixels.

**Example** To set the viewer's size to 300 pixels by 300 pixels, use code similar to the following:

```
myViewer.setSize(300, 300);
```

## **setSupportSVG**

**Syntax** `void Viewer.setSupportSVG(boolean usvgFlag)`

Controls Scalable Vector Graphics support for the viewer.

**Parameter** **svgFlag**

Boolean. True enables Scalable Vector Graphic support.

**Example** To disable Scalable Vector Graphic support for the `myViewer` viewer, use code similar to the following:

```
myViewer.setSupportSVG(false);
```

## **setUIOptions**

**Syntax** `void Viewer.setUIOptions(actuate.viewer.UIOptions options)`

Sets the UI options for the viewer using an `actuate.viewer.UIOptions` object.

**Parameter** **options**  
actuate.viewer.UIOptions object. Enables or disables various controls and features.

**Example** To hide the toolbar for the viewer, use the following code:

```
uioptions.enableToolBar(false);
viewer.setUIOptions(uioptions);
viewer.submit();
```

## setViewingMode

**Syntax** void Viewer.setViewingMode(string viewer)  
Sets the dashboard viewing mode.

**Parameter** **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);
```

## setWidth

**Syntax** void Viewer.setWidth(string width)  
Sets the viewer width.

**Parameter** **width**  
String.

**Example** To set the width of the viewer to 800 pixels, use the following code:

```
viewer.setWidth(800);
```

## showDownloadReportDialog

**Syntax** void Viewer.showDownloadReportDialog( )  
Displays the export report dialog window.

**Example** Use this code to display the report dialog window:

```
myViewer.showDownloadReportDialog();
```

## showDownloadResultSetDialog

**Syntax** void Viewer.showDownloadResultSetDialog( )  
Displays the export data dialog window.

```
actuate.Viewer
```

**Example** Use this code to display the result set download dialog window:

```
viewer.showDownloadResultSetDialog();
```

## showFacebookCommentPanel

**Syntax** void Viewer.showFacebookCommentPanel( )

Displays the Facebook comments panel.

**Example** Use this code to display the Facebook comments panel:

```
viewer.showFacebookCommentPanel();
```

## showParameterPanel

**Syntax** void Viewer.showParameterPanel( )

Displays the parameter panel.

**Example** Use this code to display the parameter panel:

```
viewer.showParameterPanel();
```

## showPrintDialog

**Syntax** void Viewer.showPrintDialog( )

Displays the print dialog window.

**Example** Use this code to display the print dialog window:

```
viewer.showPrintDialog();
```

## showTocPanel

**Syntax** void Viewer.showTocPanel( )

Displays the table of contents panel.

**Example** Use this code to display the table of contents panel:

```
viewer.showTocPanel();
```

## submit

**Syntax** void Viewer.submit(function callback, boolean rerun)

Updates and reloads the viewer after submitting requests for the viewer. The submit() function triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are

rendered on the page in the viewer container. Calling submit( ) when a previous submit( ) is pending throws an exception.

**Parameters****callback**

Function. Optional. The function to execute after the asynchronous call processing is done.

**rerun**

Boolean. Optional. Indicates whether to re-run the report design when refreshing. Default to true.

**Example**

To open the Top 5 Sales Performers report, set the report by name and then call submit( ), as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples
/Top 5 Sales Performers.rptdesign");
viewer.submit();
```

## Class actuate.viewer.BrowserPanel

**Description** A container for a browser content panel in a viewer. This class defines the default scroll bars for a content panel.

### Constructor

**Syntax** `actuate.Viewer.BrowserPanel( )`

Constructs a new BrowserPanel object for the parent viewer. The browser panel has vertical and horizontal scroll bars for navigation.

---

## Class actuate.viewer.EventConstants

**Description** Defines the event constants supported by this API. Table 7-48 lists the viewer event constants.

**Table 7-48** 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 7-49 lists `actuate.viewer.PageContent` functions.

**Table 7-49** `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                                   |

### getChartByBookmark

**Syntax** `actuate.report.Chart PageContent.getChartByBookmark(string bookmark)`

Returns the chart element specified by the given bookmark.

**Parameter** `bookmark`

String. A bookmark to identify a chart element. When the bookmark value is not given, this function returns the first chart element found in the report content.

**Returns** actuate.report.Chart object.

**Example** This example retrieves the Chart object and changes the chart title:

```
this.onclick = function(event) {
 var bviewer = this.getViewer();
 var bpagecontents = bviewer.getCurrentPageContent();
 var bchart = bpagecontents.getChartByBookmark("ChartBookmark");
 bchart.setChartTitle("Orders By Country (Classic Cars)");
 bchart.submit();
}
```

## getDataItemByBookmark

**Syntax** actuate.report.DataItem PageContent.getDataItemByBookmark(string bookmark)

Returns the data element specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a data element. When the bookmark value is not given, the first data element found in the report content is returned.

**Returns** actuate.report.DataItem object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the data element "myDataItem" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent()
.getDataItemByBookmark("myDataItem");
```

## getFlashObjectByBookmark

**Syntax** actuate.report.FlashObject PageContent.getFlashObjectByBookmark(string bookmark)

Returns the Flash object specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a Flash object. When the bookmark value is not given, the first data element found in the report content is returned.

**Returns** actuate.report.FlashObject object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the Flash object "myFlashObj" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent()
.getFlashObjectByBookmark("myFlashObj");
```

## getGadgetByBookmark

**Syntax** actuate.report.Gadget PageContent.getGadgetByBookmark(string bookmark)

Returns the gadget element specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a gadget element. When the bookmark value is not given, the first data element found in the report content is returned.

**Returns** actuate.report.Gadget object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the gadget "myGadget" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent()
 .getGadgetByBookmark("myGadget");
```

## getLabelByBookmark

**Syntax** actuate.report.Label PageContent.getLabelByBookmark(string bookmark)

Returns the label element specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a label element. When the bookmark value is not given, the first label element found in the report content is returned.

**Returns** actuate.report.Label object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the label "LabelOne" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent()
 .getLabelByBookmark("LabelOne");
```

## getTableByBookmark

**Syntax** actuate.report.Table PageContent.getTableByBookmark(string bookmark)

Returns the table element specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a table element. When the bookmark value is not given, the first table element found in the report content is returned.

**Returns** actuate.report.Table object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the table mytable on the page loaded in the myViewer viewer object, use the following code:

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

## getTextByBookmark

**Syntax** actuate.report.TextItem PageContent.getTextByBookmark(string bookmark)

Returns the text element specified by the given bookmark.

**Parameter** **bookmark**

String. A bookmark to identify a text element. If the bookmark value is not given, the first text element found in the report content is returned.

**Returns** actuate.report.TextItem object.

**Example** Use this function to access the bookmarks for specific elements in the page content. For example, to access the text item "myTextItem" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent()
 .getTextByBookmark("myTextItem");
```

## getViewerId

**Syntax** string PageContent.getViewerId()

Returns the viewer ID.

**Returns** String. The viewer ID.

**Example** This example displays the viewer ID in an alert box:

```
alert("The Viewer ID is " + myViewer.getViewerId());
```

---

## Class actuate.viewer.ParameterValue

**Description** The ParameterValue class is a JavaScript version of the com.actuate.schemas.ParameterValue class.

### Constructor

**Syntax** `actuate.parameter.getParameterValue( )`

Constructs a new ParameterValue object.

### Function summary

Table 7-50 lists the actuate.viewer.ParameterValue functions.

**Table 7-50** 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>setColumnName( )</code>  | Sets the name value           |
| <code>setValue( )</code>       | Sets the value value          |
| <code>setValueIsNull( )</code> | Sets the valueIsNull value    |

### getName

**Syntax** `string ParameterValue.getName( )`

Returns the name value.

**Returns** String. The name value.

**Example** To store the name of a viewer.ParameterValue object in a variable called vPVname, use code similar to the following:

```
var vPVname = myParameterValue.getName();
```

### getValue

**Syntax** `object ParameterValue.getValue( )`

Returns the value value.

**Returns** Object. The value value, a string or array of strings.

**Example** To store a ParameterValue's value in vPVvalue, use the following code:

```
var vPVvalue = myParamValue.getValue();
```

## getValuesNull

**Syntax** boolean ParameterValue.getValuesNull( )

Returns the valueIsNull value.

**Returns** Boolean. The valueIsNull value.

**Example** This example displays an alert with the valueIsNull of the ParameterValue object:

```
alert("Value is null: " + myParamValue.getValueIsNull());
```

## setColumnName

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

Sets the columnName value.

**Parameter** **columnName**

String. The column name.

**Example** To set the column name to "Motorcycles", use code similar to the following:

```
myParamValue.setColumnName ("Motorcycles") ;
```

## setValue

**Syntax** void ParameterValue.setValue(object value)

Sets the value. A value can be a string or an array of strings.

**Parameter** **value**

Object. The value for this ParameterValue object, a string or an array of strings.

**Example** To set the value for a ParameterValue to myValues, use the following code:

```
var myValues = myParamValue.setValue(myValues) ;
```

## setValuesNull

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

Sets the valueIsNull value.

**Parameter** **valuesNull**

Boolean. The valueIsNull value.

**Example** To set a ParameterValue's setValueIsNull to true, use the following code:

```
myParamValue.setValueIsNull (true) ;
```

---

## Class actuate.viewer.RenderOptions

**Description** The RenderOptions class specifies render options for the actuate.Viewer.downloadReport( ) function. Currently, the only supported option is multisheet.

### Constructor

**Syntax** actuate.Viewer.RenderOptions( )

Constructs a new RenderOptions object for the parent viewer.

### Function summary

Table 7-51 lists actuate.viewer.RenderOptions functions.

**Table 7-51** actuate.viewer.RenderOptions functions

| Function     | Description                                |
|--------------|--------------------------------------------|
| getOptions() | Returns whether mouse scrolling is enabled |
| setOption()  | Returns whether mouse panning is enabled   |

### getOptions

**Syntax** Object[ ] RenderOptions.getOptions( )

Returns the render options map.

**Returns** Array, arranged in string and object pairs corresponding to option names and option values.

**Example** This example displays an alert box with the options status of render options:

```
alert("Rendering Options: " + options.getOptions());
```

### setOption

**Syntax** void RenderOptions.setOption(string option, boolean value)

Specifies a render option and its setting.

**Parameters** **option**

String. The permitted value is actuate.viewer.RenderOptions.IS\_MULTISHEET, which is used for xls format download only.

**value**

Boolean. Enabled value for IS\_MULTISHEET. True indicates that the xls format file has multiple worksheets.

**Example** To disable multisheet for the options object, use code similar to the following:

```
options.setOption(actuate.viewer.RenderOptions.IS_MULTISHEET,
 false);
```

---

## Class actuate.viewer.ScrollPanel

**Description** A container for a scrolling content panel in a viewer, which includes the scroll panel control, as shown in Figure 7-1.



**Figure 7-1** 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 7-52 lists actuate.viewer.ScrollPanel functions.

**Table 7-52** 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>setMouseScrollingEnabled( )</code> | Enables mouse scrolling                    |
| <code>setPanInOutEnabled( )</code>       | Enables panning                            |

### getMouseScrollingEnabled

**Syntax** `boolean ScrollPanel.getMouseScrollingEnabled( )`

Returns true when mouse scrolling is enabled.

**Returns** Boolean.

**Example** This example displays an alert with the mouse scrolling status of a scroll panel:

```
alert("Mouse scrolling enabled: " +
 sPanel.getMouseScrollingEnabled());
```

## getPanInOutEnabled

**Syntax** boolean ScrollPanel.getPanInOutEnabled( )

Returns true when panning in and out is enabled.

**Returns** Boolean.

**Example** This example displays an alert with the panning in and out status of a scroll panel:

```
alert("Panning enabled: " + scrollPanel.getPanInOutEnabled());
```

## setMouseScrollingEnabled

**Syntax** void ScrollPanel.setMouseScrollingEnabled(boolean enabled)

Enables mouse scrolling for this scroll panel.

**Parameter** **enabled**  
Boolean.

**Example** To disable mouse scrolling for sPanel, use code similar to the following:

```
sPanel.setMouseScrollingEnabled(false);
```

## setPanInOutEnabled

**Syntax** void ScrollPanel.setPanInOutEnabled(boolean enabled)

Enables panning in and out for this scroll panel.

**Parameter** **enabled**  
Boolean.

**Example** To disable panning for the sPanel object, use code similar to the following:

```
sPanel.setPanInOutEnabled(false);
```

---

## 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 7-53 lists actuate.viewer.SelectedContent functions.

**Table 7-53** actuate.viewer.SelectedContent functions

| Function              | Description                                              |
|-----------------------|----------------------------------------------------------|
| getColumnIndex( )     | Returns the currently selected table column index number |
| getSelectedElement( ) | Returns a copy of the currently selected element         |

### 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 selected column, use the following code:

```
var index = selected.getColumnIndex();
```

### getSelectedElement

**Syntax** object SelectedContent.getSelectedElement( )

Returns an instance of the currently selected element. The instance can be one of the following objects:

- actuate.report.Chart
- actuate.report.DataItem

- actuate.report.Label
- actuate.report.Table
- actuate.report.TextItem

To determine the object type, use the Object.getType( ) function. The type strings for the above objects are Chart, Data, Label, Table, or Text, respectively.

**Returns** Object. An instance of the currently selected element.

**Example** To retrieve and store a label bookmark if a selected element is a label, use the following code:

```
var selected = selected.getColumnIndex();
if (selected.getType() == "Label"){
 var bmark = Object.getBookmark();
}
```

---

## Class actuate.viewer.UIConfig

**Description** The UIConfig class specifies feature availability for the viewer.

### Constructor

**Syntax** void actuate.viewer.UIConfig( )

Generates a new UIConfig object to manage the content panel for the viewer. By default, the content panel is an actuate.viewer.ScrollPanel object with ScrollControl, PanInOut, and MouseScrolling enabled.

### Function summary

Table 7-54 lists actuate.viewer.UIConfig functions.

**Table 7-54** 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                   |

### getContentPanel

**Syntax** object UIConfig.getContentPanel( )

Returns the content panel object.

**Returns** Object. Valid objects are actuate.viewer.BrowserPanel, actuate.viewer.ScrollPanel, and null. A null value indicates a content panel configured with the browser scroll bar enabled.

**Example** To retrieve and store the content panel from the viewer, use the following code:

```
var contentpanel = viewer.getUIConfig().getContentPanel();
```

### getShowToc

**Syntax** boolean UIConfig.getShowToc( )

Returns the showToc flag.

**Returns** Boolean.

**Example** To determine if the showToc flag is set to true, use the following code:

```
if (!viewer.getUIConfig().getShowToc()){ ...}
```

## setContentPanel

**Syntax** void UIConfig.setContentPanel(objectcontentPanel)

Sets the content panel for the viewer.

**Parameter** **contentPanel**

Object. Valid objects are actuate.viewer.BrowserPanel, actuate.viewer.ScrollPanel, and null. A null value sets a content panel configured with the browser scroll bar enabled.

**Example** To set the content panel to BrowserPanel if it is null, use the following code:

```
var contentpanel = viewer.getUIConfig().getContentPanel();
if (contentpanel == null){
 var newconfig = viewer.getUIConfig();
 newconfig.setContentPanel(new actuate.viewer.BrowserPanel());
 viewer.setUIConfig(newconfig);
}
```

## setShowToc

**Syntax** void UIConfig setShowToc(boolean showToc)

Sets the showToc flag.

**Parameter** **showToc**

Boolean.

**Example** To hide the Toc in the UI, use the following code:

```
var newconfig = viewer.getUIConfig();
newconfig.setShowToc(false);
viewer.setUIConfig(newconfig);
```

---

## Class actuate.viewer.UIOptions

**Description** The UIOptions class specifies feature availability for the viewer object.

### Constructor

**Syntax** void actuate.viewer.UIOptions( )

Generates a new UIOptions object to manage the features of the viewer.

### Function summary

Table 7-55 lists actuate.viewer.UIOptions functions.

**Table 7-55** actuate.viewer.UIOptions functions

| Function                 | Description                                  |
|--------------------------|----------------------------------------------|
| enableAdvancedSort()     | Enables the advanced sort feature            |
| enableAggregation()      | Enables the aggregation feature              |
| enableCalculatedColumn() | Enables the calculated column feature        |
| enableChartProperty()    | Enables the chart properties feature         |
| enableChartSubType()     | Enables the chart subtype selection          |
| enableCollapseExpand()   | Enables the collapse/expand feature          |
| enableColumnEdit()       | Enables the column editing feature           |
| enableColumnResize()     | Enables the column resizing feature          |
| enableContentMargin()    | Enables the content margin feature           |
| enableDataAnalyzer()     | Enables the Interactive Crosstabs feature    |
| enableDataExtraction()   | Enables the data extraction feature          |
| enableEditReport()       | Enables the report editing feature           |
| enableExportReport()     | Enables the export report feature            |
| enableFacebookComments() | Enables the Facebook comments feature        |
| enableFilter()           | Enables the filter feature                   |
| enableFlashGadgetType()  | Enables the Flash gadget type change feature |
| enableFormat()           | Enables the format editing feature           |
| enableGroupEdit()        | Enables the group editing feature            |
| enableHideShowItems()    | Enables the hide/show item feature           |
| enableHighlight()        | Enables the highlight feature                |
| enableHoverHighlight()   | Enables the hover highlight feature          |

**Table 7-55** actuate.viewer.UIOptions functions

| Function                   | Description                                     |
|----------------------------|-------------------------------------------------|
| enableLaunchViewer()       | Enables the launch viewer feature               |
| enableLinkToThisPage()     | Enables the "link to this page" feature         |
| enableMainMenu()           | Enables the main menu feature                   |
| enableMoveColumn()         | Enables column moving                           |
| enablePageBreak()          | Enables the page break editing feature          |
| enablePageNavigation()     | Enables the page navigation feature             |
| enableParameterPage()      | Enables the parameter page feature              |
| enablePrint()              | Enables the print feature                       |
| enableReorderColumns()     | Enables the column reordering                   |
| enableRowResize()          | Enables row resizing                            |
| enableSaveDesign()         | Enables the report design save feature          |
| enableSaveDocument()       | Enables the report document save feature        |
| enableShowToolTip()        | Enables the show tooltip feature                |
| enableSort()               | Enables the sort feature                        |
| enableSuppressDuplicate()  | Enables the duplication suppression feature     |
| enableSwitchView()         | Enables the switch view feature                 |
| enableTextEdit()           | Enables the text editing feature                |
| enableTOC()                | Enables the table of contents feature           |
| enableToolBar()            | Enables the toolbar feature                     |
| enableToolbarContextMenu() | Enables the toolbar context menu feature        |
| enableToolbarHelp()        | Enables the toolbar help feature                |
| enableTopBottomNFilter()   | Enables the top N and bottom N filter feature   |
| enableUndoRedo()           | Enables the undo and redo feature               |
| getFeatureMap()            | Returns a list of enabled and disabled features |

## enableAdvancedSort

**Syntax** void UIOptions.enableAdvancedSort(boolean enabled)

Enables or disables the advanced sort feature.

**Parameter** **enabled**  
Boolean. True enables this option.

**Example** To disable the advanced sort feature, use code similar to the following:

```
viewerOpts.enableAdvancedSort(false);
```

## enableAggregation

**Syntax** void UIOptions.enableAggregation(boolean enabled)

Enables or disables the aggregation feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the aggregation feature, use code similar to the following:

```
viewerOpts.enableAggregation(false);
```

## enableCalculatedColumn

**Syntax** void UIOptions.enableCalculatedColumn(boolean enabled)

Enables or disables the calculated column feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the calculated column feature, use code similar to the following:

```
viewerOpts.enableCalculatedColumn(false);
```

## enableChartProperty

**Syntax** void UIOptions.enableChartProperty(boolean enabled)

Enables or disables the chart properties feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the chart properties feature, use code similar to the following:

```
viewerOpts.enableChartProperty(false);
```

## enableChartSubType

**Syntax** void UIOptions.enableChartSubType(boolean enabled)

Enables or disables the chart subtype selection feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the chart subtype selection feature, use code similar to the following:

```
viewerOpts.enableChartSubType(false);
```

## enableCollapseExpand

**Syntax** void UIOptions.enableCollapseExpand(boolean enabled)

Enables or disables the collapse/expand feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the collapse/expand feature, use code similar to the following:

```
viewerOpts.enableCollapseExpand(false);
```

## enableColumnEdit

**Syntax** void UIOptions.enableColumnEdit(boolean enabled)

Enables or disables the column editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the column editing feature, use code similar to the following:

```
viewerOpts.enableColumnEdit(false);
```

## enableColumnResize

**Syntax** void UIOptions.enableColumnResize(boolean enabled)

Enables or disables the column resizing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the column resizing feature, use code similar to the following:

```
viewerOpts.enableColumnResize(false);
```

## enableContentMargin

**Syntax** void UIOptions.enableContentMargin(boolean enabled)

Enables or disables the content margin feature.

**Parameter** **enabled**

Boolean. True enables this option.

## actuate.viewer.UIOptions

**Example** To disable the content margin feature, use code similar to the following:

```
viewerOpts.enableContentMargin(false) ;
```

### enableDataAnalyzer

**Syntax** void UIOptions.enableDataAnalyzer(boolean enabled)

Enables or disables the Interactive Crosstabs feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the Interactive Crosstabs feature, use code similar to the following:

```
viewerOpts.enableDataAnalyzer(false) ;
```

### enableDataExtraction

**Syntax** void UIOptions.enableDataExtraction(boolean enabled)

Enables or disables the data extraction feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the data extraction feature, use code similar to the following:

```
viewerOpts.enableDataExtraction(false) ;
```

### enableEditReport

**Syntax** void UIOptions.enableEditReport(boolean enabled)

Enables or disables the report editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the report editing feature, use code similar to the following:

```
viewerOpts.enableEditReport(false) ;
```

### enableExportReport

**Syntax** void UIOptions.enableExportReport(boolean enabled)

Enables or disables the export report feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the export report feature, use code similar to the following:

```
viewerOpts.enableExportReport(false);
```

## enableFacebookComments

**Syntax** void UIOptions.enableFacebookComments(boolean enabled)

Enables or disables the Facebook comments feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the Facebook comments feature, use code similar to the following:

```
viewerOpts.enableFacebookComments(false);
```

## enableFilter

**Syntax** void UIOptions.enableFilter(boolean enabled)

Enables or disables the filter feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the filter feature, use code similar to the following:

```
viewerOpts.enableFilter(false);
```

## enableFlashGadgetType

**Syntax** void UIOptions.enableFlashGadgetType(boolean enabled)

Enables or disables the Flash gadget type change control.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the Flash gadget type change control, use code similar to the following:

```
viewerOpts.enableFlashGadgetType(false);
```

## enableFormat

**Syntax** void UIOptions.enableFormat(boolean enabled)

Enables or disables the format editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

## actuate.viewer.UIOptions

**Example** To disable the format editing feature, use code similar to the following:

```
viewerOpts.enableFormat(false);
```

### enableGroupEdit

**Syntax** void UIOptions.enableGroupEdit(boolean enabled)

Enables or disables the group editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the group editing feature, use code similar to the following:

```
viewerOpts.enableGroupEdit(false);
```

### enableHideShowItems

**Syntax** void UIOptions.enableHideShowItems(boolean enabled)

Enables or disables the hide/show item feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the hide/show feature, use code similar to the following:

```
viewerOpts.enableHideShowItems(false);
```

### enableHighlight

**Syntax** void UIOptions.enableHighlight(boolean enabled)

Enables or disables the highlight feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the highlight feature, use code similar to the following:

```
viewerOpts.enableHighlight(false);
```

### enableHoverHighlight

**Syntax** void UIOptions.enableHoverHighlight(boolean enabled)

Enables or disables the hover highlight feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the hover highlight feature, use code similar to the following:

```
viewerOpts.enableHoverHighlight(false);
```

## enableLaunchViewer

**Syntax** void UIOptions.enableLaunchViewer(boolean enabled)

Enables or disables the launch viewer feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the launch viewer feature, use code similar to the following:

```
viewerOpts.enableLaunchViewer(false);
```

## enableLinkToThisPage

**Syntax** void UIOptions.enableLinkToThisPage(boolean enabled)

Enables or disables the "link to this page" feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the "link to this page" feature, use code similar to the following:

```
viewerOpts.enableLinkToThisPage(false);
```

## enableMainMenu

**Syntax** void UIOptions.enableMainMenu(boolean enabled)

Enables or disables the main menu feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the main menu feature, use code similar to the following:

```
viewerOpts.enableMainMenu(false);
```

## enableMoveColumn

**Syntax** void UIOptions.enableMoveColumn(boolean enabled)

Enables or disables the option to move columns.

**Parameter** **enabled**

Boolean. True enables this option.

## actuate.viewer.UIOptions

**Example** To disable the option to move columns, use code similar to the following:

```
viewerOpts.enableMoveColumn(false) ;
```

## enablePageBreak

**Syntax** void UIOptions.enablePageBreak(boolean enabled)

Enables or disables the page break editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the page break editing feature, use code similar to the following:

```
viewerOpts.enablePageBreak(false) ;
```

## enablePageNavigation

**Syntax** void UIOptions.enablePageNavigation(boolean enabled)

Enables or disables the page navigation feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the page navigation feature, use code similar to the following:

```
viewerOpts.enablePageNavigation(false) ;
```

## enableParameterPage

**Syntax** void UIOptions.enableParameterPage(boolean enabled)

Enables or disables the parameter page feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the parameter page feature, use code similar to the following:

```
viewerOpts.enableParameterPage(false) ;
```

## enablePrint

**Syntax** void UIOptions.enablePrint(boolean enabled)

Enables or disables the print feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the print feature, use code similar to the following:

```
viewerOpts.enablePrint(false) ;
```

## enableReorderColumns

**Syntax** void UIOptions.enableReorderColumns(boolean enabled)

Enables or disables the column reordering feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the column reordering feature, use code similar to the following:

```
viewerOpts.enableReorderColumns(false) ;
```

## enableRowResize

**Syntax** void UIOptions.enableRowResize(boolean enabled)

Enables or disables row resizing.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable row resizing, use code similar to the following:

```
viewerOpts.enableRowResize(false) ;
```

## enableSaveDesign

**Syntax** void UIOptions.enableSaveDesign(boolean enabled)

Enables or disables the report design save feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the report design save feature, use code similar to the following:

```
viewerOpts.enableSaveDesign(false) ;
```

## enableSaveDocument

**Syntax** void UIOptions.enableSaveDocument(boolean enabled)

Enables or disables the report document save feature.

**Parameter** **enabled**

Boolean. True enables this option.

## actuate.viewer.UIOptions

**Example** To disable the report document save feature, use code similar to the following:

```
viewerOpts.enableSaveDocument (false) ;
```

### enableShowToolTip

**Syntax** void UIOptions.enableShowToolTip(boolean enabled)

Enables or disables the showing of tooltips.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the showing of tooltips, use code similar to the following:

```
viewerOpts.enableShowToolTip (false) ;
```

### enableSort

**Syntax** void UIOptions.enableSort(boolean enabled)

Enables or disables the sort feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the sort feature, use code similar to the following:

```
viewerOpts.enableSort (false) ;
```

### enableSuppressDuplicate

**Syntax** void UIOptions.enableSuppressDuplicate(boolean enabled)

Enables or disables the duplication suppression feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the duplication suppression feature, use code similar to the following:

```
viewerOpts.enableSuppressDuplicate (false) ;
```

### enableSwitchView

**Syntax** void UIOptions.enableSwitchView(boolean enabled)

Enables or disables the switch view feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the switch view feature, use code similar to the following:

```
viewerOpts.enableSwitchView(false);
```

## enableTextEdit

**Syntax** void UIOptions.enableTextEdit(boolean enabled)

Enables or disables the text editing feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the text editing feature, use code similar to the following:

```
viewerOpts.enableTextEdit(false);
```

## enableTOC

**Syntax** void UIOptions.enableTOC(boolean enabled)

Enables or disables the table of contents feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the table of contents feature, use code similar to the following:

```
viewerOpts.enableTOC(false);
```

## enableToolBar

**Syntax** void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the toolbar feature, use code similar to the following:

```
viewerOpts.enableToolBar(false);
```

**Example** This code initializes a new viewer display, using enableToolBar(false) to disable the toolbar:

```
function initDisplay(){
 var uioptions = new actuate.viewer.UIOptions();
 viewer = new actuate.Viewer("viewerpane");
 var viewerwidth = 800;
 var viewerheight = 600;
 viewer.setWidth(viewerwidth);
 viewer.setHeight(viewerheight);
 uioptions.enableToolBar(false);
}
```

```
actuate.viewer.UIOptions
```

```
 viewer.setUIOptions(uioptions);
 document.getElementById("display").disabled = false;
}
```

## enableToolbarContextMenu

**Syntax** void UIOptions.enableToolbarContextMenu(boolean enabled)

Enables or disables the context menu feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** This code initializes a new viewer display, using enableToolBarHelp(true) to enable the toolbar help feature:

```
function initDisplay(){
 var uioptions = new actuate.viewer.UIOptions();
 viewer = new actuate.Viewer("viewerpane");
 var viewerwidth = 800;
 var viewerheight = 600;
 viewer.setWidth(viewerwidth);
 viewer.setHeight(viewerheight);
 uioptions.enableToolBar(true);
 uioptions.enableToolbarHelp(true);
 viewer.setUIOptions(uioptions);
 document.getElementById("display").disabled = false;
}
```

## enableToolbarHelp

**Syntax** void UIOptions.enableToolbarHelp(boolean enabled)

Enables or disables the toolbar help feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the toolbar help feature, use code similar to the following:

```
viewerOpts.enableToolbarHelp(false);
```

## enableTopBottomNFilter

**Syntax** void UIOptions.enableTopBottomNFilter(boolean enabled)

Enables or disables the top N and bottom N filter feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the top N and bottom N filter feature, use code similar to the following:

```
viewerOpts.enableTopBottomNFilter(false);
```

## enableUndoRedo

**Syntax** void UIOptions.enableUndoRedo(boolean enabled)

Enables or disables the undo and redo feature.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** To disable the undo and redo feature, use code similar to the following:

```
viewerOpts.enableUndoRedo(false);
```

## getFeatureMap

**Syntax** object UIOptions.getFeatureMap()

Returns the features and their Boolean values as an associative array. This function makes the name of each feature an object property and sets the value of that property to the associated enabled Boolean value.

**Returns** Object.

## Class actuate.viewer.ViewerException

**Description** A container for an exception. ViewerException provides an object to pass to a handler when the user-defined ON\_EXCEPTION event occurs. It contains a reference to the element that generated the exception.

### Constructor

The ViewerException object is constructed when an ON\_EXCEPTION event occurs. The exceptions are divided into three types, which determine the contents of the Exception object. These types are:

- ERR\_CLIENT: Exception type for a client-side error
- ERR\_SERVER: Exception type for a server error
- ERR\_USAGE: Exception type for a JSAPI usage error

### Function summary

Table 7-56 lists actuate.viewer.ViewerException functions.

**Table 7-56** actuate.viewer.ViewerException functions

| Function           | Description                                          |
|--------------------|------------------------------------------------------|
| getElement( )      | Returns the element for which the exception occurred |
| getErrorMessage( ) | Returns the exception message                        |

### getElement

**Syntax** object ViewerException.getElement( )

Returns an instance of the element that caused the exception, if applicable. The instance can be an object of one of following types:

- actuate.report.Chart
- actuate.report.DataItem
- actuate.report.Label
- actuate.report.Table
- actuate.report.TextItem

To determine the object type, use the Object.getType( ) function. The type strings for the above objects are "Chart", "Data", "Label", "Table", or "Text", respectively.

**Returns** Object. An instance of the element that generated the exception.

**Example** This example displays the type of element that generated the exception in an alert box:

```
alert("Exception in " + vException.getElement.getType());
```

## getErrorMessage

**Syntax** string ViewerException.getErrorMessage( )

Returns the error message for the exception.

**Returns** String. A server error message.

**Example** This example displays the server error code in an alert box:

```
alert("Server error message: " + vException.getErrorMessage());
```

actuate.viewer.ViewerException

# 8

## BIRT Interactive Crosstabs API classes

This chapter contains the following topics:

- About the BIRT Interactive Crosstabs JavaScript API
- Interactive Crosstabs API reference
- Interactive Crosstabs JavaScript classes quick reference

---

## About the BIRT Interactive Crosstabs JavaScript API

The Interactive Crosstabs portion of the Actuate JavaScript API is a set of JavaScript classes that modify, analyze, and display data within cross tab elements. These classes are available to users of iHub Visualization Platform client. The Actuate JavaScript API functions that are described in this chapter invoke and control the Interactive Crosstabs viewer and elements that are associated with the viewer. The Interactive Crosstabs JavaScript can be placed within a web page or any other location where the Actuate JavaScript API interfaces with a cross tab.

The `actuate.xtabAnalyzer` class represents the Interactive Crosstabs viewer that contains cross tab information. Load the analyzer with `actuate.load()`.

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

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

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

Load the XTabAnalyzer and dialog components to prepare the `actuate.XTabAnalyzer` component for use within a web page. Call `actuate.XTabAnalyzer` functions to create and prepare an analytics cross tab. Call the `XTabAnalyzer.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 an Interactive Crosstabs 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 Interactive Crosstabs viewer launches in a pop-up window.

To load a cross tab or a data cube, use `setReportName()`.

```
ctViewer.setReportName("/Public/BIRT and BIRT Studio Examples
/Crosstab Sample Revenue.rptdocument");
```

The example code loads a report document that consists of a single data cube and cross tab. The report document can be loaded into the Interactive Crosstabs viewer directly.

To access a cross tab element that is part of a larger report, use the cross tab element's bookmark after setting the report name. A bookmark is set in a report designer or by an external function. Retrieve a cross tab element with `actuate.xtabanalyzer.PageContent.getCrossstabByBookmark()`. For example, the following code retrieves a cross tab with the bookmark SampleRevenue:

```
var content = ctViewer.getCurrentPageContent();
var crosstab = content.getCrossstabByBookmark("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.

---

## Interactive Crosstabs API reference

This section provides an alphabetic listing of the Interactive Crosstabs API classes.

The examples in this section consist of JavaScript functions usable by a typical web page. These examples use a sample report document called `reportfile.rptdocument`. The sample report document contains a cross tab that has been bookmarked within BIRT Designer Professional with the value of Sample Revenue. Use any equivalent file of that design. Place the Interactive Crosstabs viewer in the `acviewer` container. The `acviewer` container is a `<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></pre>
```

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();">
```

---

## Interactive Crosstabs JavaScript classes quick reference

Table 8-1 lists the Interactive Crosstabs JavaScript classes.

**Table 8-1** Actuate Interactive Crosstabs JavaScript classes

JavaScript class	Description
actuate.XTabAnalyzer	An Interactive Crosstabs viewer component that can be embedded in an HTML page
actuate.xtabalyzer.Crosstab	A cross tab element
actuate.xtabalyzer.Dimension	A data dimension
actuate.xtabalyzer.Driller	A helper class for drilling down through cross tab data
actuate.xtabalyzer.EventConstants	Global constants for Interactive Crosstabs events class
actuate.xtabalyzer.Exception	Exception object sent to calling function
actuate.xtabalyzer.Filter	Filter conditions to filter data
actuate.xtabalyzer.GrandTotal	A cross tab grand total
actuate.xtabalyzer.Level	A cross tab level
actuate.xtabalyzer.LevelAttribute	An attribute for a level
actuate.xtabalyzer.Measure	A data measure
actuate.xtabalyzer.MemberValue	Data as a member value
actuate.xtabalyzer.Options	Options for the cross tab
actuate.xtabalyzer.PageContent	The content shown in the Interactive Crosstabs viewer
actuate.xtabalyzer.ParameterValue	A cross tab parameter value
actuate.xtabalyzer.Sorter	Conditions for sorting data
actuate.xtabalyzer.SubTotal	A cross tab subtotal
actuate.xtabalyzer.Total	A cross tab total
actuate.xtabalyzer.UIOptions	Enables UI elements of Interactive Crosstabs

---

## Class actuate.XTabAnalyzer

**Description** The XTabAnalyzer class represents an Interactive Crosstabs viewer, used to view and operate a cross tab.

### Constructor

**Syntax** `actuate.XTabAnalyzer()`

Constructs a new Interactive Crosstabs object.

`actuate.XTabAnalyzer(object xtabContainer, actuate.xtabanalyzer.UIOptions uiOptions)`

`actuate.XTabAnalyzer(string xtabContainerId, actuate.xtabanalyzer.UIOptions uiOptions)`

Constructs a new Interactive Crosstabs object in the specified container.

**Parameters** **xtabContainer**

Object. A document object referencing the HTML `<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 8-2 lists `actuate.XTabAnalyzer` functions.

**Table 8-2** `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 8-2** actuate.XTabAnalyzer functions (continued)

Function	Description
getLeft()	Returns the viewer left margin
getParameterValues()	Returns the parameter values
getPosition()	Returns the CSS position attribute value
getTop()	Returns the viewer top margin
getTotalPageCount()	Returns the total page count
getUIOptions()	Returns the actuate.xtabanalyzer.UIOptions object assigned to this viewer
getViewer()	Gets a viewer within a container
getWidth()	Returns the viewer width
getXTabBookmark()	Returns the bookmark of the cross tab displayed in the viewer
getXTabId()	Returns the instance ID of the cross tab displayed in the viewer
isActive()	Checks if current viewer pop-up is active
isDashboard()	Checks if the current viewer pop-up is a dashboard
isInteractive()	Checks if the current viewer is interactive
registerEventHandler()	Registers an event handler
removeEventHandler()	Removes an event handler
reset()	Resets the viewer object
resizeTo()	Resizes the viewer
rollback()	Rolls back all changes in the viewer and refreshes its content
setGadgetId	Sets the gadget id of the cross tab
setHeight()	Sets the viewer height
setIVMode()	Sets whether the viewer is in IV mode
setLeft()	Sets the viewer left margin
setOnClosed()	Sets callback when the pop-up window is closed
setPageNum()	Sets the page number
setParameterValues()	Sets the parameter values
setPosition()	Sets the CSS position attribute

*(continues)*

**Table 8-2** actuate.XTabAnalyzer functions (continued)

Function	Description
setReportletDocumentMode( )	Sets a Reportlet to document mode
setReportName( )	Sets the report to load into the interactive cross tab
setService( )	Sets the iHub System 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

## commit

**Syntax** void XTabAnalyzer.commit(function callback)

Commits all design changes to a generated document as a single operation. If ivMode is not set to true, call setIVMode( ) to set the value of ivMode to true before calling commit( ).

**Parameter** **callback**

Function. The callback function called after commit finishes.

**Example** This example opens a design with a cross tab and pivots the cross tab:

```
function pivot(){
// make a change to the cross tab.
 crosstab.pivot();
 crosstab.submit();
 viewer.commit();
}
```

## forceSoftRestart

**Syntax** void XTabAnalyzer.forceSoftRestart( )

Forces the viewer to restart.

**Example** This example restarts the viewer:

```
this.onclick = function(event) {
 forceSoftRestart();
}
```

## getCurrentPageContent

**Syntax** actuate.xtabanalyzer.PageContent XTabAnalyzer.getCurrentPageContent( )

Returns the Current Page Content object.

**Returns** actuate.xtabanalyzer.PageContent object. Content from the current page.

**Example** This example retrieves the cross tab from the current page:

```
function getCrosstab(analyzerViewer) {
 var content = analyzerViewer.getCurrentPageContent();
 return content.getCrosstabByBookmark();
}
```

## getCurrentPageNum

**Syntax** integer XTabAnalyzer.getCurrentPageNum( )

Returns the current page number.

**Returns** Integer. The current page number.

**Example** This example retrieves the page number:

```
function retrievePageNum(){
 return analyzerViewer.getCurrentPageNum();
}
```

## getGadgetId

**Syntax** string XTabAnalyzer.getGadgetId( )

Returns the gadget ID of the shown cross tab. This function is used for dashboard integration.

**Returns** String. A gadget ID.

**Example** This example retrieves the gadget ID:

```
function retrieveGadgetID(){
 return analyzerViewer.getGadgetId();
}
```

## getHeight

**Syntax** integer XTabAnalyzer.getHeight( )

Returns the height of the viewer.

**Returns** Integer. The height in pixels.

**Example** This example retrieves the current height of the viewer and doubles the height if the current height is lower than 630 pixels:

```
function doubleHeight(){
 var height = viewer.getHeight();
 if (height < 630){
 viewer.setHeight(height * 2);
 viewer.submit();
 }
}
```

## getLeft

**Syntax** integer XTabAnalyzer.getLeft( )

Returns the left margin of the viewer.

**Returns** Integer. The left margin in pixels.

**Example** This example retrieves the position of the viewer's left margin and moves the margin 20 pixels to the right if the left margin is fewer than 45 pixels from the left edge of the screen:

```
function moveLeftMargin(){
 var left = viewer.getLeft();
 if (left < 45){
 viewer.setLeft(left + 20);
 viewer.submit();
 }
}
```

## getParameterValues

**Syntax** actuate.xtabanalyzer.ParameterValue[ ] XTabAnalyzer.getParameterValues( )

Returns the parameter values.

**Returns** actuate.xtabanalyzer.ParameterValue[ ] or actuate.parameterParameterValue[ ]. An array of parameter values.

## getPosition

**Syntax** string XTabAnalyzer.getPosition( )

Returns the CSS position attribute for the viewer.

**Returns** String. The CSS position attribute.

**Example** This example changes the CSS positioning type from relative to absolute:

```
function changePosition() {
 if (viewer.getPosition() == 'relative'){
 viewer.setPosition('absolute');
 viewer.submit();
 }
}
```

## getTop

**Syntax** integer XTabAnalyzer.getTop( )

Returns the top margin of the viewer.

**Returns** Integer. The top margin in pixels.

**Example** This example retrieves the value for the viewer's top margin and moves the margin 20 pixels down the screen if the margin was fewer than 45 pixels from the top of the screen:

```
function moveTopMargin(){
 var top = viewer.getTop();
 if (top < 45){
 viewer.setTop(top + 20);
 viewer.submit();
 }
}
```

## getTotalPageCount

**Syntax** integer XTabAnalyzer.getTotalPageCount( )

Returns the total page count.

**Returns** Integer. The total number of pages.

**Example** This example displays an alert with the total page count from viewer:

```
alert("Total pages: " + viewer.getTotalPageCount());
```

## getUIOptions

**Syntax** actuate.xtabanalyzer.UIOptions getUIOptions( )

Returns the user interface options object for the cross tab analyzer. The UIOptions object specifies what features are used within the viewer.

## actuate.XTabAnalyzer

**Returns** actuate.xtabanalyzer.UIOptions object. Interactive Crosstabs user interface options.

**Example** This example retrieves the user interface options and sets one of the UIOptions values:

```
function resetUIOptions(){
 var options = viewer.getUIOptions();
 options.enableToolbar(false);
 viewer.setUIOptions(options);
}
```

## getViewer

**Syntax** static XTabAnalyzer.getViewer(HTMLElement container)

Returns a viewer by container. To retrieve the viewer for the current object, do not specify a container. This function is useful to retrieve the instance ID for a specific viewer when there are multiple viewers on a page.

**Parameter** **container**

HTMLElement. The container instance ID from which to retrieve the viewer.

**Returns** XTabAnalyzer object. The Interactive Crosstabs viewer.

**Example** This example retrieves the viewer:

```
function retrieveViewer(){
 return viewer.getViewer();
}
```

## getWidth

**Syntax** string XTabAnalyzer.getWidth( )

Returns the width value of the viewer.

**Returns** String. The width in pixels.

**Example** This example retrieves the width of the viewer, then alters it based on the size:

```
function doubleWidth(){
 var width = viewer.getWidth();
 if (width < 630){
 viewer.setWidth(width * 2);
 viewer.submit();
 }
}
```

## getXTabBookmark

**Syntax** string XTabAnalyzer.getXTabBookmark( )

Returns the bookmark name for the cross tab set to render in the viewer.

**Returns** String. The bookmark for a cross tab.

**Example** This example retrieves the bookmark that the cross tab is associated with, changes the bookmark, and resets the bookmark. This functionality supports the use of multiple cross tab elements within a single design.

```
function changeBookmark() {
 var oldBookMark = viewer.getXTabBookmark();
 viewer.setXTabBookmark("crosstab2");
 viewer.submit();
}
```

## getXTabId

**Syntax** string XTabAnalyzer.getXTabId()

Returns the current instance ID of the interactive cross tab. This function is useful in integration with Interactive Viewer and supports the ability of Interactive Viewer to obtain and use the interactive cross tab instance ID.

**Returns** String. An interactive cross tab instance ID.

**Example** This example retrieves the interactive cross tab instance ID:

```
function retrieveXTabId(myviewer){
 return myviewer.getXTabId();
}
```

## isActive

**Syntax** boolean XTabAnalyzer.isActive()

Returns true when a pop-up containing an interactive cross tab is active and false in all other cases.

**Returns** Boolean. True indicates an active interactive cross tab pop-up window.

**Example** This example checks if a viewer exists by checking two conditions: the viewer variable exists, or isActive() returns true. When both conditions fail, the example code creates a new viewer object within a container:

```
function checkViewer(){
 if(!viewer || !viewer.isActive()) {
 viewer = new actuate.XTabAnalyzer(container);
 }
}
```

## isDashboard

**Syntax** boolean XTabAnalyzer.isDashboard( )

Returns true when dashboard mode is active and false in all other cases.

**Returns** Boolean. True indicates dashboard mode.

## isInteractive

**Syntax** boolean XTabAnalyzer.isInteractive( )

Returns whether this Interactive Crosstabs Viewer is in Interactive mode.

**Returns** Boolean. True indicates dashboard mode.

**Example** This example displays whether myDataAnalyzer is interactive:

```
alert("Interactive mode: " + myDataAnalyzer.isInteractive());
```

## registerEventHandler

**Syntax** void XTabAnalyzer.registerEventHandler(string viewerEvent, function handler)

Registers an event handler for the specified event. This function throws actuate.xtabanalyzer.Exception when invalid arguments are passed.

**Parameters** **viewerEvent**

String. Specifies the event that triggers the handler call. For a list of supported events, see actuate.xtabanalyzer.EventConstants.

**handler**

Function. Called when the event occurs.

**Example** This example changes an event handler from one function to another:

```
function changeEventHandler(event) {
 viewer.removeEventHandler(actuate.xtabanalyzer.EventConstants
 .ON_CONTENT_CHANGED,
 oldChangedHandler);
 viewer.registerEventHandler(actuate.xtabanalyzer
 .EventConstants.ON_CONTENT_CHANGED,
 newChangedHandler);
}
```

## removeEventHandler

**Syntax** void XTabAnalyzer.removeEventHandler(string viewerEvent, function handler)

Removes an event handler from the specified event. This function throws actuate.xtabanalyzer.Exception when invalid arguments are passed.

**Parameters** **viewerEvent**

String. Specifies the event from which to remove the event handler. For a list of supported events, see `actuate.xtabanalyzer.EventConstants`.

**handler**

Function. The function to deregister from the event.

**Example** This example changes an event handler from one function to another:

```
function changeEventHandler(event){
 viewer.removeEventHandler(actuate.xtabanalyzer.EventConstants
 .ON_CONTENT_CHANGED,
 oldChangedHandler);
 viewer.registerEventHandler(actuate.xtabanalyzer
 .EventConstants.ON_CONTENT_CHANGED,
 newChangedHandler);
}
```

**reset****Syntax** `void XTabAnalyzer.reset()`

Resets the viewer to its initial state.

**Example** This example resets the viewer. All changes to the viewer made prior to this call are lost:

```
function resetViewer(){
 viewer.reset();
}
```

**resizeTo****Syntax** `void XTabAnalyzer.resizeTo(integer width, integer height)`

Resizes the viewer to the specified height and width.

**Parameters** **width**

Integer. The width in pixels.

**height**

Integer. The height in pixels.

**Example** This example resizes the viewer when the new width is fewer than 1000 pixels and the new height is fewer than 650 pixels:

```
function resizeViewer(width,height){
 if ((width < 1000) && (height < 650)){
 viewer.resizeTo(width,height);
 }
}
```

## rollback

**Syntax** void XTabAnalyzer.rollback(function callback)

Rolls back all changes in the viewer since the last commit( ) call and refreshes the viewer's content. The value of ivMode must be true for rollback( ) to function.

**Parameter** **callback**

Function. The callback function called after rollback finishes.

**Example** This example rolls back all changes to the viewer made since the last commit or submit function call:

```
function rollbackViewer(){
 viewer.rollback();
}
```

## setGadgetId

**Syntax** void XTabAnalyzer.setGadgetId(string gadgetId)

Sets the cross tab gadget ID. This function is used for dashboard integration.

**Parameter** **gadgetId**

String. The gadget ID used to render the cross tab.

**Example** This example sets the gadget ID:

```
function setGadgetID(id){
 viewer.setGadgetId(id);
}
```

## setHeight

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

Changes the height of the viewer.

**Parameter** **height**

Integer. The height in pixels.

**Example** This example retrieves the viewer's current height. When the current height is fewer than 630 pixels, the example code doubles the viewer's height.

```
function doubleHeight(){
 var height = viewer.getHeight();
 if (height < 630){
 height = height * 2;
 viewer.setHeight(height);
 viewer.submit();
 }
}
```

## setIVMode

**Syntax** void XTabAnalyzer.setIVMode(boolean ivMode)

Sets IVMode for the viewer. Integrating a Data Analytics viewer with the Interactive Viewer affects the undo/redo feature. When set to true, all changes to the Data Analytics viewer must be committed as one transaction. The Interactive Viewer can undo or redo the entire batch.

**Parameter**

**ivMode**

Boolean. Set to true if using IV mode.

**Example** This example sets IVMode for the viewer:

```
function setViewerMode (mode) {
 viewer.setIVMode (mode) ;
}
```

## setLeft

**Syntax** void XTabAnalyzer.setLeft(integer left)

Sets the position of the viewer's left margin.

**Parameter**

**left**

Integer. The left margin for the viewer in pixels.

**Example**

This example retrieves the left margin of the viewer and moves the margin 20 pixels to the right when the margin is less than 45 pixels from the edge of the screen:

```
function moveLeftMargin(){
 var left = viewer.getLeft();
 if (left < 45){
 viewer.setLeft(left + 20);
 viewer.submit();
 }
}
```

## setOnClosed

**Syntax** void XTabAnalyzer.setOnClosed(function callback)

Sets a callback function to call when a viewer pop-up closes.

**Parameter**

**callback**

Function. The function to call when the pop-up closes.

**Example**

This example checks to see if a pop-up window is active and sets a callback function to trigger when the pop-up closes:

```
function setPopupCloser(){
```

```
actuate.XTabAnalyzer
```

```
 if(viewer.isActive()){
 viewer.setOnClosed(closerCallbackFunctionName);
 }
}
```

## setPageNum

**Syntax** void XTabAnalyzer.setPageNum(function pageNum)

Sets the page number.

**Parameter** **pageNum**

Integer. The page number.

**Example** This example sets the sets the page number to the first page:

```
function setPageNumberToFirst(){
 if(viewer.isActive()){
 viewer.setPageNum(1);
 }
}
```

## setParameterValues

**Syntax** void XTabAnalyzer.setParameterValues(actuate.xtabalyzer.ParameterValue[ ] parameterValues)

Sets the parameter values.

**Parameter** **parameterValues**

actuate.xtabalyzer.ParameterValue[ ] or actuate.parameterParameterValue[ ]. An array of parameter values.

## setPosition

**Syntax** void XTabAnalyzer.setPosition(string position)

Sets the CSS position attribute.

**Parameter** **position**

String. The value for the CSS position attribute.

**Example** This example changes the type of CSS positioning in use:

```
function changePosition(){
 var pos = viewer.getPosition();
 if (pos == 'relative'){
 viewer.setPosition('absolute');
 viewer.submit();
 }
}
```

## setReportletDocumentMode

**Syntax** void XTabAnalyzer.setReportletDocumentMode(boolean reportletMode)

Sets whether the viewer displays documents as Reportlets.

**Parameter** **reportletMode**

Boolean. True indicates Reportlet display mode.

## setReportName

**Syntax** void XTabAnalyzer.setReportName(string reportName, string connectionHandle)

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.

**connectionHandle**

String. Optional. The unique identifier generated by iHub for a temporary report.

**Example** This example sets the report name to reportfile.rptdocument and reloads the Interactive Crosstabs viewer with its content:

```
function run(){
 container = document.getElementById("acviewer");
 viewer = new actuate.XTabAnalyzer(container);
 viewer.setReportName("reportfile.rptdocument");
 viewer.submit();
}
```

## setService

**Syntax** void XTabAnalyzer.setService(string iPortalURL, actuate.RequestOptions requestOptions)

Sets the Actuate web application URL. This function can request options for that URL.

**Parameters** **iPortalURL**

String. The URL of the Actuate web application.

**requestOptions**

actuate.RequestOptions object. Request options for the web application. This parameter is optional.

**Example** This example sets the service and request options:

```
function setServerOptions(URL,options) {
 viewer.setService(URL,options);
}
```

## setSupportSVG

**Syntax** void XTabAnalyzer.setSupportSVG(boolean svgFlag)

Sets a flag indicating whether or not the browser supports SVG.

**Parameter** **svgFlag**

Boolean. Flag indicating SVG support in the browser. This parameter's value is true when the browser supports SVG and false in all other cases.

**Example** This example sets the browser's level of SVG support:

```
function setSVG(flag) {
 viewer.setSupportSVG(flag);
}
```

## setUp

**Syntax** void XTabAnalyzer.setUp(integer top)

Sets the top margin for the viewer.

**Parameter** **top**

Integer. The top margin for the viewer in pixels.

**Example** This example retrieves the current top margin for the viewer and moves the margin 20 pixels down the screen when the current position of the margin is fewer than 45 pixels from the top of the screen:

```
function moveTopMargin() {
 var top = viewer.getTop();
 if (top < 45){
 top = top + 20;
 viewer.setTop(top);
 viewer.submit();
 }
}
```

## setUIOptions

**Syntax** void XTabAnalyzer.setUIOptions(actuate.xtabanalyzer.uioptions options)

Sets the user interface options enabled for the viewer.

**Parameter** **options**

Actuate.xtabanalyzer.uioptions object. The options object for the viewer.

**Example** This example retrieves the user interface options and sets one of the UIOptions values:

```
function resetUIOptions(){
 var options = viewer.getUIOptions();
```

```

 options.enableToolbar(false);
 viewer.setUIOptions(options);
 }
}
```

## setWidth

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

Sets the width for the viewer.

**Parameter** **width**

Integer. The width for the viewer in pixels.

**Example** This example retrieves the width of the viewer. When the viewer is fewer than 630 pixels wide, the example code doubles the viewer's width:

```

function doubleWidth(){
 var width = viewer.getWidth();
 if (width < 630){
 viewer.setWidth(width * 2);
 viewer.submit();
 }
}
```

## setXTabBookmark

**Syntax** void XTabAnalyzer.setXTabBookmark(string bookmark)

Sets the bookmark for a cross tab to render in the viewer.

**Parameter** **bookmark**

String. The bookmark for a cross tab.

**Example** This example retrieves the bookmark for the cross tab the viewer is associated with, changes the bookmark, and reloads the bookmark. This functionality enables the use of multiple cross tab elements within a single design.

```

function changeBookmark(){
 var oldBookMark = viewer.getXTabBookmark();
 viewer.setXTabBookmark("crosstab2");
 viewer.submit();
}
```

## 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.

## actuate.XTabAnalyzer

### Parameter iid

String. The instance ID.

**Example** This example sets the cross tab instance ID:

```
function setxtabInstance(id){
 viewer.setXTabId(id);
}
```

## submit

**Syntax** void XTabAnalyzer.submit(function callback, boolean rerun)

Submits requests to the server for the Interactive Crosstabs viewer. This method triggers an AJAX request to submit all pending operations for this object. The server returns a response after processing the pending operations. The results render on the page in the Interactive Crosstabs container. The submit( ) function throws an exception when another submit( ) operation is pending. A CONTENT\_CHANGED event fires when the Interactive Crosstabs content changes.

### Parameters

**callback**

Function. Optional. A function called when submit completes. This function receives the current XTabAnalyzer object as an input parameter.

### rerun

Boolean. Optional. Indicates whether re-run the report design when it refreshes. Default to true.

**Example** This example retrieves the left margin of the viewer and expands the margin. The change does not take effect until submit( ) executes. The submit( ) function calls the function in the submitCallback parameter when submit( ) finishes executing. The callback function contains any processing that must occur after submit( ) finishes. Do not place code after the submit( ) call in the same function because submit( ) is asynchronous.

```
function moveLeftMargin(){
 var left = viewer.getLeft();
 if (left < 45){
 viewer.setLeft(left + 20);
 viewer.submit(submitCallback);
 }
}
```

---

## Class actuate.xtabanalyzer.Crosstab

**Description** The actuate.xtabanalyzer.Crosstab class represents a cross tab report element.

### Constructor

**Syntax** `actuate.xtabanalyzer.Crosstab()`

Constructs a new cross tab object.

### Function summary

Table 8-3 lists actuate.xtabanalyzer.Crosstab functions.

**Table 8-3** 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 8-3** actuate.xtabanalyzer.Crosstab functions (continued)

Function	Description
pivot()	Pivots the cross tab
removeDimension()	Removes a dimension from the cross tab
removeMeasure()	Removes a measure from the cross tab
reorderDimension()	Reorders a dimension
reorderMeasure()	Reorders a measure
setFilters()	Sets the cross tab's filters
setSorters()	Sets the cross tab's sorters
setTotals()	Sets the cross tab's totals
showDetail()	Shows details to the lower level
submit()	Applies changes made to the cross tab

## addDimension

**Syntax** void Crosstab.addDimension(actuate.xtabanalyzer.Dimension dimension)  
Adds a dimension to the cross tab object.

**Parameter** **dimension**  
actuate.xtabanalyzer.Dimension object. The dimension to add.

**Example** This example adds a date-based, multi-level dimension to a cross tab:

```
function addDimension() {
 // Create a dimension for dates in the first column
 var dimension = new actuate.xtabanalyzer.Dimension();
 dimension.setIndex(0);
 dimension.setAxisType(actuate.xtabanalyzer.Dimension
 .COLUMN_AXIS_TYPE);
 dimension.setDimensionName("dates");

 // Create levels using levels from the data cube.
 var level = new actuate.xtabanalyzer.Level();
 level.setLevelName("year");
 dimension.addLevel(level);
 var level = new actuate.xtabanalyzer.Level();
 level.setLevelName("quarter");
 dimension.addLevel(level);

 // Add the dimension to the cross tab.
 crosstab.addDimension(dimension);
 crosstab.submit();
}
```

## addMeasure

**Syntax** void Crosstab.addMeasure(actuate.xtabanalyzer.Measure measure, integer options)

Adds a measure to the cross tab object.

**Parameters**

- measure** actuate.xtabanalyzer.Measure object. The measure to add.

### options

Integer. The options for the add measure operation. These options distinguish the origin of the function call, which can be from another dialog or directly from the Actuate JavaScript API.

**Example** This example adds a measure to a cross tab:

```
function addMeasure(){
 //Create a measure for revenue organized by date and product line.
 var measure = new actuate.xtabanalyzer.Measure();
 measure.setIndex(1);
 measure.setMeasureName("Quarter Rate");
 measure.setExpression("[revenue]/[revenue_SalesDate
 /year_Product/PRODUCTLINE]");

 // Apply the measure to the cross tab
 crosstab.addMeasure(measure);
 crosstab.submit();
}
```

In this example, the expression set with setExpression() is in EasyScript, which is described in *Using Actuate BIRT Designer Professional*.

## applyOptions

**Syntax** void Crosstab.applyOptions(string | actuate.xtabanalyzer.Options measureDirection, string rowMirrorStartingLevel, string columnMirrorStartingLevel, string emptyCellValue)

Sets measure direction, empty settings, row mirror starting level, column mirror starting level, and empty cell value.

**Parameters**

- measureDirection** String or actuate.xtabanalyzer.Options object. When measureDirection is a string, measureDirection is set to horizontal or vertical and the other parameters set options individually. When an actuate.xtabanalyzer.Options object is specified, all the options are set using settings from this object and applyOptions ignores all subsequent parameters.

## actuate.xtabanalyzer.Crosstab

### **rowMirrorStartingLevel**

String. Sets the mirror starting level empty setting for a row.

### **columnMirrorStartingLevel**

String. Sets the mirror starting level empty setting for a column.

### **emptyCellValue**

String. Sets the value of an empty cell.

## **changeMeasureDirection**

**Syntax** void Crosstab.changeMeasureDirection( )

Switches the measure direction between horizontal and vertical.

**Example** This example changes the measure direction:

```
function changeMeasureDirection(){
 if(crosstab){
 crosstab.changeMeasureDirection();
 crosstab.submit();
 }
}
```

## **clearFilters**

**Syntax** void Crosstab.clearFilters(actuate.xtabanalyzer.Level level, String filterType)

Clears the filters from a level.

**Parameters** **level**

actuate.xtabanalyzer.Level object. Optional. The level from which to clear the filters. To clear all filters, do not specify a level.

### **filterType**

String. Optional. The filter type. To clear all filter types, do not specify a filter type.

**Example** This example clears the filters from the level filterLevel:

```
function clearLevelFilters(){
 if(crosstab){
 crosstab.clearFilters("filterLevel");
 crosstab.submit();
 }
}
```

## **drill**

**Syntax** void Crosstab.drill(actuate.xtabanalyzer.Driller driller)

Drills up or down a dimension level. Removes all drill/filter conditions defined on specified dimension first, then adds new drill/filter conditions.

**Parameter****driller**

actuate.xtabanalyzer.Driller object. The driller object specifies drill conditions on a dimension.

**Example**

This example drills to a level within a dimension. Any existing drill conditions are replaced.

```
function drillToDimension(memberVal) {
 var driller = new actuate.xtabanalyzer.Driller();
 driller.setAxisType(actuate.xtabanalyzer.Dimension
 .ROW_AXIS_TYPE);
 driller.addMember(memberVal);
 myCrosstab.drill(driller);
 myCrosstab.submit();
}
```

## drillDown

**Syntax**

void Crosstab.drillDown(actuate.xtabanalyzer.Driller driller)

Drills down a dimension level. This method updates the drill conditions specified in the Driller object and leaves all other conditions in place.

**Parameter****driller**

actuate.xtabanalyzer.Driller object. A drill condition object.

**Example**

This example drills down a level within a dimension. Any existing drill conditions are unchanged.

```
function drillToDimension(memberVal) {
 var driller = new actuate.xtabanalyzer.Driller();
 driller.setAxisType(actuate.xtabanalyzer.Dimension
 .ROW_AXIS_TYPE);
 driller.addMember(memberVal);
 myCrosstab.drillDown(driller);
 myCrosstab.submit();
}
```

## drillUp

**Syntax**

void Crosstab.drillUp(actuate.xtabanalyzer.Driller driller)

Drills up a dimension level. This method updates the drill conditions specified in the Driller object and leaves all other conditions in place.

**Parameter****driller**

A drill condition object.

## actuate.xtabalyzer.Crosstab

- Example** This example drills up a level within a dimension. Any existing drill conditions are unchanged.

```
function drillToDimension(){
 var driller = new actuate.xtabalyzer.Driller();
 driller.setAxisType(actuate.xtabalyzer.Dimension
 .ROW_AXIS_TYPE);
 // Add the member list to the Driller. Add the Driller to the
 // crosstab.
 driller.addMember(memberVal);
 myCrosstab.drillUp(driller);
 myCrosstab.submit();
}
```

## editMeasure

- Syntax** void Crosstab.editMeasure(actuate.xtabalyzer.Meaure Measure, integer opts)  
Edits a measure in the Computed Measure view.

- Parameters** **Measure**  
actuate.xtabalyzer.Measure object. A measure to change.  
**opts**  
Integer. Optional. Options for the editMeasure function. These options distinguish the origin of the function call, 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.xtabalyzer.Measure();
 measure.setMeasureName("measureName");
 measure.setExpression("measureExpression");
 crosstab.editMeasure(measure);
 crosstab.submit();
 }
}
```

## getBookmark

- Syntax** string Crosstab.getBookmark()  
Returns the bookmark that is associated with the cross tab element.  
**Returns** String. The cross tab bookmark.  
**Example** The following code retrieves the bookmark that is associated with the cross tab object:

```

function getCrosstabBookmark(){
 var crosstabBookmark = crosstab.getBookmark();
 if(!crosstabBookmark){
 alert("No cross tab bookmark found!")
 return null;
 }
 return crosstabBookmark;
}

```

## getColumn

**Syntax** string[ ] Crosstab.getColumn(integer columnIndex)

Returns the table data by column index.

**Parameter** **columnIndex**

Integer. The column index, starting with 1.

**Returns** String[ ]. The column data as an array of strings. This function returns null when the value of columnIndex is out of range. This function only returns data from the current visible page.

**Example** The following code retrieves data from a data column:

```

function getColumnData(index,value){
 var columnData = crosstab.getColumn(index);
 if(!columnData){
 alert("Invalid column index!")
 return null;
 }
 return columnData[value];
}

```

## getData

**Syntax** String[ ] Crosstab.getData(boolean forceReparse)

Returns the data in a cross tab.

**Parameter** **forceReparse**

Boolean. Forces a cache refresh when true.

**Returns** String[ ]. The data from the cross tab as an array of strings.

## getHtmlDom

**Syntax** HTMLElement Crosstab.getHtmlDom()

Returns the HTML element DOM object.

**Returns** HTMLElement. The DOM element containing the cross tab.

## actuate.xtabanalyzer.Crosstab

**Example** The following code retrieves the DOM object and uses the DOM object to retrieve an element within the document:

```
function getContainer(containerName){
 var HTMLDom = crosstab.getHtmlDom();
 var container = HTMLDom.getElementById(containerName);
 return container;
}
```

## getPageContent

**Syntax** actuate.xtabanalyzer.PageContent Crosstab.getPageContent( )

Returns the page content from the current page to which this cross tab belongs. This function returns the same information as XTabAnalyzer.getCurrentPageContent( ).

**Returns** actuate.xtabanalyzer.PageContent. The report content.

**Example** This example retrieves the page content:

```
function retrievePageContent(){
 return crosstab.getPageContent();
}
```

## getRow

**Syntax** string[ ] Crosstab.getRow(integer rowIndex)

Returns table data based on row index.

**Parameter** **rowIndex**

Integer. The row index, starting with 1.

**Returns** String[ ]. The row data as an array of string values. This function returns null when the value of rowIndex is out of range. This function only returns data from the current visible page.

**Example** The following code retrieves data from a data row:

```
function getRowData(index,value){
 var rowData = crosstab.getRow(index);
 if(!rowData){
 alert("Invalid row index!")
 return null;
 }

 return rowData[value];
}
```

## getType

**Syntax** string Crosstab.getType( )

Returns the report element type.

**Returns** String containing the value "Crosstab".

## hideDetail

**Syntax** void Crosstab.hideDetail(string levelName)

Hides details of the specified level.

**Parameter** **levelName**

String. The full name of a dimension level to hide.

**Example** This example hides lower level details in a level:

```
function hideDetail(){
 if(crosstab){
 var levelName = "rollLevelName";
 crosstab.hideDetail(levelName);
 crosstab.submit();
 }
}
```

## pivot

**Syntax** void Crosstab.pivot( )

Pivots the cross tab.

**Example** This example pivots a cross tab:

```
function pivot(crosstab){
 crosstab.pivot();
 crosstab.submit();
}
```

## removeDimension

**Syntax** void Crosstab.removeDimension(object dimension, integer axisType, integer[ ] levels)

Removes a dimension from the cross tab.

**Parameters** **dimension**

actuate.xtabanalyzer.dimension object, a dimension index, or a dimension name. The dimension to remove.

## actuate.xtabanalyzer.Crosstab

### 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();
 }
}
```

## removeMeasure

**Syntax** void Crosstab.removeMeasure(actuate.xtabanalyzer.Measure measure)

void Crosstab.removeMeasure(integer measure)

void Crosstab.removeMeasure(string measure)

Removes a measure from the cross tab.

**Parameter** **measure**

actuate.xtabanalyzer.measure object, index, or name. The measure to remove.

**Example** This example removes a measure from a cross tab:

```
function removeMeasure(){
 crosstab.removeMeasure("measureName");
 crosstab.submit();
}
```

## 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();
}
```

**reorderMeasure**

**Syntax** void Crosstab.reorderMeasure(actuate.xtabanalyzerMeasure measure,  
integer newIndex)

void Crosstab.reorderMeasure(integer measure,integer newIndex)

void Crosstab.reorderMeasure(string measure,integer newIndex)

Reorders a measure within a cross tab.

**Parameters** **measure**  
actuate.xtabanalyzer.Measure object, or a measure index or a measure name. The measure to reorder.

**newIndex**

The new index for the measure.

**Example** This example reorders a measure:

```
function changeMeasureOrder() {
 var index = 6;
 var newIndex = 3;
 crosstab.reorderMeasure(index,newIndex);
```

## actuate.xtabanalyzer.Crosstab

```
 crosstab.submit();
}
```

## setFilters

**Syntax** void Crosstab.setFilters(actuate.xtabanalyzer.Filter[ ] filters)

Sets an array of filters for the cross tab.

**Parameter** **filters**

Array of actuate.xtabanalyzer.Filter objects. The filter conditions.

**Example** This example creates a Filter object and then places it into the cross tab:

```
function filterLevel(){
 var levelName = "levelName";
 var operator = actuate.xtabanalyzer.Filter.BETWEEN;
 var filterValue = "20000;50000";
 var filter = new actuate.xtabanalyzer.Filter(levelName,
 operator);
 filter.setValues(filterValue.split(";"));
 crosstab.setFilters(filter);
 crosstab.submit();
}
```

## setSorters

**Syntax** void Crosstab.setSorters(actuate.xtabanalyzer.Sorter[ ] sorters)

Sets an array of sorters for the cross tab.

**Parameter** **sorters**

Array of actuate.xtabanalyzer.Sorter objects. The sort settings.

**Example** This example creates a sorter and adds it to the cross tab:

```
function sortLevel(){
 var levelName = "levelName";
 var sortAscending = true;
 var sorter = new actuate.xtabanalyzer.Sorter(levelName);
 sorter.setAscending(sortAscending);
 crosstab.setSorters(sorter);
 crosstab.submit();
}
```

## setTotals

**Syntax** void Crosstab.setTotals(actuate.xtabanalyzer.GrandTotal[ ] grandTotals,  
actuate.xtabanalyzer.SubTotal[ ] subTotals)

Sets totals for the cross tab.

**Parameters** **grandTotals**

Array of actuate.xtabanalyzer.GrandTotal objects. Grand totals. To set a subtotal, set this parameter to null.

**subTotals**

Array of actuate.xtabanalyzer.SubTotal objects. Subtotals.

**Example** This example adds a grand total to a cross tab:

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

 var total = new actuate.xtabanalyzer.Total();
 total.setMeasureIndex(1);
 total.setAggregationFunction("SUM");
 total.setEnabled(true);
 grandTotal.addTotal(total);

 crosstab.setTotals(grandTotal);
 crosstab.submit();
}
```

**showDetail****Syntax** void Crosstab.showDetail(string axisType)

Shows a level of detail within a cross tab.

**Parameter** **axisType**

String. The dimension axis type. Axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN\_AXIS\_TYPE
- actuate.xtabanalyzer.Dimension.ROW\_AXIS\_TYPE

**Example** This example uses showDetail to expose extra detail on a level:

```
function showDetail(){
 var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
 crosstab.showDetail(axisType);
 crosstab.submit();
}
```

**submit****Syntax** void Crosstab.submit(function callback)

Applies the changes made to this element. This is an asynchronous operation.

## actuate.xtabanalyzer.Crosstab

### Parameter **callback**

Function. Optional. The function called when submit( ) completes. This function receives the current XTabAnalyzer object as an input parameter.

**Example** This example uses submit( ) to confirm changes to the cross tab:

```
function showDetail(crosstab) {
 var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
 crosstab.showDetail(axisType);
 crosstab.submit();
}
```

---

## Class actuate.xtabanalyzer.Dimension

**Description** The Dimension class specifies a cross tab Dimension object.

### Constructor

**Syntax** `actuate.xtabanalyzer.Dimension( )`

The Dimension class is used to specify a Dimension object.

### Function summary

Table 8-4 lists actuate.xtabanalyzer.Dimension functions.

**Table 8-4** 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

### addLevel

**Syntax** `void Dimension.addLevel(actuate.xtabanalyzer.Level level)`

Adds a level to the dimension.

**Parameter** **level**

`actuate.xtabanalyzer.Level` object. A level to add to the dimension.

**Example** This example adds a level to a dimension:

```
function addLvl(dimension, levelName) {
```

## actuate.xtabanalyzer.Dimension

```
var level = new actuate.xtabanalyzer.Level();
level.setLevelName(levelName);
dimension.addLevel(level);
}
```

### getAxisType

**Syntax** integer Dimension.getAxisType( )

Returns the axis type for the dimension.

**Returns** Integer. The axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN\_AXIS\_TYPE
- actuate.xtabanalyzer.Dimension.ROW\_AXIS\_TYPE

**Example** This example retrieves and sets the axis type:

```
function swapAxis(dimension){
 if (dimension.getAxisType() ==
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE) {
 dimension.setNewAxisType(
 actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
 } else {
 dimension.setNewAxisType(
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
 }
}
```

### getDimensionName

**Syntax** string Dimension.getDimensionName( )

Returns the name of this dimension.

**Returns** String. The dimension name.

**Example** This example retrieves the dimension name:

```
function getDimName(dimension) {
 if(dimension){
 return dimension.getDimensionName();
 }
 return null;
}
```

### getIndex

**Syntax** integer Dimension.getIndex( )

Returns the dimension index.

**Returns** Integer. The dimension index.

**Example** This example retrieves and increments the index:

```
function incrementIndex(dimension) {
 var newIndex = dimension.getIndex() + 1;
 dimension.setNewIndex(newIndex) ;
}
```

## getLevels

**Syntax** actuate.xtabanalyzer.Level[ ] Dimension.getLevels( )

Returns the dimension levels.

**Returns** actuate.xtabanalyzer.Level[ ]. Array of dimension levels.

**Example** This example retrieves the dimension levels:

```
function getDimLevels(dimension) {
 if(dimension){
 return dimension.getLevels();
 }
 return null;
}
```

## getNewAxisType

**Syntax** integer Dimension.getNewAxisType( )

Returns the new axis type.

**Returns** Integer containing the new axis type.

**Example** This example retrieves the new axis type:

```
function getNewDimAxis(dimension) {
 if(dimension){
 return dimension.getNewAxisType();
 }
 return null;
}
```

## getNewIndex

**Syntax** integer Dimension.getNewIndex( )

Returns the new index.

**Returns** Integer. The new index.

## actuate.xtabanalyzer.Dimension

**Example** This example retrieves the new index:

```
function getNewDimIndex(dimension) {
 if(dimension){
 return dimension.getNewIndex();
 }
 return null;
}
```

## setAxisType

**Syntax** void Dimension.setAxisType(integer axisType)

Sets the axis type when creating a new dimension. Use setNewAxisType( ) to change a dimension that already exists.

**Parameter** **axisType**

The axis type for the dimension. The axis type has the following legal values:

- actuate.xtabanalyzer.Dimension.COLUMN\_AXIS\_TYPE
- actuate.xtabanalyzer.Dimension.ROW\_AXIS\_TYPE

**Example** This example sets the axis type for a new dimension:

```
function setRowAxis(dimension) {
 dimension.setAxisType(
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
}
```

## setDimensionName

**Syntax** void Dimension.setDimensionName(string dimensionName)

Sets the name for a dimension during its creation.

**Parameter** **dimensionName**

String. The name of the dimension.

**Example** This example sets the dimension name to a value taken from a page element:

```
function setDimensionName(dimension) {
 var dimensionName = document.getElementById("dimensionName")
 .value;
 dimension.setDimensionName(dimensionName);
}
```

## setIndex

**Syntax** void Dimension.setIndex(integer index)

Sets the index for the dimension.

**Parameter** **index**

The index of the dimension.

**Example** This example sets the dimension index to a value taken from a page element:

```
function setDimensionIndex(dimension) {
 var dimensionIndex = document.getElementById("dimensionIndex")
 .value;
 dimension.setIndex(dimensionIndex);
}
```

**setLevels****Syntax** void Dimension.setLevels(xtabanalyzer.Level[ ] levels)

Sets levels for the dimension.

**Parameter** **levels**

Array of xtabanalyzer.Level objects representing the levels for the dimension.

**Example** This example sets the dimension levels:

```
function setDimensionLevels(dimension, levels) {
 if (dimension && levels) {
 dimension.setLevels(levels);
 }
}
```

**setNewAxisType****Syntax** void Dimension.setNewAxisType(integer newAxisType)

Sets the new axis type.

**Parameter** **newAxisType**

Integer. The new axis type.

**Example** This example retrieves and changes the axis type:

```
function swapAxis(dimension) {
 if (dimension.getAxisType() ==
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE) {
 dimension.setNewAxisType(
 actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
 } else {
 dimension.setNewAxisType(
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
 }
}
```

## setNewIndex

**Syntax** void Dimension.setNewIndex(integer newIndex)

Sets the new index.

**Parameter** **newIndex**

Integer. The new index.

**Example** This example retrieves and increments the index:

```
function incrementIndex(dimension) {
 var newIndex = dimension.getIndex() + 1;
 dimension.setNewIndex(newIndex);
}
```

---

## Class actuate.xtabalyzer.Driller

**Description** The Driller class enables an application to drill down or up levels on a member within a dimension.

### Constructor

**Syntax** `actuate.xtabalyzer.Driller()`

Creates a Driller object.

### Function summary

Table 8-5 lists actuate.xtabalyzer.Driller functions.

**Table 8-5** actuate.xtabalyzer.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

### addMember

**Syntax** `void Driller.addMember(actuate.xtabalyzer.MemberValue member)`

Adds a dimension member to the drill condition. Functional candidates are Dimension members with levels.

**Parameter** **member**  
actuate.xtabalyzer.MemberValue object. A member value to add.

**Example** This example adds a member to a Driller object:

```
function drillDownDimension(){
 var driller = new actuate.xtabalyzer.Driller();
 driller.setDimension(actuate.xtabalyzer.Dimension
 .ROW_AXIS_TYPE);
 var memberValue = new actuate.xtabalyzer
 .MemberValue("drillLevelName");
 memberValue.setValue("drillLevelValue");
 driller.addMember(memberValue);
 crosstab.drill(driller);
```

```
actuate.xtabanalyzer.Driller
```

```
 crosstab.submit();
}
```

## getDimension

**Syntax** string Driller.getDimension( )

Returns the dimension name for the drill condition.

**Returns** String. A dimension name.

**Example** This example retrieves the dimension of the driller:

```
function getDrillerAxis(driller) {
 if (driller) {
 return driller.getDimension();
 }
 return null;
}
```

## getMembers

**Syntax** actuate.xtabanalyzer.MemberValue[ ] Driller.getMembers( )

Returns the list of members assigned to the driller.

**Returns** Array of actuate.xtabanalyzer.MemberValue. A dimension member.

**Example** This example retrieves the members that a driller uses:

```
function getDrillerMembers(driller) {
 if (driller) {
 return driller.getMembers();
 }
 return null;
}
```

## setDimension

**Syntax** void Driller.setDimension(string dimension)

Sets the dimension for the driller by name.

**Parameter** dimension

String. A dimension name.

**Example** This example sets the dimension name for the driller:

```
function setRowAxis(driller) {
 if (driller) {
 dimension.setDimension("Row");
 }
}
```

```
 }
}
```

## setMembers

**Syntax** void Driller.setMembers(actuate.xtabanalyzer.MemberValue[ ] member)

Sets an array of members to the drill condition.

**Parameter** **member**

Array of actuate.xtabanalyzer.MemberValue objects. An array of members.

**Example** This example sets the axis type for the driller:

```
function setDrillerMembers(driller, members) {
 if (driller && members) {
 driller.setMembers(members);
 }
}
```

---

## Class actuate.xtabanalyzer.EventConstants

**Description** Defines constants for xtabanalyzer events. Table 8-6 lists the cross tab analyzer event constants.

**Table 8-6** actuate.xtabanalyzer.Dimension constants

Constant	Description
ON_CONTENT_CHANGED	Content changed event. Triggers when the displayed content has changed, for example when changing cross tab report content. The event handler takes an actuate.XTabAnalyzer object that represents the viewer for which the event occurred, as the only parameter.
ON_CONTENT_SELECTED	Content selected event. Triggers when a user clicks on report elements. The event handler takes the following parameters: <ul style="list-style-type: none"> <li>■ actuate.XTabAnalyzer: object viewer for which event occurred</li> <li>■ actuate.xtabanalyzer.SelectedContent: the SelectedContent object</li> </ul>
ON_EXCEPTION	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>■ actuate.XTabAnalyzer: viewer for which the event occurred</li> <li>■ actuate.Exception: Exception object</li> </ul>
ON_SESSION_TIMEOUT	Session time-out event. When a session time-out event occurs and the user tries to perform any operation on a viewer, a prompt dialog appears asking the user whether or not to log in again. When the user chooses to log in again, the ON_SESSION_TIMEOUT event triggers. When no handler is registered for this event, a default built-in login dialog will be displayed. <p>The event handler takes one parameter: an actuate.XTabAnalyzer object, representing the viewer where the event occurred.</p>

---

## 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 8-7 lists actuate.xtabanalyzer.Exception functions.

**Table 8-7** actuate.xtabanalyzer.Exception functions

Function	Description
getDescription( )	Returns details of the exception
getElement( )	Returns the report element for which the exception occurred, if available
getErrCode( )	Returns the error code for ERR_SERVER
getMessage( )	Returns a short message about the error
getType( )	Returns the type of error exception
isExceptionType( )	Returns Boolean indicating whether exception is of certain type

### getDescription

**Syntax** string Exception.getDescription( )

Returns exception details as provided by the Server, Client, and User objects.

**Returns** String. A detailed description of the error. Information is provided according to the type of exception generated, as shown below:

- ERR\_SERVER: The SOAP string

## actuate.xtabanalyzer.Exception

- ERR\_CLIENT: For the Firefox browser, a list comprised of fileName+number+stack
- ERR\_USAGE: Any value set when the object was created

**Example** This example consists of a function that registerEventHandler( ) set as a callback. The callback function takes an instance of the Exception class. Each of the functions for the Exception class can be called with the results formatted to create a message or for some other use.

```
function errorHandler(viewerInstance, exception) {
 alert(exception.getDescription());
}
```

## getElement

**Syntax** string Exception.getElement( )

Returns the report element for which the exception occurred, if available.

**Returns** String. The report element for which the exception occurred.

**Example** This example uses getElement( ):

```
function errorHandler(viewerInstance, exception) {
 alert("Error in " + exception.getElement());
}
```

## getErrCode

**Syntax** string Exception.getErrCode( )

Returns the error code for ERR\_SERVER.

**Returns** String. The error code for ERR\_SERVER.

**Example** This example uses getErrCode( ):

```
function errorHandler(viewerInstance, exception) {
 alert(exception.getErrCode());
}
```

## getMessage

**Syntax** string Exception.getMessage( )

Returns a short message about the error.

**Returns** String. A short message about the exception.

**Example** This example uses getMessage( ):

```
function errorHandler(viewerInstance, exception) {
```

```

 alert(exception.getMessage());
 }
}
```

## getType

**Syntax** string Exception.getType( )

Returns the type of exception error.

**Returns** String. The errType exception type.

**Example** This example uses getType():

```

function errorHandler(viewerInstance, exception) {
 alert(exception.getType());
}
```

## isExceptionType

**Syntax** boolean Exception.isExceptionType(object exceptionType)

Checks an exception's type for a match against a specified type.

**Parameter** exceptionType

An exception type as string, or exception class. For example, "actuate.viewer.ViewerException" or actuate.viewer.ViewerException.

**Returns** True if the exception is of the stated type, false otherwise.

**Example** This example checks to see if the exception is a client error type:

```

function errorHandler(viewerInstance, exception) {
 if (exception.isExceptionType(ERR_CLIENT)) {
 alert("CLIENT ERROR");
 }
}
```

---

## Class actuate.xtabanalyzer.Filter

**Description** The Filter class creates a filter condition on a cross tab dimension level. The condition is expressed as value1 operator value2. The values can either be a single value, or an array of values, depending on the operator. For example, IN can be expressed as value1 IN value2 value3 ... valueN.

### Constructor

**Syntax**

```
actuate.xtabanalyzer.Filter(string levelName, string levelAttributeName, string
operator, string value, string filterType)

actuate.xtabanalyzer.Filter(string levelName, string levelAttributeName, string
operator, string value1, string value2, string filterType)

actuate.xtabanalyzer.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 8-8 lists the valid filter operators and the number of arguments to pass to the constructor or setValues( ).

**Table 8-8** 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 8-8** 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.

## actuate.xtabanalyzer.Filter

### filterType

String. The filter type.

## Function summary

Table 8-9 lists actuate.xtabanalyzer.Filter functions.

**Table 8-9** actuate.xtabanalyzer.Filter functions

Function	Description
getFilterType()	Returns the filter type
getLevelAttributeName( )	Returns the dimension level attribute name
getLevelName( )	Returns the name of the filtered level
getOperator( )	Returns the filter operator
getValues( )	Returns the set of values the filter is using
setFilterType()	Sets the filter type
setLevelAttributeName( )	Sets the dimension level attribute name
setLevelName( )	Sets the dimension level name
setOperator()	Sets the filter operator
setValues( )	Sets the values for the filter

### getFilterType

**Syntax** string Filter.getFilterType( )

Returns the filter type.

**Returns** String. The filter type.

**Example** This example retrieves the filter type for a filter:

```
function getType(filter){
 if(filter){
 return filter.getFilterType();
 }else{
 return null;
 }
}
```

### getLevelAttributeName

**Syntax** string Filter.getLevelAttributeName( )

Returns the name of the dimension level attribute to which this filter applies.

**Returns** String. The level attribute name.

**Example** This example retrieves the filter level attribute name for a filter:

```
function getLevelAttribute(filter){
 if(filter){
 return filter.getLevelAttributeName();
 }else{
 return null;
 }
}
```

## getLevelName

**Syntax** string Filter.getLevelName( )

Returns the name of the dimension level to which this filter applies.

**Returns** String. A level name.

**Example** This example retrieves the filter level name for a filter:

```
function getLevel(filter){
 if(filter){
 return filter.getLevelName();
 }else{
 return null;
 }
}
```

## getOperator

**Syntax** string Filter.getOperator( )

Returns the filter operator.

**Returns** String. The filter operator.

**Example** This example retrieves the filter operator:

```
function getFilterOp(filter){
 if(filter){
 return filter.getOperator();
 }else{
 return null;
 }
}
```

## getValues

**Syntax** string[ ] Filter.getValues( )

## actuate.xtabanalyzer.Filter

Returns an array containing the values used in the filter.

**Returns** Array of strings. The values for the filter.

**Example** This example retrieves the filter level name for a filter:

```
function getFilterOp(filter){
 if(filter){
 return filter.getValues();
 }else{
 return null;
 }
}
```

## setFilterType

**Syntax** void Filter.setFilterType(string filterType)

Sets the filter type to filter.

**Parameter** **filterType**

String. The type of filter.

**Example** This example sets the filter type to equality:

```
function filterLevel(){
 var filterType = "equality";
 var filter = new actuate.xtabanalyzer.Filter("levelName",
 "attributeName",actuate.xtabanalyzer.Filter.EQ,
 "2000","blank");
 filter.setFilterType(filterType);
 crosstab.setFilters(filter);
 crosstab.submit();
}
```

## setLevelAttributeName

**Syntax** void Filter.setLevelAttributeName(string levelAttributeName)

Sets the dimension level attribute to filter on by name.

**Parameter** **levelAttributeName**

String. The name of the level attribute to filter.

**Example** This example sets the level attribute name to attributeName:

```
function filterLevel(){
 var attributeName = "attributeName";
 var filter = new actuate.xtabanalyzer.Filter("levelName",
 "blank",actuate.xtabanalyzer.Filter.EQ,
 "2000","equality");
 filter.setLevelAttributeName(attributeName);
```

```

 crosstab.setFilters(filter);
 crosstab.submit();
 }
}

```

## setLevelName

**Syntax** void Filter.setLevelName(string level)

Sets the level to filter by name.

**Parameter** **level**

String. The name of the level to filter.

**Example** This example sets the filter level name to levelName:

```

function filterLevel(){
 var levelName = "levelName";
 var filter = new actuate.xtabanalyzer.Filter("blank",
 "attributeName",actuate.xtabanalyzer.Filter.EQ,
 "2000","equality");
 filter.setLevelName(levelName);
 crosstab.setFilters(filter);
 crosstab.submit();
}

```

## setOperator

**Syntax** void Filter.setOperator(string operator)

Sets the filter operator.

**Parameter** **operator**

String. The filter operator.

**Example** This example sets the filter operator to EQ:

```

function filterLevel(){
 var operator = "EQ";
 var filter = new actuate.xtabanalyzer.Filter("levelName",
 "attributeName",actuate.xtabanalyzer.Filter.NOT,
 "2000","equality");
 filter.setOperator(operator);
 crosstab.setFilters(filter);
 crosstab.submit();
}

```

## setValues

**Syntax** void Filter.setValues(string[ ] value1, string[ ] value2)

Sets the values for the filter.

## actuate.xtabanalyzer.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",
 actuate.xtabanalyzer.Filter.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 8-10 lists actuate.xtabanalyzer.GrandTotal functions.

**Table 8-10** 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

### addTotal

**Syntax** `void GrandTotal.addTotal(object total)`

Adds a total to the cross tab.

**Parameter** **total**

`actuate.xtabanalyzer.total`. The total to add to the cross tab.

**Example** This example adds totals to a grand total:

```
function addTotal(grandTotal) {
// The indexStr can be set from a web page or other source as
// necessary.
 var indexStr = "0;1;2;3;4";
 var indexs = indexsStr.split(";");
 var count = indexs.length;
 var measureIndexs = [];
 for(var i = 0;i < count;i++) {
 measureIndexs.push(parseInt(indexs[i]));
 }
 for(var i = 0; i < measureIndexs.length; i++) {
```

## actuate.xtabanalyzer.GrandTotal

```
 var total = new actuate.xtabanalyzer.Total();
 total.setMeasureIndex(measureIndexs[i]);
 total.setAggregationFunction("SUM");
 total.setEnabled(true);
 grandTotal.addTotal(total);
 }
}
```

### getAxisType

**Syntax** integer GrandTotal.getAxisType( )

Returns the axis type for the total.

**Returns** Integer. The following values are legal axis types:

- actuate.xtabanalyzer.Dimension.COLUMN\_AXIS\_TYPE
- actuate.xtabanalyzer.Dimension.ROW\_AXIS\_TYPE

**Example** This example retrieves and sets the axis type:

```
function swapAxis(grandtotal) {
 if (grandtotal.getAxisType() ==
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE) {
 grandtotal.setNewAxisType(
 actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
 } else {
 grandtotal.setNewAxisType(
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
 }
}
```

### getTotals

**Syntax** object[ ] GrandTotal.getTotals( )

Returns an array containing the totals.

**Returns** Array of Total objects. The totals.

**Example** This example retrieves totals from a GrandTotal object:

```
var totalsArray = [];
function getTotals(grandTotal,totalsArray){
 totalsArray = grandTotal.getTotals();
}
```

### getType

**Syntax** string GrandTotal.getType( )

Returns the type for the total.

**Returns** String. The total type.

## setAxisType

**Syntax** void GrandTotal.setAxisType(integer axisType)

Sets the axis type for the total.

**Parameter**

**axisType**

Integer. Axis type for the total.

**Example** This example retrieves and sets the axis type:

```
function swapAxis(grandtotal) {
 if (grandtotal.getAxisType() ==
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE) {
 grandtotal.setNewAxisType(
 actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
 } else {
 grandtotal.setNewAxisType(
 actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
 }
}
```

## setTotals

**Syntax** void GrandTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)

Sets totals as an array.

**Parameter**

**totals**

Array of actuate.xtabanalyzer.Total objects to add to the grand total.

**Example** This example copies the totals from grandtotal1 into grandtotal2:

```
grandtotal2.setTotals(grandtotal1.getTotals());
```

---

## Class actuate.xtabanalyzer.Level

**Description** Defines a cross tab dimension level, its controls, and content.

### Constructor

**Syntax** actuate.xtabanalyzer.Level( )

Creates a cross tab Level object.

### Function summary

Table 8-11 lists actuate.xtabanalyzer.Level functions.

**Table 8-11** actuate.xtabanalyzer.Level functions

Function	Description
addAttribute()	Adds the level attribute
getAttributes()	Returns the level attributes
getIndex()	Returns the index of the level
getLevelName()	Returns the level name
setIndex()	Sets the index level
setLevelName()	Sets the level name

### addAttribute

**Syntax** void Level.addAttribute(actuate.xtabanalyzer.LevelAttribute attr)

Adds the level attribute.

**Parameter** **index**

actuate.xtabanalyzer.LevelAttribute object. A level attribute.

**Example** This example sets a name for newly created level attribute and assigns the attribute to a level:

```
var attribute = new actuate.xtabanalyzer.LevelAttribute();
attribute.setName("pounds");
level.addAttribute(attribute);
```

### getAttributes

**Syntax** actuate.xtabanalyzer.LevelAttribute[ ] Level.getAttributes( )

Returns the level attributes.

**Returns** Array of actuate.xtabanalyzer.LevelAttribute objects. The level attributes.

**Example** This example retrieves the level index and stores it in a variable called lattributes:

```
var lattributes = new actuate.xtabanalyzer.LevelAttribute[];
lattributes = level.getAttributes();
```

## getIndex

**Syntax** integer Level.getIndex( )

Returns the level index.

**Returns** Integer. The level index.

**Example** This example retrieves the level index:

```
function levelIndex(level){
 if (level){
 return level.getIndex();
 }
 return null;
}
```

## getLevelName

**Syntax** string Level.getLevelName( )

Returns the level name.

**Returns** String. The level name.

**Example** This example retrieves the level name:

```
function levelName(level){
 if (level){
 return level.getLevelName();
 }
 return null;
}
```

## setIndex

**Syntax** void Level.setIndex(integer index)

Sets the level index.

**Parameter** **index**

Integer. The level index.

**Example** This example sets the level index:

```
function assignIndex(level, index){
```

```
actuate.xtabanalyzer.Level
```

```
 if (level) {
 return level.setIndex(index);
 }
}
```

## setLevelName

**Syntax** void Level.setLevelName(string levelName)

Sets the level name.

**Parameter** **levelName**

String. The level name.

**Example** This example sets level names for newly created levels:

```
var levelNames = "year;month;day";
...
function addLevels(dimension,levelNames){
 var levelNamesArray = levelNames.split(",");
 for(var i = 0; i < levelNames.length; i++){
 var level = new actuate.xtabanalyzer.Level();
 level.setLevelName(levelNames[i]);
 dimension.addLevel(level);
 }
}
```

---

## Class actuate.xtabanalyzer.LevelAttribute

**Description** Defines an attribute for a level.

### Constructor

**Syntax** `actuate.xtabanalyzer.LevelAttribute( )`

Creates a cross tab level attribute object.

### Function summary

Table 8-12 lists `actuate.xtabanalyzer.LevelAttribute` functions.

**Table 8-12** `actuate.xtabanalyzer.Level` functions

Function	Description
<code>getName()</code>	Returns the level attribute name
<code>setName( )</code>	Sets the level attribute name

### getName

**Syntax** `string LevelAttribute.getName( )`

Returns the level attribute name.

**Returns** String. A level attribute name.

**Example** This example retrieves the level attribute name and stores it in a variable `attname`:

```
var attname = levelattribute.getName();
```

### setName

**Syntax** `void LevelAttribute.setName(string attributeName)`

Sets the level attribute name.

**Parameter** **attributeName**

String. The level attribute name.

**Example** This example sets a name for newly created level attribute and assigns the attribute to a level:

```
var attribute = new actuate.xtabanalyzer.LevelAttribute();
attribute.setName("pounds");
level.addLevelAttribute(attribute);
```

---

## Class actuate.xtabanalyzer.Measure

**Description** Defines a cross tab measure.

### Constructor

**Syntax** actuate.xtabanalyzer.Measure( )

Creates a cross tab measure object.

### Function summary

Table 8-13 lists actuate.xtabanalyzer.Measure functions.

**Table 8-13** actuate.xtabanalyzer.Measure functions

Function	Description
getAggregationFunction( )	Returns the aggregation function name
getDataType( )	Returns the computed column data type
getExpression( )	Returns the computed measure expression
getIndex( )	Returns the measure index
getMeasureName( )	Returns the measure name
getNewIndex( )	Returns the new index
setAggregationFunction( )	Sets the aggregation function name
setDataType( )	Sets the computed column data type
setExpression( )	Sets the computed measure expression
setIndex( )	Sets the measure index
setMeasureName( )	Sets the measure name
setNewIndex( )	Sets the new index

### getAggregationFunction

**Syntax** string Measure.getAggregationFunction( )

Returns the aggregation function name.

**Returns** String. An aggregation function name.

**Example** This example changes the aggregation function:

```
function swapMeasureAggregation(measure) {
 if (measure.getAggregation() == "EQ") {
 measure.setAggregation("NE");
 }
}
```

```

 }else{
 measure.setAggregation("EQ");
 }
 }
}

```

## getDataType

**Syntax** string Measure.getDataType( )

Returns the computed column data type.

**Returns** String. The data type.

**Example** This example retrieves the computed column data type:

```

function getColumnDataType(measure) {
 if (measure) {
 return measure.getDataType();
 }
 return null;
}

```

## getExpression

**Syntax** string Measure.getExpression( )

Returns the computed measure expression.

**Returns** String. An expression.

**Example** This example retrieves the computed measure expression:

```

function getMeasureExpression(measure) {
 if (measure) {
 return measure.getExpression();
 }
 return null;
}

```

## getIndex

**Syntax** integer Measure.getIndex( )

Returns the measure index.

**Returns** Integer. The measure index.

**Example** This example retrieves the measure index:

```

function getMeasureIndex(measure) {
 if (measure) {
 return measure.getIndex();
 }
}

```

```
actuate.xtabanalyzer.Measure
```

```
 }
 return null;
}
```

## getMeasureName

**Syntax** string Measure.getMeasureName( )

Returns the measure name.

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

**Example** This example retrieves the measure name:

```
function getMeasureName(measure) {
 if (measure) {
 return measure.getMeasureName();
 }
 return null;
}
```

## getNewIndex

**Syntax** integer Measure.getNewIndex( )

Retrieves the new index. The new index is set by setNewIndex and represents the index value the measure has after submit( ) finishes executing.

**Returns** Integer. The new index.

**Example** This example retrieves the new measure index:

```
function getNewMeasureIndex(measure) {
 if (measure) {
 return measure.getNewIndex();
 }
 return null;
}
```

## setAggregationFunction

**Syntax** void Measure.setAggregationFunction(string aggregationFunction)

Sets the aggregation function name.

**Parameter** **aggregationFunction**

String. The aggregation function name.

**Example** This example changes the aggregation function:

```
function swapMeasureAggregation(measure) {
 if (measure.getAggregation() == "EQ") {
```

```

 measure.setAggregation("NE");
 }else{
 measure.setAggregation("EQ");
 }
}
}

```

## setDataType

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

Sets the computed column data type name.

**Parameter** **dataType**

String. The data type.

## setExpression

**Syntax** void Measure.setExpression(string expression)

Sets the computed measure expression.

**Parameter** **expression**

String. The computed measure expression.

**Example** This example uses setExpression:

```

function addMeasure(viewer) {
 var crosstab = getCrosstab(viewer);
 if(crosstab) {
 var measureName = "measureName";
 var measureExpression =
 "[revenue]/[revenue_SalesDate/year_Product/PRODUCTLINE]";

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

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

```

## setIndex

**Syntax** void Measure.setIndex(integer index)

Sets the index.

## actuate.xtabanalyzer.Measure

### Parameter index

Integer. The index of this measure.

### Example

This example uses setIndex to add a new measure to a cross tab:

```
function setIndex(measure, index) {
 measure.setIndex(index);
}
```

## setMeasureName

### Syntax

```
void Measure.setMeasureName(string measureName)
```

Sets the measure name.

### Parameter measureName

String. The measureName.

### Example

This example sets the measure name which is taken from a page element:

```
function renameMeasure(measure) {
 var measureName = document.getElementById("measureName").value;
 measure.setMeasureName(measureName);
}
```

## setNewIndex

### Syntax

```
void Measure.setNewIndex(integer newIndex)
```

Sets a new measure index.

### Parameter newIndex

Integer. The new measure index.

### Example

This example changes the index for the measure:

```
function changeIndex(measure, index) {
 if (measure) {
 measure.setNewIndex(index);
 }
}
```

---

## Class actuate.xtabanalyzer.MemberValue

**Description** Defines a member value used for sort, filter, or drill functionality.

### Constructor

**Syntax** `actuate.xtabanalyzer.MemberValue(levelName, value, (MemberValue))`

Creates a MemberValue object for a given level and value. The object can contain multiple member values.

**Parameters** **levelName**

String. Dimension level name of member.

**value**

String. Value for the member to contain.

**MemberValue**

Optional `actuate.xtabanalyzer.MemberValue` object. MemberValue object to add during construction.

### Function summary

Table 8-14 lists `actuate.xtabanalyzer.MemberValue` functions.

**Table 8-14** `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

### addMember

**Syntax** `void MemberValue.addMember(actuate.xtabanalyzer.MemberValue member)`

Adds a member value.

**Parameter** **member**

`actuate.xtabanalyzer.MemberValue` object. A member value.

**Example** `MemberValue` is an embedded class that can be a single value or an array of values. This example has a single member that contains four members:

## actuate.xtabanalyzer.MemberValue

```
function addMembers(memberData) {
 var mv1 = new MemberValue('dim/state', 'CA');
 var mv2 = new MemberValue('dim/state', 'CN');
 var mv3 = new MemberValue(memberData);
 var mv = new MemberValue('dim/country', 'USA');
 mv.addMember(mv1);
 mv.addMember(mv2);
 mv.addMember(mv3);
 return mv;
}
```

## getLevelName

**Syntax** string MemberValue.getLevelName( )

Returns the level name of the member.

**Returns** String. The level name.

**Example** This example retrieves the level name for the member value:

```
function getLevelName(level) {
 if (level) {
 return level.getLevelName();
 }
 return null;
}
```

## getMembers

**Syntax** actuate.xtabanalyzer.MemberValue[ ] MemberValue.getMembers( )

Returns all the member value objects contained in this member value object.

**Returns** Array of actuate.xtabanalyzer.MemberValue. An array of MemberValue objects.

**Example** This example returns the number of members in a member object:

```
function getMemberCount(members) {
 if (members) {
 var membersArray[] = members.getMembers();
 return membersArray.length;
 }
 return null;
}
```

## getValue

**Syntax** string MemberValue.getValue( )

Returns the level value.

**Returns** String. The level value.

**Example** This example returns the value for the level:

```
function getMemberValue(members) {
 if (members) {
 return members.getValue();
 }
 return null;
}
```

## setLevelName

**Syntax** void MemberValue.setLevelName(string level)

Sets the level name.

**Parameter** **level**

String. The name of the level.

**Example** This example sets the level name:

```
function getMemberValue(members) {
 if (members) {
 return members.getValue();
 }
 return null;
}
```

## setValue

**Syntax** void MemberValue.setValue(string level)

Sets the level value.

**Parameter** **level**

String. The value for the level.

**Example** This example sets the level value:

```
function setMemberLevelValue(member, lvlValue) {
 if (member) {
 member.setValue(lvlValue);
 }
}
```

## Class actuate.xtabanalyzer.Options

**Description** The Options class specifies options for the cross tab.

### Constructor

**Syntax** `actuate.xtabanalyzer.Options(string measureDirection, string rowMirrorStartingLevel, string columnMirrorStartingLevel, string emptyCellValue, boolean enablePageBreak, integer rowPageBreakInterval, integer columnPageBreakInterval)`

Creates an options object that contains options for how the cross tab displays data.

**Parameters**

**measureDirection**

String. The measure direction. Legal values for measure direction are:

- DIRECTION\_HORIZONTAL
- DIRECTION\_VERTICAL

**rowMirrorStartingLevel**

String. Row mirror starting level name.

**columnMirrorStartingLevel**

String. Column mirror starting level name.

**emptyCellValue**

String. Value to display for an empty cell.

**enablePageBreak**

Boolean. Enables page breaks when true.

**rowPageBreakInterval**

Integer. Row page break interval.

**columnPageBreakInterval**

Integer. Column page break interval.

**grandTotalsDisplayOption**

String. Grand totals display option. Legal values for total display options are:

- DIRECTION\_HORIZONTAL
- DIRECTION\_VERTICAL

**subtotalsDisplayOption**

String. Subtotals display option. Legal values for total display options are:

- DIRECTION\_HORIZONTAL
- DIRECTION\_VERTICAL

## Function summary

Table 8-15 lists actuate.xtabanalyzer.Options functions.

**Table 8-15** actuate.xtabanalyzer.Options functions

Function	Description
getColumnMirrorStartingLevel( )	Returns the column mirror starting level full name
getColumnPageBreakInterval( )	Returns the column page break interval
getEmptyCellValue( )	Returns the empty cell value
getEnablePageBreak( )	Returns the page break enabled or disabled status
getMeasureDirection( )	Returns the measure direction
getRowMirrorStartingLevel( )	Returns the row mirror starting level full name
getRowPageBreakInterval( )	Returns the row page break interval
setColumnMirrorStartingLevel( )	Sets the column mirror starting level full name
setColumnPageBreakInterval( )	Sets the column page break interval
setEmptyCellValue( )	Sets the empty cell value
setEnablePageBreak( )	Sets the flag to enable page breaks
setMeasureDirection( )	Sets the measure direction
setRowMirrorStartingLevel( )	Sets the row mirror starting level full name
setRowPageBreakInterval( )	Sets the row page break interval

### getColumnMirrorStartingLevel

**Syntax** string Options.getColumnMirrorStartingLevel( )

Returns the column mirror starting level name.

**Returns** String. Column mirror starting level name.

**Example** This example retrieves the column mirror starting level:

```
function getColumnMirrorStart(options) {
 if (options) {
 return options.getColumnMirrorStartinglevel();
 }
 return null;
}
```

## getColumnPageBreakInterval

**Syntax** integer Options.getColumnPageBreakInterval( )

Returns the column page break interval.

**Returns** Integer. The column page break interval.

**Example** This example retrieves the column page break interval:

```
function getColumnPBIInterval(options) {
 if (options) {
 return options.getColumnPageBreakInterval();
 }
 return null;
}
```

## getEmptyCellValue

**Syntax** string Options.getEmptyCellValue( )

Returns the empty cell value.

**Returns** String. Value to display for an empty cell.

**Example** This example retrieves the empty cell:

```
function getEmptyCell(options) {
 if (options) {
 return options.getEmptyCellValue();
 }
 return null;
}
```

## getEnablePageBreak

**Syntax** boolean Options.getEnablePageBreak( )

Returns the page break status.

**Returns** Boolean. Page breaks are enabled when the value is true.

**Example** This example retrieves the column page break interval when page breaks are enabled:

```
function getColumnPBEEnabled(options) {
 if (options.getEnablePageBreak())
 return options.getColumnPageBreakInterval();
 else
 alert ("Page Breaks Not Enabled.");
 return null;
}
```

## getMeasureDirection

**Syntax** string Options.getMeasureDirection( )

Returns the measure direction.

**Returns** String. The measure direction. Legal values for measure direction are:

- DIRECTION\_HORIZONTAL
- DIRECTION\_VERTICAL

**Example** This example retrieves the measure direction:

```
function getMeasureDirection(options) {
 if (options) {
 return options.getMeasureDirection();
 }
 return null;
}
```

## getRowMirrorStartingLevel

**Syntax** string Options.getRowMirrorStartingLevel( )

Returns the row mirror starting level name.

**Returns** String. Row mirror starting level name.

**Example** This example retrieves the row mirror starting level:

```
function getRowMirrorStart(options) {
 if (options) {
 return options.getRowMirrorStartinglevel();
 }
 return null;
}
```

## getRowPageBreakInterval

**Syntax** integer Options.getRowPageBreakInterval( )

Returns the row page break interval.

**Returns** Integer. The row page break interval.

**Example** This example retrieves the row page break interval:

```
function getRowPBIInterval(options) {
 if (options) {
 return options.getRowPageBreakInterval();
 }
}
```

```
actuate.xtabanalyzer.Options
```

```
 return null;
}
```

## setColumnMirrorStartingLevel

**Syntax** void Options.setColumnMirrorStartingLevel(string levelName)

Sets the column mirror starting level name.

**Parameter** **levelName**

String. The column mirror starting level name.

**Example** This example sets the column mirror starting level:

```
function setColumnMirrorLevel(options, level) {
 if (options) {
 options.setColumnMirrorStartingLevel(level);
 }
}
```

## setColumnPageBreakInterval

**Syntax** void Options.setColumnPageBreakInterval(integer columnPageBreakInterval)

Sets the column page break interval.

**Parameter** **columnPageBreakInterval**

Integer. The column page break interval.

**Example** This example sets the column page break interval:

```
function setColumnPBInterval(options, interval) {
 if (options) {
 options.setColumnPageBreakInterval(interval);
 }
}
```

## setEmptyCellValue

**Syntax** void Options.setEmptyCellValue(string emptyCellValue)

Sets the empty cell value.

**Parameter** **emptyCellValue**

String. The empty cell value.

**Example** This example sets the empty cell value:

```
function setEmptyCell(options, cellValue) {
 if (options) {
 options.setEmptyCellValue(cellValue);
 }
}
```

```

 }
 }
}
```

## setEnablePageBreak

**Syntax** void Options.setEnablePageBreak(boolean enablePageBreak)

Enables or disables page breaks.

**Parameter** **enablePageBreak**

Boolean. Enables page breaks when true.

**Example** This example enables page breaks and sets the row page break interval:

```

function enablesetRowPBIInterval(options,interval) {
 if (options){
 options.setEnablePageBreak(true);
 options.setRowPageBreakInterval(interval);
 }
}
```

## setMeasureDirection

**Syntax** void Options.setMeasureDirection(string measureDirection)

Sets the measure direction.

**Parameter** **measureDirection**

String. The measure direction. The measure direction. Legal values for measure direction are:

- DIRECTION\_HORIZONTAL
- DIRECTION\_VERTICAL

**Example** This example sets the measure direction:

```

function setMeasureDirection(options,direction) {
 if (options){
 options.setMeasureDirection(direction);
 }
}
```

## setRowMirrorStartingLevel

**Syntax** void Options.setRowMirrorStartingLevel(string levelName)

Sets the row mirror starting level.

**Parameter** **levelName**

String. Row mirror starting level name.

## actuate.xtabanalyzer.Options

**Example** This example sets the row mirror starting level:

```
function setRowMirrorLevel(options,level) {
 if (options) {
 options.setRowMirrorStartingLevel(level);
 }
}
```

## setRowPageBreakInterval

**Syntax** void Options.setRowPageBreakInterval(integer rowPageBreakInterval)

Sets the row page break interval.

**Parameter** **rowPageBreakInterval**

Integer. The row page break interval.

**Example** This example sets the row page break interval:

```
function setRowPBInterval(options,interval) {
 if (options) {
 options.setRowPageBreakInterval(interval);
 }
}
```

---

## Class actuate.xtabanalyzer.PageContent

**Description** A container for the content of a cross tab page. It contains a comprehensive list of report elements, such as tables, charts, labels, and data items.

### Constructor

**Syntax** `actuate.xtabanalyzer.PageContent()`

Creates a PageContent object that represents the report content that is generated by a report design file or document file.

### Function summary

Table 8-16 lists `actuate.xtabanalyzer.PageContent` functions.

**Table 8-16** `actuate.xtabanalyzer.PageContent` functions

Function	Description
<code>getCrossstabByBookmark()</code>	Returns a report cross tab object
<code>getViewerId()</code>	Returns the cross tab viewer ID

### getCrossstabByBookmark

**Syntax** `actuate.xtabanalyzer.crosstab PageContent.getCrossstabByBookmark(string bookmark)`

Returns a cross tab object associated with a bookmark.

**Parameter** **bookmark**  
The bookmark name of the item requested.

**Returns** `actuate.xtabanalyzer.crosstab` object.

**Example** This example retrieves the viewer ID, then retrieves the cross tab:

```
function getCrosstab() {
 var viewer = PageContent.getViewerId();
 var content = viewer.getCurrentPageContent();
 var crosstab = content.getCrossstabByBookmark();
 return crosstab;
}
```

### getViewerId

**Syntax** `string PageContent.getViewerId()`

Returns the XTabAnalyzer ID. The XTabAnalyzer is the cross tab viewer element.

## actuate.xtabanalyzer.PageContent

**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 valuesNull)`

The ParameterValue class is used to specify a cross tab ParameterValue object.

**Parameters**

- name**  
String. The parameter name.

- value**  
String. The parameter value.

- valuesNull**  
Boolean. Whether the value is null.

### Function summary

Table 8-17 lists actuate.xtabanalyzer.ParameterValue functions.

**Table 8-17** 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>setValue()</code>	Sets the parameter value
<code>setValueIsNull()</code>	Sets whether the parameter has a null value

### getName

**Syntax** `string ParameterValue.getName( )`

Returns the name for the parameter.

**Returns** String. The parameter name.

**Example** This example retrieves the parameter name:

```
function getParameterName(parametervalue) {
 if (parametervalue) {
 return parametervalue.getName();
 }
}
```

```
actuate.xtabanalyzer.ParameterValue
```

```
 }
 return null;
 }
```

## getValue

**Syntax** String[ ] Dimension.getValue( )

Returns the name for the ParameterValue.

**Returns** String or array of strings. The parameter value or values.

**Example** This example retrieves the parameter value:

```
function getParameterValue(parametervalue) {
 if (parametervalue) {
 return parametervalue.getValue();
 }
 return null;
}
```

## getValuesNull

**Syntax** boolean ParameterValue.getValuesNull( )

Returns whether the parameter value is null.

**Returns** Boolean. True indicates the parameter value is null.

**Example** This example switches whether the parameter value is null:

```
if (parametervalue) {
 if (parametervalue.getValueIsNull) {
 parametervalue.setValueIsNull(false);
 } else {
 parametervalue.setValueIsNull(true);
 }
}
```

## setName

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

Sets the parameter name.

**Parameter** name

String. The parameter name.

**Example** This example sets the parameter name:

```
function setParameterName(parametervalue, name) {
 parametervalue.setName(name);
}
```

## setValue

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

Sets the parameter value.

**Parameter** **value**

String. The parameter value.

**Example** This example sets the parameter value:

```
function setParameterValue(parametervalue, value) {
 parametervalue.setValue(value);
}
```

## setValuesIsNull

**Syntax** void ParameterValue.setValuesIsNull(boolean valuesIsNull)

Sets the valuesIsNull for the ParameterValue.

**Parameter** **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 8-18 lists actuate.xtabanalyzer.Sorter functions.

**Table 8-18** actuate.xtabanalyzer.Sorter functions

Function	Description
getKey()	Returns the sort key
getLevelName()	Returns the level name
getMember()	Returns the sort member
isAscending()	Returns the sort direction
setAscending()	Sets ascending or descending sort
setKey()	Sets the sort key
setLevelName()	Sets the level name
setMember()	Sets the sort member

### getKey

**Syntax** string Sorter.getKey( )

Returns the sort key. This is the name of the measure or dimension level to sort the cross tab on.

**Returns** String. The key to sort on.

**Example** This example retrieves the sort key:

```
function getSortKey(sorter) {
 if (sorter) {
 return sorter.getKey();
 }
 return null;
}
```

## getLevelName

**Syntax** string Sorter.getLevelName( )

Returns dimension level to sort on.

**Returns** String. The name of a dimension level.

**Example** This example retrieves the level name associated with the sorter:

```
function getSortLevel(sorter) {
 if (sorter) {
 return sorter.getLevelName();
 }
 return null;
}
```

## getMember

**Syntax** actuate.xtabanalyzer.MemberValue Sorter.getMember( )

Returns the member value to sort on.

**Returns** actuate.xtabanalyzer.MemberValue object. A member value.

**Example** This example retrieves the sort member:

```
function getSortMember(sorter) {
 if (sorter) {
 return sorter.getMember();
 }
 return null;
}
```

## isAscending

**Syntax** boolean Sorter.isAscending( )

Returns the sort order.

**Returns** Boolean. True when the sorter is ascending and false in all other cases.

**Example** This example retrieves the level name that is associated with the sorter:

```
function ascending(sorter) {
 if (sorter) {
 return sorter.isAscending();
 }
 return null;
}
```

## setAscending

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

Sets the sort order to ascending or descending.

**Parameter** **ascending**

Boolean. Set to true for ascending, set to false for descending.

**Example** This example swaps the sort direction:

```
sorter.setAscending(! (sorter.isAscending));
```

## setKey

**Syntax** void Sorter.setSortKey(string sortKey)

Sets the key to sort on.

**Parameter** **sortKey**

String. The sort key.

**Example** This example sets the sort key:

```
function setSortKey(sorter,key) {
 sorter.setKey(key);
}
```

## setLevelName

**Syntax** void Sorter.setLevelName(string levelName)

Sets the dimension level name to sort on.

**Parameter** **levelName**

String. A dimension level name.

**Example** This example sets the level name to sort:

```
function setSortLevel(sorter,level){
 sorter.setLevelName(level);
}
```

## setMember

**Syntax** void Sorter.setMember(actuate.xtabanalyzer.MemberValue member)

Sets the member value to sort on.

**Parameter** **member**

actuate.xtabanalyzer.MemberValue object. A member value.

**Example** This example sets the sort member:

```
function setSortMember(sorter,member){
 sorter.setMember(member);
}
```

---

## Class actuate.xtabanalyzer.SubTotal

**Description** A SubTotal object.

### Constructor

**Syntax** actuate.xtabanalyzer.SubTotal( )

Constructs a new SubTotal object.

### Function summary

Table 8-19 lists actuate.xtabanalyzer.SubTotal functions.

**Table 8-19** actuate.xtabanalyzer.SubTotal functions

Function	Description
addTotal( )	Add a total
getLevelName( )	Returns the full level name
getLocation( )	Returns the location
getTotals( )	Returns the totals array
getType( )	Returns the type string
setLevelName( )	Sets the full level name
setLocation( )	Sets the location
setTotals( )	Sets the totals array

### addTotal

**Syntax** void SubTotal.addTotal(actuate.xtabanalyzer.Total total)

Adds a total to the subtotal.

**Parameter** **total**

actuate.xtabanalyzer.Total. The total object being added.

**Example** This example uses addTotal( ) to create a subtotal:

```
function addSubTotal() {
 var subTotal = new actuate.xtabanalyzer.SubTotal();
 subTotal.setLevelName("year");
 subTotal.setLocation("after");
 var indexStr = "0;1;2;3;4";
 var indexs = indexsStr.split(";");
 var measureIndexs = [];
 for(var i = 0;i < indexs.length;i++) {
```

```

 measureIndexes.push(parseInt(indexes[i]));
 }
 for(var i = 0; i < measureIndexes.length; i++) {
 var total = new actuate.xtabanalyzer.Total();
 total.setMeasureIndex(measureIndexes[i]);
 total.setAggregationFunction("SUM");
 total.setEnabled(true);
 subTotal.addTotal(total);
 }
 crosstab.setTotals(null,subTotal);
 crosstab.submit();
}

```

## getLevelName

**Syntax** string SubTotal.getLevelName( )

Returns the level for the subtotal.

**Returns** String. The level name for the subtotal.

**Example** This example retrieves the level name from the subtotal:

```

function getLevelName(subTotal) {
 if (subTotal){
 return subTotal.getLevelName();
 }
 return null;
}

```

## getLocation

**Syntax** string SubTotal.getLocation( )

Returns the location name for the subtotal.

**Returns** String. The location name.

**Example** This example retrieves the level name from the subtotal:

```

function getLocation(subTotal) {
 if (subTotal){
 return subTotal.getLocation();
 }
 return null;
}

```

## getTotals

**Syntax** object[ ] SubTotal.getTotals( )

## actuate.xtabanalyzer.SubTotal

Returns the totals used to calculate the subtotal.

**Returns** actuate.xtabanalyzer.Total[ ]. An array of total objects.

**Example** This example retrieves the totals from a SubTotal object:

```
var totalsArray = [];
function getTotals(subTotal,totalsArray){
 totalsArray = subTotal.getTotals();
}
```

## getType

**Syntax** string SubTotal.getType( )

Returns the type for the subtotal.

**Returns** String. The type for the subtotal.

**Example** This example retrieves the type from the subtotal:

```
function getLevelName(subTotal){
 if (subTotal){
 return subTotal.getType();
 }
 return null;
}
```

## setLevelName

**Syntax** void SubTotal.setLevelName(string levelName)

Sets the level for the subtotal by name.

**Parameter** **levelName**

String. The level name.

**Example** This example sets the level name for a subtotal:

```
function subTotalLevel(subTotal,levelName){
 if(subTotal){
 subTotal.setLevelName(levelName);
 }
}
```

## setLocation

**Syntax** void SubTotal.setLocation(string location)

Sets the location for the subtotal.

**Parameter** **location**

String. The location. Value can be either before or after.

**Example** This example sets the location for a subtotal:

```
function subTotalLocation(subTotal,location){
 if(subTotal){
 subTotal.setLocation(location);
 }
}
```

## setTotals

**Syntax** void SubTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)

Sets the totals using an array.

**Parameter** **totals**

Array of actuate.xtabanalyzer.Total objects to add to the subtotal.

**Example** This example uses setTotals( ) to create a subtotal:

```
function addSubTotal(){
 var subTotal = new actuate.xtabanalyzer.SubTotal();
 subTotal.setLevelName("year");
 subTotal.setLocation("after");
 var indexStr = "0;1;2;3;4";
 var indexs = indexStr.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 8-20 lists actuate.xtabanalyzer.Total functions.

**Table 8-20** actuate.xtabanalyzer.Total functions

Function	Description
getAggregationFunction( )	Returns the aggregation function name
getMeasureIndex( )	Returns the measure index
isEnabled( )	Returns whether or not the total is enabled
setAggregationFunction( )	Sets the aggregation function name
setEnabled( )	Sets the enabled flag
setMeasureIndex( )	Sets the index for the total

### getAggregationFunction

**Syntax** string Total.getAggregationFunction( )

Returns the aggregation function for the total.

**Returns** String. An aggregation function.

**Example** This example changes the aggregation function:

```
function swapTotalAggregation(total) {
 if (total.getAggregationFunction() == "SUM") {
 total.setAggregationFunction("COUNT");
 } else {
 total.setAggregationFunction("SUM");
 }
}
```

## getMeasureIndex

**Syntax** integer Dimension.getMeasureIndex( )  
 Retrieves the measure index for the total.

**Returns** Integer. The measure index.

**Example** This example retrieves the measure index:

```
function getMeasureIndex(total) {
 if (total) {
 return total.getIndex();
 }
 return null;
}
```

## isEnabled

**Syntax** boolean Total.isEnabled( )  
 Returns whether the total is enabled.

**Returns** Boolean. True indicates this total is enabled.

**Example** This example enables and disables a total:

```
if (total) {
 if (total.isEnabled) {
 total.setEnabled(false);
 } else {
 total.setEnabled(true);
 }
}
```

## setAggregationFunction

**Syntax** void Total.setAggregationFunction(string aggregationFunction)  
 Sets the aggregation function name.

**Parameter** **aggregationFunction**  
 String. The aggregation function name.

**Example** This example changes the aggregation function:

```
function swapTotalAggregation(total) {
 if (total.getAggregationFunction() == "SUM") {
 total.setAggregationFunction("COUNT");
 } else {
 total.setAggregationFunction("SUM");
```

```
actuate.xtabanalyzer.Total
```

```
 }
}
```

## setEnabled

**Syntax** void Total.setEnabled(boolean enabled)

Sets whether total is enabled or disabled.

**Parameter** **enabled**

Boolean. True if the total is enabled. False for disabled.

**Example** This example enables and disables a total:

```
if (total){
 if (total.isEnabled){
 total.setEnabled(false);
 } else {
 total.setEnabled(true);
 }
}
```

## setMeasureIndex

**Syntax** void Total.setMeasureIndex(integer measureIndex)

Sets the measure index for the total.

**Parameter** **measureIndex**

Integer. The measure index for the total.

**Example** This example uses setMeasureIndex( ) to create a subtotal:

```
function addSubTotal(){
 var subTotal = new actuate.xtabanalyzer.SubTotal();
 subTotal.setLevelName("year");
 subTotal.setLocation("after");
 var indexStr = "0;1;2;3;4";
 var indexs = indexsStr.split(",");
 var count = indexs.length;
 var measureIndexs = [];
 for(var i = 0;i < count;i++){
 measureIndexs.push(parseInt(indexs[i]));
 }
 for(var i = 0; i < measureIndexs.length; i++) {
 var total = new actuate.xtabanalyzer.Total();
 total.setMeasureIndex(measureIndexs[i]);
 total.setAggregationFunction("SUM");
 total.setEnabled(true);
 subTotal.addTotal(total);
 }
}
```

```
actuate.xtabanalyzer.Total
```

```
}
```

```
crosstab.setTotals(null,subTotal);
```

```
crosstab.submit();
```

```
}
```

---

## Class actuate.xtabanalyzer.UIOptions

**Description** Specifies feature availability for the Interactive Crosstabs viewer.

### Constructor

**Syntax** void actuate.xtabanalyzer.UIOptions( )

Generates a new UIOptions object to manage the features of the xtabanalyzer.

### Function summary

Table 8-21 lists actuate.xtabanalyzer.UIOptions functions.

**Table 8-21** actuate.xtabanalyzer.UIOptions functions

Function	Description
enableCrosstabView( )	Enables the cross tab layout view feature
enableCubeView( )	Enables the cube view feature
enableFilterSummaryView( )	Enables the filter summary view
enableToolBar( )	Enables the toolbar feature
enableToolbarHelp( )	Enables the toolbar help feature
enableToolbarSave( )	Enables the toolbar save feature
enableToolbarSaveDesign( )	Enables the toolbar save design feature
enableToolbarSaveDocument( )	Enables the toolbar save document feature
getFeatureMap( )	Returns a list of enabled and disabled features

### enableCrosstabView

**Syntax** void UIOptions.enableCrosstabView(boolean enabled)

Enables or disables the cross tab layout view.

**Parameter** **enabled**

Boolean. True enables this option.

**Example** This example enables or disables the cross tab view:

```
function setCrosstabView(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableCrosstabView(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableCubeView

**Syntax** void UIOptions.enableCubeView(boolean enabled)

Enables or disables the cube view.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables the cube view:

```
function setCubeView(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableCubeView(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableFilterSummaryView

**Syntax** void UIOptions.enableFilterSummaryView(boolean enabled)

Enables or disables the filter summary view.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables the filter summary view:

```
function setFilterSummary(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableFilterSummaryView(enabled);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableToolBar

**Syntax** void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables the toolbar:

```
function setToolbar(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableToolBar(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableToolbarHelp

**Syntax** void UIOptions.enableToolbarHelp(boolean enabled)

Enables or disables the toolbar help feature.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables toolbar help:

```
function setToolbarHelp(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableToolbarHelp(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableToolbarSave

**Syntax** void UIOptions.enableToolbarSave(boolean enabled)

Enables or disables the toolbar save feature.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables toolbar save:

```
function setToolbarSave(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableToolbarSave(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableToolbarSaveDesign

**Syntax** void UIOptions.enableToolbarSaveDesign(boolean enabled)

Enables or disables the toolbar save design feature.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables toolbar save design:

```
function setToolbarSave(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableToolbarSaveDesign(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## enableToolbarSaveDocument

**Syntax** void UIOptions.enableToolbarSaveDocument(boolean enabled)

Enables or disables the toolbar save document feature.

**Parameter** **enabled**

Boolean. A value of true enables this option.

**Example** This example enables or disables toolbar save document:

```
function setToolbarSave(flag) {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 uiOptions.enableToolbarSaveDocument(flag);
 myXTabAnalyzer.setUIOptions(uiOptions);
}
```

## getFeatureMap

**Syntax** Object UIOptions.getFeatureMap( )

Returns the features and their Boolean values as an associative array. This function makes the name of each feature an object property and sets the value of that property to the associated enabled Boolean value.

**Returns** Object. An associative array of string name and Boolean value pairs.

**Example** This example retrieves the feature map:

```
function retrieveFeatureMap() {
 var uiOptions = new actuate.xtabanalyzer.UIOptions();
 var features = uiOptions.getFeatureMap();
 return features;
}
```

actuate.xtabanalyzer.UIOptions

# Part **Three**

---

**Introduction to the Actuate  
Information Delivery API**



# Understanding the Information Delivery API and schema

This chapter contains the following topics:

- About the Actuate Information Delivery API
- About web services and WSDL
- Understanding the elements of the iHub WSDL schema
- Accessing the Actuate schema using a web browser

---

## About the Actuate Information Delivery API

The Actuate Information Delivery Application Programming Interface (IDAPI) is a Simple Object Access Protocol (SOAP) message schema that supports integrating and administering BIRT iHub using Extensible Markup Language (XML). Using the Information Delivery API, developers create applications that perform such tasks as uploading and downloading files, generating a document and scheduling document generation, sending an e-mail notification when a job completes, and accessing external libraries.

SOAP is the underlying layer that provides a messaging framework for web services. The IDAPI schema defines the message elements for a well-formed request to BIRT iHub and the format of the response.

The Actuate Information Delivery API has the following features:

- Platform- and language-independent access to Actuate web services  
Using SOAP messaging, Actuate's web services integrate into applications developed in Java, Visual Basic, C++, C#, and other programming languages. The SOAP serialization and deserialization framework translates XML messages to the language of the calling application.  
Deployment of these applications can be across multiple platforms, including UNIX, Windows, or Linux, and integrate with web technologies such as J2EE and Microsoft Visual Studio .NET.
- Comprehensive volume administration  
The volume is the central repository for the design files, folders, and other items that BIRT iHub stores and manages. The Actuate Information Delivery API, can manage the items in a volume from a single machine, and can send success or failure notices for immediate and scheduled jobs using Simple Mail Transfer Protocol (SMTP), or UNIX sendmail. The notifications can include an attachment or embedded file.
- Open infrastructure  
The Actuate Information Delivery API supports BIRT iHub's open-server infrastructure. This infrastructure generates, distributes, and manages Actuate and third-party designs and integrates the designs into a volume. The API also automates extracting parameters from a third-party executable file when integrating the file into a volume.
- Localization support  
By setting the Locale parameter in the header of a SOAP request, localization generates data in a language other than the BIRT iHub default language. The data display uses the currency format, date format, and other conventions for specific locales.

---

## About web services and WSDL

Actuate web services support interactions with BIRT iHub, from running and viewing designs to managing volume items. Actuate describes its services using Web Services Description Language (WSDL), an XML schema that provides the structure for requests to and responses from BIRT iHub.

A WSDL schema provides an abstract definition of the operations that a web service supports. The schema describes web services by providing such information as the name of the service, the transport protocol, data types, messages and operations, and input and output parameters. The schema binds these definitions to a concrete network protocol and message format. The WSDL schema resides on BIRT iHub.

The iHub WSDL schema serves as an interface to applications that integrate with BIRT iHub. The schema encompasses all web services accessible using the Actuate Information Delivery API. A developer can use the schema to generate a code library that contains the classes, including proxies, that the developer uses to write an application that communicates with a web service.

A developer can use an integrated development environment (IDE) to develop an application that uses the Information Delivery API schema. Actuate supplies WSDL files for the Apache Axis and Microsoft .NET development environments.

The elements of the iHub schema are case-sensitive. Capitalization must occur as the schema indicates. For example, to use the AuthId element of the complex data type Header, write AuthId instead of AuthID or authid.

---

## Understanding the elements of the iHub WSDL schema

A WSDL file defines every element used in a SOAP message. Figure 9-1 shows the basic structure of a WSDL file. Additional details about how the Actuate Information Delivery API defines each element in the following sections.

### About the definitions element

Because it is a definitions document, the WSDL schema begins with a definitions element. The opening tag defines the following Actuate namespace and attribute declarations:

```
<definitions
 name="ActuateAPI"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:typens="http://schemas.actuate.com/actuate11"
```

```
xmlns:wsdlNs="http://schemas.actuate.com/actuate11/wsdl"
targetNamespace="http://schemas.actuate.com/actuate11/wsdl">
```

```
<definitions>
```

The entire WSDL file is wrapped in `<definitions>` `</definitions>` tags.

```
<types>
```

This element defines the data types used in the Actuate WSDL file.

```
</types>
```

```
<message>
```

This element defines the structure of each Actuate message. A message is a request to perform an operation or a response to a request.

```
</message>
```

```
<portType>
```

This element defines each operation that iHub System recognizes. An operation is a task to perform.

```
<operation>
```

Within the portType, input and output structures further define each operation.

```
</operation>
```

```
</portType>
```

```
<binding>
```

The binding element describes how to invoke the service. Actuate uses HTTP and the SOAP protocol. The binding defines the SOAP elements for each operation.

```
</binding>
```

```
<service>
```

The service element names the service and provides its location.

```
</service>
```

```
</definitions>
```

**Figure 9-1** The basic structure of a WSDL file

Table 9-1 describes each declaration. These declarations are subject to change over time.

**Table 9-1** Namespace and attribute declarations for the definitions element

Declaration	Description
name="ActuateAPI"	Names the Actuate service

**Table 9-1** Namespace and attribute declarations for the definitions element

Declaration	Description
xmlns="http://schemas.xmlsoap.org/wsdl/"	Defines a namespace for the WSDL specification to which Actuate adheres
xmlns:xsd="http://www.w3.org/2001/XMLSchema"	Defines a namespace prefix, xsd, for the XML schema standard
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"	Defines a namespace prefix, soap, for the SOAP specification to which Actuate messages adhere
xmlns:typens="http://schemas.actuate.com/actuate11"	Defines a namespace prefix, typens, for the Actuate 11 XML schema
xmlns:wsdlNs="http://schemas.actuate.com/actuate11/wsdl"	Defines a namespace prefix, wsdlNs, for Actuate 11 web services
targetNamespace="http://schemas.actuate.com/actuate11/wsdl"	Scopes messages to the Actuate WSDL file for Actuate 11

## About data type definitions

The types element of a schema describes every complex data type that Actuate web services recognize. The types element begins with the following declarations:

```
<types>
 <xsd:schema
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="http://schemas.actuate.com/actuate11"
 elementFormDefault="qualified">
```

Two of these declarations are unique to the types element:

- xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" specifies the encoding scheme for serializing and deserializing SOAP messages.
- elementFormDefault indicates whether the target namespace must qualify all locally declared elements in the instance document. A value of qualified requires checking the target namespace to see that the instance document conforms to the target namespace element declarations and type definitions. A value of unqualified does not require checking.

The types element gives a data type a name and defines the structure. The types element describes whether there is a required sequence for the elements that define the structure. The following example shows the complete description of the complex data type for the Login request:

```

<xsd:complexType name="Login">
 <xsd:sequence>
 <xsd:element name="User" type="xsd:string"/>
 <xsd:element name="Password" type="xsd:string"
 minOccurs="0"/>
 <xsd:element name="EncryptedPwd" type="xsd:string"
 minOccurs="0"/>
 <xsd:element name="Credentials" type="xsd:base64Binary"
 minOccurs="0"/>
 <xsd:element name="Domain" type="xsd:string" minOccurs="0"/>
 <xsd:element name="UserSetting" type="xsd:boolean"
 minOccurs="0"/>
 <xsd:element name="ValidateUserGroups"
 type="typens:ArrayOfString" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:element name="Login" type="typens:Login"/>

```

In the preceding example:

- The xsd: namespace prefix refers to the version of the XML schema that Actuate uses. Actuate reserves the xsd: namespace prefix to refer to the 2001 version of the standard XML schema.
- ```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```
- The xsd: prefix appears in every tag in the schema.
- The complex data type is Login.
Child elements define this data type. Each child element has attributes such as the data type, the name, and the minimum or maximum number of occurrences. If the child element defines a data value that the data type requires, there is no minOccurs attribute. In this example, User is the only required element.
 - Because <sequence> </sequence> tags enclose this element list, you must use these elements in the sequence shown. If <all> </all> tags enclose the elements, they can appear in any order. If <choice> </choice> tags enclose the elements, you can choose one element.

About message definitions

The message element describes the parts of each request and response message in the Actuate Information Delivery API.

Actuate provides versions of its schema for the Microsoft .NET and Apache Axis environments. These environments require slightly different syntax for WSDL definitions. For example, a message definition contains a header element in

the .NET version but not in the Apache Axis version. The code examples in this chapter use the Apache Axis version.

The following example shows the Apache Axis message description for a request to search for jobs:

```
<message name="SelectJobs">
  <part name="Request" element="typens:SelectJobs" />
</message>
```

Like most requests, this one has a corresponding response.

```
<message name="SelectJobsResponse">
  <part name="Response" element="typens:SelectJobsResponse" />
</message>
```

Operations that do not require a response do not have a corresponding response.

About the portType definition

The portType element defines a structure for each operation in the Actuate schema. A unique name identifies the portType:

```
<portType name="ActuateSoapPort">
```

For most operations, portType defines an input message, the request, and an output message, the response.

The following example shows the portType definition for ActuateSoapPort:

```
<operation name="GetFileACL">
  <input message="wsdlns:GetFileACL"/>
  <output message="wsdlns:GetFileACLRResponse"/>
</operation>
```

If an operation does not require a response, portType defines only an input message. In the Microsoft .NET version of the schema, the operations also include a header element.

About the binding definition

The binding element defines how the operations and messages in the schema communicate. Actuate defines its binding using the following attributes:

- name="ActuateSoapBinding"
- type="wsdlns:ActuateSoapPort"
- style="document"
- transport="http://schemas.xmlsoap.org/soap/http"

The following example shows the binding definition for ActuateSoapBinding:

```
<binding name="ActuateSoapBinding" type="wsdlns:ActuateSoapPort">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

Within the <binding> </binding> tags, input and output child elements define the bindings for each operation that the portType section describes. Each input message consists of a SOAP header with attributes and a SOAP body with attributes. Each output message consists of a SOAP body with attributes.

The following example shows the input and output child elements for the Login operation:

```
<operation name="Login">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" parts="Request"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
```

These elements specify that Actuate does not use soapAction, a required element for SOAP messages. They further show that Actuate operations are literal messages that do not use separate wrappers for each element.

About the service definition

The WSDL schema presents its various operations as a single web service. The service element defines that service, ActuateAPI. The service definition includes:

- The name of the requested service.

```
service name="ActuateAPI"
```

- The name of the port through which the client accesses the iHub web services.

```
port name="ActuateSoapPort"
```

- The binding definition expressed as a namespace.

```
binding="wsdlns:ActuateSoapBinding"
```

- The location of the port, defined as host_name:port_number, expressed as a valid URL.

```
soap:address location="http://localhost:8000"
```

The following example shows the service definition for ActuateAPI:

```
<service name="ActuateAPI">
  <port name="ActuateSoapPort"
    binding="wsdl:ActuateSoapBinding">
    <soap:address location="http://localhost:8000"/>
  </port>
</service>
```

Accessing the Actuate schema using a web browser

To access the list of BIRT iHub WSDL files using a web browser, type the following URL:

`http://localhost:8000/wsdl`

where

- localhost is the machine name.
- 8000 is the port to which the SOAP endpoint binds.

This URL retrieves the WSDL file list for the Apache Axis and Microsoft .NET development environments, as shown in Figure 9-2.

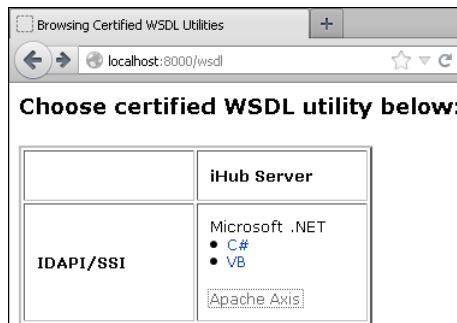


Figure 9-2 Accessing the iHub WSDL files using a web browser

To run a utility that displays the list of classes defined in a WSDL file, choose one of the items in the list. For example, in IDAPI/SSI, choose Apache Axis to view the list of operation classes defined in that WSDL file. The list of Information Delivery API operation classes appears, as shown in Figure 9-3.

To view the WSDL schema definition for a particular operation class, choose the class name. To view the entire contents of the WSDL file, choose All or type the following URL:

`http://localhost:8000/wsdl/v11/axis/all`

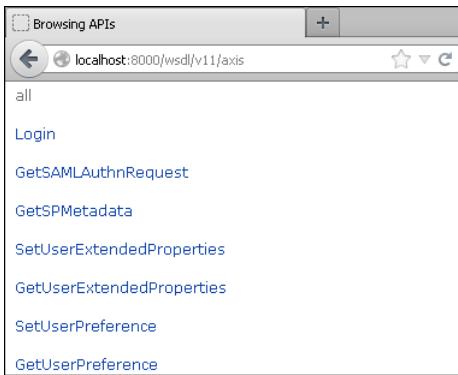


Figure 9-3 Actuate Information Delivery API operations

Choosing All displays the entire WSDL document, as shown in Figure 9-4. Choosing Login displays the schema for the Login operation, preceded by iHub namespace information, as shown in Figure 9-4.

```

<definitions name="ActuateAPI" targetNamespace="http://schemas.actuate.com/actuate11/wsdl">
  - <types>
    - <xsd:schema targetNamespace="http://schemas.actuate.com/actuate11" elementFormDefault="qualified">
      - <xsd:complexType name="Header">
        - <xsd:sequence>
          <xsd:element name="AuthId" type="xsd:string"/>
          <xsd:element name="TargetVolume" type="xsd:string" minOccurs="0"/>
          <xsd:element name="Locale" type="xsd:string" minOccurs="0"/>
          <xsd:element name="ConnectionHandle" type="xsd:string" minOccurs="0"/>
          <xsd:element name="TargetServer" type="xsd:string" minOccurs="0"/>
          <xsd:element name="DelayFlush" type="xsd:boolean" minOccurs="0"/>
          <xsd:element name="FileType" type="xsd:string" minOccurs="0"/>
          <xsd:element name="TargetResourceGroup" type="xsd:string" minOccurs="0"/>
          <xsd:element name="RequestID" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="Header" type="typens:Header"/>
    - <xsd:complexType name="Login">
      - <xsd:sequence>
        <xsd:element name="User" type="xsd:string"/>
        <xsd:element name="Password" type="xsd:string" minOccurs="0"/>
        <xsd:element name="EncryptedPwd" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Credentials" type="xsd:base64Binary" minOccurs="0"/>
        <xsd:element name="Domain" type="xsd:string" minOccurs="0"/>
        <xsd:element name="UserSetting" type="xsd:boolean" minOccurs="0"/>
        <xsd:element name="ValidateUserGroups" type="typens:ArrayOfString" minOccurs="0"/>
        <xsd:element name="RunAsUser" type="xsd:string" minOccurs="0"/>
        <xsd:element name="SAMLToken" type="xsd:string" minOccurs="0"/>
        <xsd:element name="GetVolumeList" type="xsd:boolean" minOccurs="0"/>
        <xsd:element name="CapabilityCategories" type="typens:ArrayOfString" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
<operations>
  <operation name="GetSAMLAuthnRequest">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="GetSPMetadata">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="SetUserExtendedProperties">
    <parameters>
      <parameter name="Header" type="Header"/>
      <parameter name="UserExtProp" type="xsd:string" minOccurs="0"/>
    </parameters>
  </operation>
  <operation name="GetUserExtendedProperties">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="SetUserPreference">
    <parameters>
      <parameter name="Header" type="Header"/>
      <parameter name="UserPref" type="xsd:string" minOccurs="0"/>
    </parameters>
  </operation>
  <operation name="GetUserPreference">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
</operations>
<binding name="AxisBinding" type="http://schemas.xmlsoap.org/soap/http">
  <operations>
    <operation name="GetSAMLAuthnRequest">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="GetSPMetadata">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="SetUserExtendedProperties">
      <parameters>
        <parameter name="Header" type="Header"/>
        <parameter name="UserExtProp" type="xsd:string" minOccurs="0"/>
      </parameters>
    </operation>
    <operation name="GetUserExtendedProperties">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="SetUserPreference">
      <parameters>
        <parameter name="Header" type="Header"/>
        <parameter name="UserPref" type="xsd:string" minOccurs="0"/>
      </parameters>
    </operation>
    <operation name="GetUserPreference">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
  </operations>
  <message name="GetSAMLAuthnRequestMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="GetSAMLAuthnRequest" />
  </message>
  <message name="GetSPMetadataMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="GetSPMetadata" />
  </message>
  <message name="SetUserExtendedPropertiesMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="SetUserExtendedProperties" />
  </message>
  <message name="GetUserExtendedPropertiesMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="GetUserExtendedProperties" />
  </message>
  <message name="SetUserPreferenceMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="SetUserPreference" />
  </message>
  <message name="GetUserPreferenceMessage">
    <part name="Header" type="Header"/>
    <part name="Body" type="GetUserPreference" />
  </message>
  <operation name="GetSAMLAuthnRequest">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="GetSPMetadata">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="SetUserExtendedProperties">
    <parameters>
      <parameter name="Header" type="Header"/>
      <parameter name="UserExtProp" type="xsd:string" minOccurs="0"/>
    </parameters>
  </operation>
  <operation name="GetUserExtendedProperties">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
  <operation name="SetUserPreference">
    <parameters>
      <parameter name="Header" type="Header"/>
      <parameter name="UserPref" type="xsd:string" minOccurs="0"/>
    </parameters>
  </operation>
  <operation name="GetUserPreference">
    <parameters>
      <parameter name="Header" type="Header"/>
    </parameters>
  </operation>
</binding>
<service name="ActuateService">
  <operations>
    <operation name="GetSAMLAuthnRequest">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="GetSPMetadata">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="SetUserExtendedProperties">
      <parameters>
        <parameter name="Header" type="Header"/>
        <parameter name="UserExtProp" type="xsd:string" minOccurs="0"/>
      </parameters>
    </operation>
    <operation name="GetUserExtendedProperties">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
    <operation name="SetUserPreference">
      <parameters>
        <parameter name="Header" type="Header"/>
        <parameter name="UserPref" type="xsd:string" minOccurs="0"/>
      </parameters>
    </operation>
    <operation name="GetUserPreference">
      <parameters>
        <parameter name="Header" type="Header"/>
      </parameters>
    </operation>
  </operations>
</service>

```

Figure 9-4 WSDL schema definitions

10

Constructing a SOAP message

This chapter contains the following topics:

- About SOAP messaging
- Calling an Actuate web service
- About SOAP message elements
- About SOAP Fault messages

About SOAP messaging

This chapter describes the elements of Actuate Simple Object Access Protocol (SOAP) messages. The Actuate Information Delivery API uses SOAP messaging in the request and response pattern for communications between the client and BIRT iHub. SOAP messages are written in XML to ensure standard message formatting and standard data representation.

The principal advantage of SOAP is that it supports communications among applications written in different programming languages and running on different platforms. SOAP supports Java, Visual Basic, C++, C#, and other programming languages. It operates on Windows, UNIX, Linux, Mac, and other operating systems.

Certain messages in the Actuate Information Delivery API can be composite messages, supporting multiple operations in a single message.

The Actuate Information Delivery API packages an XML request into a SOAP envelope and sends it to the BIRT iHub using a HyperText Transfer Protocol (HTTP) connection. Although a SOAP message can use other transport mechanisms, Actuate supports HTTP because this protocol is ubiquitous and because it simplifies external firewall management.

The client application sends the request and reads the response in the client's native language. The system's SOAP endpoints, ports that accept SOAP messages, listen for requests and direct them to the appropriate BIRT iHub node.

As with any other XML document, a SOAP message must be well formed and valid. A well-formed message has a single root, is correctly nested, and displays tags in starting and ending pairs. Valid XML is well formed and adheres to a schema. XML instruction is outside the scope of this book.

Calling an Actuate web service

When accessing Actuate's web services, you create a library of proxy objects for the client application. In Java, a proxy object is a class implementation in a JAR file. In C#, a proxy is a CS file.

Use a proxy object directly. Actuate does not support subclassing an Actuate Information Delivery API class generated from an Actuate WSDL document.

Access proxy objects using a request and response pattern. As Figure 10-1 shows, the client uses a proxy object to send a SOAP request to BIRT iHub and receives a response in the client's native language.

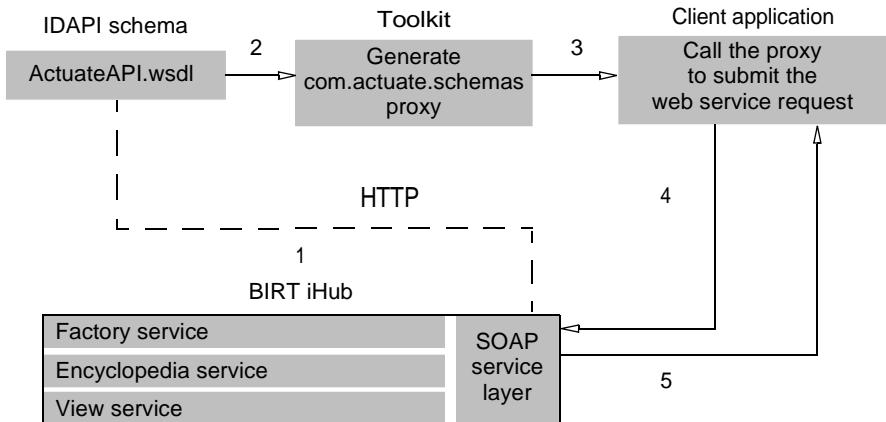


Figure 10-1 Calling an Actuate web service

In the sequence shown in Figure 10-1:

- 1 BIRT iHub sends the Information Delivery API schema over the web in response to a client query.
- 2 The client generates a proxy object that corresponds to a service or an operation in Actuate's schema.
- 3 The deployed client application calls a proxy object.
- 4 Using the proxy, the client generates a SOAP request, adds an HTTP header, and sends this serialized XML package to BIRT iHub over the web.
- 5 BIRT iHub processes the SOAP message header, deserializes the SOAP envelope, and invokes the appropriate service. In the preceding diagram, the Factory service processes the request.
- 6 The service serializes the result, creates the response XML, places the encoded result into a SOAP response, and returns the package to the client application. The application then extracts and decodes the result.

About SOAP message elements

The Actuate Information Delivery API uses the standard SOAP message structure. A message consists of an HTTP header followed by a SOAP envelope, header, and body. The SOAP envelope wraps the header and body elements. The HTTP header encloses the entire message, which goes over the web to a server that can accept SOAP messages.

The following sections provide details about each element of a SOAP message.

Understanding the HTTP header

This header is a mandatory element that specifies items such as the HTTP version, the host machine and port, the content type, and character set. The elements of an HTTP header can vary from message to message. The following example shows a typical HTTP header:

```
POST / HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related,
        text/*
User-Agent: Axis/2.0
Host: localhost:8080
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 1387
```

In the preceding example:

- POST routes the message to a servlet running on a web server, using HTTP version 1.0 or 1.1.
The message determines which HTTP version to use. Version 1.0 treats an attachment as a single block of data. Version 1.1 supports sending chunked attachments.
- Content-Type specifies the message's media type. Set Content-Type to text/xml when calling an Actuate service.
- The default character set is UTF-8. To use the UTF-8 character set, it is not necessary to include this element in the HTTP header.
- Accept indicates the acceptable types of media in the response:
 - The application/soap+xml media type describes a SOAP message serialized as XML.
 - The application/dime media type supports processing a message either using MIME or by reference to a uniform resource identifier (URI) that accesses a plug-in. A URI is a unique string that can be a uniform resource locator (URL), a uniform resource name (URN), or both. Using a URI ensures uniqueness.
 - The multipart/related media type indicates a compound object containing several inter-related parts in the body of a message.
 - The asterisk in text/* indicates the response can contain any type of text.
- User-Agent is the client that initiates the request.
- Host is the name of the host machine where the target BIRT iHub resides and, optionally, the port number.

- Cache-Control is a directive to any caching mechanism operating along the request-response transport layer. A no-cache directive keeps a cache from using the response to satisfy a subsequent request. This directive prevents the cache from returning a stale response to a client.
- Pragma is a directive to all recipients along the request-response transport layer. This no-cache directive requires the system to forward the original request to the target BIRT iHub even when a cached copy exists. Forwarding the original request prevents the transmission of a stale copy of a request.
- SOAPAction, a required element, tells the BIRT iHub that the message is a SOAP message. For an Actuate message, use a set of empty quotation marks for the SOAPAction value. The volume determines the action based on the message body. The SOAPAction attribute requires empty quotes because it has no default value.
- Content-Length is the number of characters in the message.

Understanding the SOAP envelope

The SOAP envelope is a required element of each message. It defines an overall framework for the message and contains other elements of the message. The envelope contains namespace declarations that apply to the specific message that contains them and to any child elements of that message.

An XML namespace is a unique identifier for the elements and attributes of an XML document. When declaring an XML namespace in a SOAP envelope, define the rules by which the system interprets the content and structure of the message.

To ensure that a namespace is globally unique, the namespace must be a URI. A namespace does not have to point to a web site or online document.

In the following example, the namespace declarations indicate that the message adheres to specific XML and SOAP standards and identifies the version of the Actuate XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header> ...
    <soapenv:Body>
        <GetVolumeProperties xmlns="http://schemas.actuate.com
            /actuate11">
            ...
        </GetVolumeProperties>
    </soapenv:Body>
</soapenv:Envelope>
```

In the preceding example:

- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"> refers to the SOAP standard that the message elements follow. A SOAP envelope must have an element that references this namespace or a SOAP VersionMismatch error occurs.
- xmlns:xsd="http://www.w3.org/2001/XMLSchema" defines the scope of the XML namespace. In this case, the namespace indicates that the message is based on the World Wide Web Consortium XML schema initially published in 2001. This namespace is declared in every SOAP message.
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" refers to the specific instance of the XML schema. In order to use namespace attribute types in an XML document, first define the xsi namespace.
- xmlns="http://schemas.actuate.com/actuate11" in <soapenv:Body> refers to the Actuate XML schema version to use.

About XML namespace declarations

XML provides support for combining data from multiple sources. In the process, however, it is possible to create confusion to tag disparate message elements with the same name.

For example, if you use the <Target> tag for data about quarterly sales goals in a Sales and Marketing application, and you use the <Target> tag to denote fiscal year revenue targets in a Finance application, errors can occur. Combining data from these two sources into one XML document, can cause naming collisions, or duplications, which may result in error messages or erroneous data. To avoid these collisions, use XML namespace declarations.

When constructing a request to BIRT iHub, use the namespaces the Actuate XML schema specifies. To use a namespace, first declare it in a header and then refer to it using the appropriate tag in the message. The following example shows how to declare two new namespaces by placing them in the SOAP envelope header:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    xmlns:fin="http://www.actuate.com/soap/Finance"
    xmlns:sales="http://www.actuate.com/soap/Sales and
    Marketing">
```

These new namespaces define prefixes, fin and sales in the example above, as shorthand for data sources. Add the prefix to the appropriate tag name to indicate

which data source to use. Use a simple prefix, separated from the tag name by a colon, as shown in the following example:

```
<fin:Target> </fin:Target>
```

Understanding the SOAP header

The SOAP header contains authentication data, locale information, and other required or optional data. The SOAP header element is mandatory for calls to the BIRT iHub. Using the SOAP header, parser tools can locate key information without having to parse the entire message.

The following example shows a typical SOAP header with authentication and locale information:

```
<soapenv:Header>
  <AuthId>
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0">
      u4yxAKHFJg9FY0JssYijJI5XvnpqDOPBOoWPbgRak20wIZIFDX6NY1o
      NsYg7RKzFt7GgtrOKqaas5HwLSkwhYEHEB19PuZim4kDS5g==
  </AuthId>
  <Locale
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0">en_US
  </Locale>
  <TargetVolume
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0">
  </TargetVolume>
  <ConnectionHandle
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0">
  </ConnectionHandle>
  <DelayFlush
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0">true
  </DelayFlush>
</soapenv:Header>
```

The Actuate Information Delivery API extends the standard SOAP header to use the following elements.

AuthId

When the client logs in using the Actuate Information Delivery API, the system returns a system-generated, encrypted AuthId string in the Login response. All requests except Login requests must have a valid AuthId in the SOAP header. The header passes this ID to BIRT iHub for validation.

The example shows a typical AuthId. In subsequent requests in the same session, the AuthId identifier appears in the SOAP header. AuthId expires after a configurable period of time.

In the example, AuthId contains two attributes:

- soapenv:actor
The value for actor, "http://schemas.xmlsoap.org/soap/actor/next", is a URI indicating that this message is for the first SOAP application capable of processing it.
- soapenv:mustUnderstand
Indicates that the actor must understand and process the message and, if it cannot, the actor must return a SOAP fault containing the value specified by the attribute. The mustUnderstand attribute can have a value of 0 or False, or 1 or True.

ConnectionHandle

An optional element that supports keeping a connection open to view a persistent document. ConnectionHandle is a session ID of the object.

ConnectionHandle supports phased downloading and viewing of a persistent report in the volume to improve performance. When ConnectionHandle is present in the header, iHub System ignores the value for TargetVolume.

ConnectionHandle returns in two ways:

- As an element of a document generation response when the document is transient and progressive viewing is enabled. BIRT iHub System routes subsequent viewing requests to the BIRT iHub that generated the transient document.
- As a response to a viewing request, to ensure that subsequent requests by the same user go to the same View service until the report data changes. If the report name changes but the data remains the same, the View service displays the same report. If ConnectionHandle is present in the SOAP header, the system routes subsequent viewing requests to the same View service that returned the ConnectionHandle.

DelayFlush

A Boolean element that tells BIRT iHub to write updates to the disk when the value is False.

FileType

An element that specifies the file type to run, such as an HTML or BIRT design (.rptdesign) file. Specify FileType in the SOAP header for all execute, submit, and view IDAPI requests, such as ExecuteReport, SelectPage, or GetContent.

When running a design that specifies a TargetResourceGroup, specify FileType in the SOAP header. BIRT iHub System looks for a machine that can execute that file type. The default setting is for BIRT iHub System to look for any available iHub to manage a request, whether or not that iHub can run the requested file type.

Locale

BIRT iHub uses this element to format data using the language, date and time conventions, currency and other locale-specific conventions before returning the data to the client. If the client does not specify another locale, BIRT iHub System uses the client's default locale.

TargetResourceGroup

An optional element that supports assigning a synchronous report generation request to a specific resource group at run time. When using ExecuteReport to run a BIRT design that specifies a TargetResourceGroup, you must specify FileType in the SOAP header.

TargetServer

An optional element that refers to the BIRT iHub within a cluster to which to direct the request. Use TargetServer for requests pertaining to system administration, such as GetFactoryServiceJobs and GetFactoryServiceInfo.

TargetVolume

Refers to the volume to which to direct the request. Use this optional element of the SOAP header to route a request to a volume other than the default volume for the system. In Release 10 and earlier, TargetVolume is an optional element. In Release 11 and iHub 2, the Login message required specifying the volume name using TargetVolume in the SOAP header and Domain in the SOAP message body. In iHub 3, the Login message once again no longer requires specifying a volume name using TargetVolume in the SOAP header or Domain in the SOAP message body.

RequestID

An optional element that represents a unique value identifying the SOAP request.

Understanding the SOAP message body

The body of the message contains either the request for a specific operation, the response to a request, or an error message. The following example requests detailed data about users on the volume:

```
<SOAP-ENV:Body>
  <SOAP-ACTU:SelectUsers
    xmlns:SOAP-ACTU="http://schemas.actuate.com/actuate11">
```

```

<ResultDef SOAP-ENC:arrayType="xsd:String[5]">
    <String>Id</String>
    <String>Name</String>
    <String>EmailAddress</String>
    <String>Homefolder</String>
    <String>Description</String>
</ResultDef>
<Search>
    <CountLimit>201</CountLimit>
    <FetchSize>100</FetchSize>
    <FetchDirection>true</FetchDirection>
</Search>
</SOAP-ACTU:SelectUsers>
</SOAP-ENV:Body>

```

The response includes details for each attribute in the request, including the attribute name and data type, when the attribute takes effect, and whether it is required.

```

<SOAP-ENV:Body>
    <ACTU:SelectUsersResponse
        xmlns:ACTU="http://schemas.actuate.com/actuate11"
        xmlns="http://schemas.actuate.com/actuate11">
        <Users>
            <User>
                <Name>Administrator</Name>
                <Id>1</Id>
                <EmailAddress></EmailAddress>
                <HomeFolder></HomeFolder>
                <Description></Description>
            </User>
            <User>
                <Name>User0</Name>
                <Id>2</Id>
                <EmailAddress>User0@localhost</EmailAddress>
                <HomeFolder>/home/User0</HomeFolder>
                <Description></Description>
            </User>
            ...
        </Users>
        <TotalCount>15</TotalCount>
    </ACTU:SelectUsersResponse>
</SOAP-ENV:Body>

```

About SOAP Fault messages

A SOAP Fault occurs when a request cannot be completed. Fault contains information identifying the source of the error or the component returning the error, the request, an error code, and a text description of the error.

In the following example, a request to download a file results in a SOAP Fault. The Description element contains a text error message and a reference to the requested file.

```
<SOAP-ENV:Body>
  <SOAP-ENV:Fault>
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <faultcode>Server</faultcode>
      <faultstring>Soap Server error.</faultstring>
      <detail>
        <RequestName>DownloadFile</RequestName>
        <ErrorCode>3072</ErrorCode>
        <Description>
          <Message>Cannot find the specified file or folder, or
            you do not have permission to access it.
          </Message>
          <Parameter1>/report/SampleReports.rptdesign</Parameter1>
        </Description>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
```


Part Four

**Developing Actuate Information
Delivery API applications**

11

Developing Actuate Information Delivery API applications using Java

This chapter contains the following topics:

- About the Apache Axis 2 client
- About the Actuate Information Delivery API framework
- Developing Actuate Information Delivery API applications

About the Apache Axis 2 client

BIRT iHub System contains a WSDL document that defines an Actuate web services schema for the Apache Axis 2 client. The Apache Axis 2 client is a Java-based framework for constructing a SOAP processor.

The Actuate Information Delivery API framework uses elements of the Apache Axis 2 code libraries to support the following features:

- The code emitter, org.apache.axis.wsdl.WSDL2Java, generates the Java source code package, com.actuate.schemas, from the Actuate WSDL document. The package contains the classes, including proxies, that you can use to write an Actuate Information Delivery API application to communicate with BIRT iHub System using SOAP messaging.
- The SOAP processor provides automatic JavaBean serialization and deserialization, using the com.actuate.schemas proxies, to encode and decode SOAP messages.

The following sections describe how to generate the com.actuate.schemas library and list the third-party code libraries required by the Actuate Information Delivery API development environment.

Generating the com.actuate.schemas library

The Apache Axis 2 client ships with BIRT iHub Integration Technology. In the web services examples, the Apache Axis 2 client is in the following directory:

\Actuate\ServerIntTech3\Web Services\Examples\Axis Client

You can generate the source code for the package, com.actuate.schemas, compile the classes, and archive the classes into a library file, using one of the following supplied methods:

- Batch
 - To use the batch file, build.bat, open a command prompt. Navigate to the Axis Client directory. At the command line, type:
build
- Apache Ant
 - To use Apache Ant, you must first install the build tool on your computer. To obtain the software and installation instructions, go to the Apache Ant Project web site at <http://ant.apache.org/>.
 - To use Apache Ant, open a command prompt. Navigate to the Axis Client directory.
 - To generate the source code for the package, com.actuate.schemas, type:
ant

- To compile the source code and generate the com.actuate.schemas library, type:

```
ant dist
```
- To generate Javadoc for the com.actuate.schemas library, type:

```
ant documentation
```

Each of these methods performs the operations by setting the properties that specify the locations and file names for the following resources:

- WSDL document
 The Actuate WSDL document is available at the following URL:
`http://localhost:8000/wsdl/v11/axis/all`
- Source code
 The code emitter, WSDL2Java, generates Java source code from the Actuate WSDL document, placing the package, com.actuate.schemas, in the directory, ACTUATE_HOME\ServerIntTech3\Web Services\Examples\Axis Client \source.
- Compiled code
 Both methods use javac to compile the source code, placing the compiled package, com.actuate.schemas, in the directory, ACTUATE_HOME \ServerIntTech3\Web Services\Examples\Axis Client\build.
- Library files
 Both methods use jar to archive the package, com.actuate.schemas. The batch method places the library file, idapi.jar, in the directory, \Actuate \ServerIntTech3\Web Services\Examples\Axis Client\lib.
 The Apache Ant method places the library file, ActuateClient-\${DSTAMP}.jar, in the directory, ACTUATE_HOME\ServerIntTech3\Web Services\Examples \Axis Client\dist\lib. DSTAMP is a variable in the file, build.xml, that represents the time when the JAR file was created. To run the example applications, copy the file, ActuateClient-\${DSTAMP}.jar, to the directory, \Actuate\ServerIntTech3\Web Services\Examples\Axis Client\dist\lib, and change the file name to idapi.jar.

About third-party code libraries

The BIRT iHub Integration Technology example applications require code libraries from the following third-party sources:

- Apache Axis
`http://xml.apache.org/axis`
- Apache log4j
`http://logging.apache.org`

- Jakarta Commons
<http://jakarta.apache.org/commons>
- JavaBeans™ Activation Framework (JAF)
<http://java.sun.com/products/javabeans/glasgow/jaf.html>
- JavaMail™
<http://java.sun.com/products/javamail>
- SOAP with Attachments API for Java™
<http://java.sun.com/xml/downloads/saaj.html>
- SourceForge.net Web Services Description Language for Java Toolkit (WSDL4J)
<http://sourceforge.net/projects/wsdl4j>
- Xerces XML Parser
<http://xml.apache.org/xerces2-j>

Actuate supplies the necessary libraries in the \lib directory of the BIRT iHub Integration Technology installation.

Tutorial 14: Generating com.actuate.schemas library

In this tutorial, you use the Apache Axis WSDL2Java tool to generate, compile, and deploy the com.actuate.schemas code library from the Actuate WSDL document using the build.bat script file packaged with Actuate iServer Integration Technology. You perform the following tasks:

- Access the Actuate WSDL document using a web browser
- Generate the com.actuate.schemas library from the Actuate WSDL document
- Examine the generated Java code library

To complete this tutorial, you need access to the following files from the ActuateiHubIntegrationTechnology-ihub3.zip archive:

```
<Extraction path>\ServerIntTech3\Web Services\Examples  
    \Axis Client\build.bat
```

To use the build.bat script file in this tutorial, you also need the Apache Ant build tool and environment variables set to access the iHub JDK and Apache Ant installations.

Task 1: Set the JAVA_HOME, ANT_HOME, and Path environment variables

To configure the Apache Ant to build iHub Integration Technology Java RSSE applications, you must set the JAVA_HOME, ANT_HOME, and Path variables on your computer.

- 1 Choose Start→Settings→Control Panel. In Windows 7, choose Start→Computer→System Properties.

Control Panel appears.

- 2 On Control Panel, double-click System.

System Properties—General appears.

- 3 On System Properties—General, choose Advanced.

System Properties—Advanced appears, as shown in Figure 11-1.

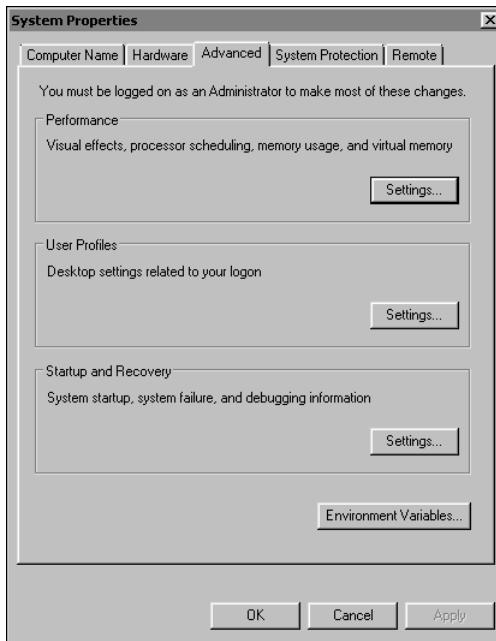


Figure 11-1 System Properties—Advanced

- 4 On System Properties—Advanced, choose Environment Variables. Environment Variables appears, as shown in Figure 11-2.

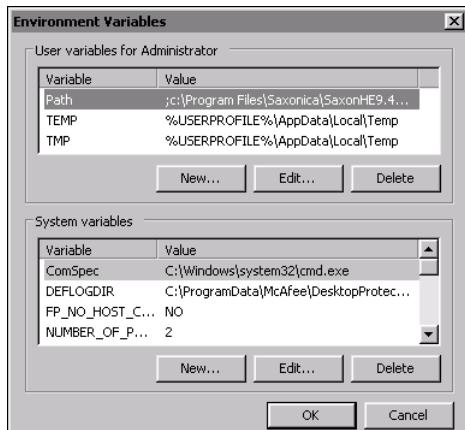


Figure 11-2 Environment Variables

- 5** If not already set, add the JAVA_HOME environment variable by performing the following tasks:

- 1 In System Variables, choose New.

New System Variable appears.

- 2 In New System Variable, in Variable Name, type:

JAVA_HOME

- 3 In Variable Value, type the following path specifying the directory where Actuate installed the Java, similar to the following:

C:\Actuate3\BIRTHubVisualization\modules\JDK64

- 4 New System Variable appears as shown in Figure 11-3.

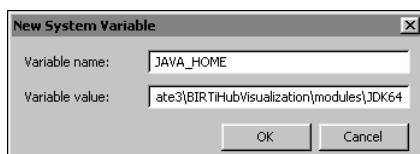


Figure 11-3 Creating a new JAVA_HOME system variable

Choose OK.

- 6** If already set, perform the following tasks to change the JAVA_HOME environment variable:

- 1 In System Variables, select JAVA_HOME. Choose Edit.

Edit System Variable appears.

- 2 On Edit System Variable, in Variable Name, type:

JAVA_HOME

- 3 In Variable Value, type the following path specifying the directory where Actuate installed the Java SDK, similar to the following:
C:\Actuate3\BIRTHubVisualization\modules\JDK64
- 4 Edit System Variable appears as shown in Figure 11-4.



Figure 11-4 Editing the JAVA_HOME system variable

Choose OK.

- 7 On Environment Variables, add the ANT_HOME environment variable by performing the following tasks:
 - 1 In System Variables, choose New.
New System Variable appears.
 - 2 On New System Variable, in Variable Name, type:
ANT_HOME
 - 3 In Variable Value, type the path for the directory where WinZip extracted the Apache Ant files, similar to the following:
C:\apache-ant-1.7.0
New System Variable appears, as shown in Figure 11-5.

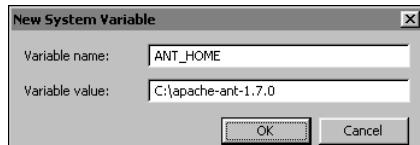


Figure 11-5 Creating a new ANT_HOME system variable

- 4 Choose OK.
- 8 On Environment Variables, edit the Path environment variable by performing the following tasks:
 - 1 In System Variables, select Path. Choose Edit.
Edit System Variable appears.
 - 2 In Variable Value, if not already set, append a semicolon at the end of the Path specification to separate the new path from the existing paths. Then, append the following text to the existing Variable Value text:
%JAVA_HOME%\bin;%ANT_HOME%\bin;

- 3 Edit System Variable appears, as shown in Figure 11-6.

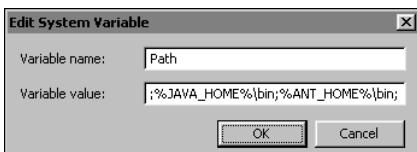


Figure 11-6 Editing the Path system variable

- 4 Choose OK.

9 On Environment Variables, choose OK.

10 To close System Properties, choose OK.

11 To close Control Panel, choose File->Close.

Task 2: Access the Actuate WSDL document using a web browser

You can use a web browser to view the Actuate 11 WSDL document on the volume.

1 In Windows, choose Start->Programs->Internet Explorer.

Internet Explorer appears.

2 Access the following URL:

<http://localhost:8000/wsdl>

Browsing Certified WSDL Utilities—Microsoft Internet Explorer appears, displaying links to the versions of Actuate WSDL documents, as shown in Figure 11-7.

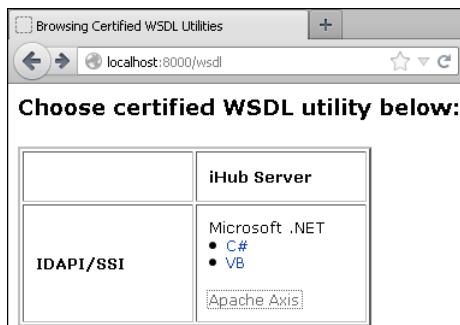


Figure 11-7 Links to environment-specific versions of the Actuate WSDL documents

- 3 In IDAPI Actuate 11, choose Apache Axis.

Browsing APIs—Microsoft Internet Explorer appears.

4 Choose Login.

The login page appears.

5 Examine the Login WSDL definition the WSDL utility displays, including the following elements:

- Namespace and attribute declarations
- Types
- Messages
- PortType
- Bindings
- Services
- Ports

Task 3: Generate the com.actuate.schemas library from the Actuate WSDL document

In this task, you generate the com.actuate.schemas library using the online version of the Actuate WSDL document. You run the WSDL2Java tool from a command window using the build.bat file.

1 Choose Start→Programs→Accessories→Command Prompt.

A command prompt window appears.

2 At the command prompt, type:

```
cd <Extraction path>\ServerIntTech3\Web Services\Examples  
    \Axis Client
```

Press Enter.

3 At the command prompt, type:

```
build
```

Press Enter.

Build.bat executes the following commands:

```
@if "%HOSTNAME%" == "" set HOSTNAME=localhost  
@if "%PORT%" == "" set PORT=8000  
@if not "%1" == "" set HOSTNAME=%1  
@if not "%2" == "" set PORT=%2  
mkdir build  
call setClassPath.bat  
ant compile -DhostName=%HOSTNAME% -Dport=%PORT%
```

Task 4: Examine the generated Java code library

In this task, you examine the source code files in the com.actuate.schemas library.

- 1 Using Windows Explorer, navigate to the following directory:

<Extraction path>\ServerIntTech3\Web Services\Examples
 \Axis Client\Source

- 2 Examine the following files in the com\actuate\schemas folder:

- Login.java

Examine the following items in the code:

- Java attribute and related accessor methods generated from the WSDL type definitions
 - Type metadata descriptors that map each Java attribute to its corresponding XML element
 - JavaBean getSerializer() and getDeserializer() methods

- ActuateSoapPort.java

Examine the following items in the code:

- ActuateSoapPort extends java.rmi.Remote defining the interface the application uses to access a web service in a remote procedure call
 - SDI for com.actuate.schemas.LoginResponse login()

- ActuateSoapBindingStub.java

Examine the following items in the code:

- Proxy code that enables the application to call a remote service as a local object
 - Stub that converts the Java call to a SOAP call for a login operation

About the Actuate Information Delivery API framework

The org.apache.axis.wsdl.WSDL2Java tool generates the Actuate Information Delivery API application framework based on the Actuate WSDL document definitions. This framework contains the client-side bindings that the Actuate IDAPI application requires to implement SOAP processing.

The SOAP processor serializes, or transforms, a remote procedure call by the application into an XML-based SOAP message that asks BIRT iHub to perform a web service. The application sends the request across the network using the HyperText Transfer Protocol (HTTP) transport layer.

BIRT iHub receives the request and deserializes the SOAP message. BIRT iHub performs an appropriate action and sends a response, in the form of a SOAP

message, back to the application. The SOAP processor embedded in the Actuate Information Delivery API framework automates the serialization and deserialization of JavaBeans, relieving the developer of the necessity to program the application at this level. The framework code is visible in the com.actuate.schemas classes.

The following sections describe these key elements of the framework as background information to provide the developer with an understanding of the way the Actuate Information Delivery API framework operates.

Using a data type from a WSDL document to generate a JavaBean

When you generate the Actuate Information Delivery API source code, the WSDL2Java tool builds a Java class from each WSDL type definition. For example, WSDL2Java translates the following Login type definition into its Java equivalent:

```
<xsd:complexType name="Login">
  <xsd:sequence>
    <xsd:element name="User" type="xsd:string"/>
    <xsd:element name="Password" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="EncryptedPwd" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="Credentials" type="xsd:base64Binary"
      minOccurs="0"/>
    <xsd:element name="Domain" type="xsd:string" minOccurs="0"/>
    <xsd:element name="UserSetting" type="xsd:boolean"
      minOccurs="0"/>
    <xsd:element name="ValidateUserGroups"
      type="typens:ArrayOfString" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The WSDL2Java tool gives the generated Java class the name that appears in the WSDL type definition. The class defines the attributes and data types for each WSDL element with corresponding accessor methods, as shown in the following code:

```
package com.actuate.schemas;

public class Login implements java.io.Serializable {
    private java.lang.String user;
    private java.lang.String password;
    private java.lang.String encryptedPwd;
    private byte[ ] credentials;
    private java.lang.String domain;
```

```

private java.lang.Boolean userSetting;
private com.actuate.schemas.ArrayOfString validateUserGroups;
public Login( ) {
}
...
public java.lang.String getUser( ) {
    return user;
}
public void setUser(java.lang.String user) {
    this.user = user;
}
...

```

Using metadata to map XML to a Java type

Mapping XML to a Java type requires creating a collection of descriptors in the class to associate each Java attribute with its corresponding XML element. This mapping system manages any naming differences between the Java and XML pairs to support the serialization and deserialization of the data.

The WSDL2Java tool generates a static type descriptor for each Java and XML pair. The following code maps the qualified names of the Java attribute and XML element for User in the Login class:

```

...
private static org.apache.axis.description.TypeDesc typeDesc =
new org.apache.axis.description.TypeDesc(Login.class, true);

static {
    typeDesc.setXmlType(new javax.xml.namespace.QName(
        "http://schemas.actuate.com/actuate11", "Login"));
    org.apache.axis.description.ElementDesc elemField =
        new org.apache.axis.description.ElementDesc();
    elemField.setFieldName("user");
    elemField.setXmlName(new javax.xml.namespace.QName(
        "http://schemas.actuate.com/actuate11", "User"));
    elemField.setXmlType(new javax.xml.namespace.QName(
        "http://www.w3.org/2001/XMLSchema", "string"));
    elemField.setNillable(false);
    typeDesc.addFieldDesc(elemField);
}
...
```

A class generated from a WSDL type is typically a JavaBean. The JavaBean uses classes from the org.apache.axis.encoding.ser package to encode and decode SOAP messages. In the following code example, getSerializer() instantiates and returns a reference to a BeanSerializer object using the Java and XML type descriptors:

```

...
public static org.apache.axis.encoding.Serializer getSerializer(
    java.lang.String mechType,
    java.lang.Class _javaType,
    javax.xml.namespace.QName _xmlType) {
    return
        new org.apache.axis.encoding.ser.BeanSerializer(
            _javaType, _xmlType, typeDesc);
}
...

```

Mapping the portType to a Service Definition Interface

WSDL2Java uses the portType and binding in the WSDL document to create the Service Definition Interface (SDI). The Service Definition Interface specifies the input and output messages of the request-response pairs for an operation and the service name and port.

The following WSDL code shows the specification of the input and output messages, Login and LoginResponse, and the binding of this request-response pair to the login operation:

```

...
<portType name="ActuateSoapPort">
    <operation name="login">
        <input message="wsdlns:Login"/>
        <output message="wsdlns:LoginResponse"/>
    </operation>
</portType>
...
<binding name="ActuateSoapBinding" type="wsdlns:ActuateSoapPort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="login">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" parts="Request" />
        </input>
        <output>
            <soap:body use="literal" parts="Response" />
        </output>
    </operation>
</binding>
...
```

The following WSDL code shows the specification of the service and port names, and the binding of the port to a machine address:

```

...
<service name="ActuateAPI">
    <port name="ActuateSoapPort"
        binding="wsdlNs:ActuateSoapBinding">
        <soap:address location="http://localhost:8000"/>
    </port>
</service>
...

```

An application uses this information to construct an interface to access the operations available from the service using a remote procedure call (RPC), as shown in the following code:

```

package com.actuate.schemas;

public interface ActuateSoapPort_PortType extends java.rmi.Remote
{
    public com.actuate.schemas.LoginResponse
        login(com.actuate.schemas.Login request)
        throws java.rmi.RemoteException;
...

```

In the example, the remote procedure call, `login()`, submits a request, passing a `Login` object as a parameter, and returns a `LoginResponse` object in response from the BIRT iHub defined by `ActuateSoapPort` in the SDI.

Using a WSDL binding to generate a Java stub

A Java stub consists of a class containing the proxy code that allows an application to call a remote service as a local object. Using a proxy, a developer does not have to specify the URL, namespace, or parameter arrays that the Service and Call objects require.

The stub converts the call to a Java method to a SOAP call. The stub constructor instantiates the service then adds the references for each qualified name, serializable class, and the JavaBean serialization and deserialization factories to Vectors to complete the implementation of the `ActuateSoapPort` interface, as shown in the following code:

```

package com.actuate.schemas;

public class ActuateSoapBindingStub extends
    org.apache.axis.client.Stub implements
    com.actuate.schemas.ActuateSoapPort_PortType {
    private java.util.Vector cachedSerClasses = new
        java.util.Vector( );
    private java.util.Vector cachedSerQNames = new
        java.util.Vector( )
    private java.util.Vector cachedSerFactories = new
        java.util.Vector( );
...

```

```

private java.util.Vector cachedDeserFactories = new
    java.util.Vector( );
...
public ActuateSoapBindingStub(javax.xml.rpc.Service service)
throws org.apache.axis.AxisFault {
    if (service == null) {
        super.service = new org.apache.axis.client.Service( );
    } else {
        super.service = service;
    }
...
qName = new javax.xml.namespace.QName(
    "http://schemas.actuate.com/actuate11", "Login");
cachedSerQNames.add(qName);
cls = com.actuate.schemas.Login.class;
cachedSerClasses.add(cls);
cachedSerFactories.add(beansf);
cachedDeserFactories.add(beandf);

```

Implementing the Actuate API service

ActuateAPI interface extends javax.xml.rpc.Service and defines the methods that get the URL for an Actuate SOAP port. The ActuateAPILocator class implements ActuateAPI interface to bind the SOAP port to a physical address. It returns this address using getActuateSoapPortAddress(), as shown in the following code:

```

package com.actuate.schemas;

public class ActuateAPILocator extends
    org.apache.axis.client.Service implements
    com.actuate.schemas.ActuateAPI {
    private final java.lang.String ActuateSoapPort_address =
        "http://ENL2509:8000";

    // Use to get a proxy class for ActuateSoapPort
    public java.lang.String getActuateSoapPortAddress( ) {
        return ActuateSoapPort_address;
    }

```

ActuateAPI interface specifies two versions of getActuateSoapPort() method to access a physical address. ActuateAPILocator.getActuateSoapPort() returns the default address set using attribute, ActuateSoapPort_address, as shown in the following code:

```

public com.actuate.schemas.ActuateSoapPort_PortType
getActuateSoapPort( )
throws javax.xml.rpc.ServiceException {
    java.net.URL endpoint;

```

```

    try {
        endpoint = new java.net.URL(ActuateSoapPort_address);
    }
    catch (java.net.MalformedURLException e) {
        throw new javax.xml.rpc.ServiceException(e);
    }
    return getActuateSoapPort(endpoint);
}

```

The overloaded version of ActuateAPILocator.getActuateSoapPort(java.net.URL portAddress) accepts a URL as a parameter. This version creates the service using the URL parameter as the endpoint, as shown in the following code:

```

public com.actuate.schemas.ActuateSoapPort_PortType
getActuateSoapPort
(java.net.URL portAddress) throws
javax.xml.rpc.ServiceException {
try {
    com.actuate.schemas.ActuateSoapBindingStub _stub =
    new com.actuate.schemas.ActuateSoapBindingStub(portAddress,
    this);
    _stub.setPortName(getActuateSoapPortWSDDServicename( ));
    return _stub;
}
catch (org.apache.axis.AxisFault e) {
    return null;
}
}

```

Developing Actuate Information Delivery API applications

To run the BIRT iHub Integration Technology example applications for the Apache Axis 2 client, open a command prompt. Navigate to the Axis Client directory.

In a Windows environment, you can run the file, setClassPath.bat, to set the environment variables needed to access the required source code, compiled classes, libraries, and other resources. setClassPath.bat contains the following environment variable settings:

```

set SAMPLEBASEDIR=.
set LIBDIR=%SAMPLEBASEDIR%\lib
set AXIS_JAR=%LIBDIR%\axis.jar;%LIBDIR%\commons-discovery.jar;
%LIBDIR%\commons-logging.jar;%LIBDIR%\jaxrpc.jar;%LIBDIR%
\log4j-1.2.4.jar;%LIBDIR%\wsdl4j.jar

```

```

set SUN_JAR=%LIBDIR%\activation.jar;%LIBDIR%\mail.jar;%LIBDIR%
\saaj.jar
set XMLPARSER_JAR=%LIBDIR%\xercesImpl.jar;%LIBDIR%
\xmlParserAPIs.jar
set CLASSPATH=%AXIS_JAR%;%SUN_JAR%;%SAMPLEBASEDIR%\build;
%SAMPLEBASEDIR%\source;%XMLPARSER_JAR%;%LIBDIR%\servlet.jar;

```

In a UNIX environment, you can run the shell script, setClassPath.sh. setClassPath.sh contains the following environment variable settings:

```

#!/bin/sh
export SAMPLEBASEDIR
export LIBDIR
export AXIS_JAR
export SUN_JAR
export XMLPARSER_JAR
export CLASSPATH
SAMPLEBASEDIR='pwd'
LIBDIR=$SAMPLEBASEDIR/lib
AXIS_JAR=$LIBDIR/axis.jar:$LIBDIR/commons-discovery.jar:$LIBDIR
/commons-logging.jar:$LIBDIR/jaxrpc.jar:$LIBDIR
/log4j-1.2.4.jar:$LIBDIR/wsdl4j.jar
SUN_JAR=$LIBDIR/activation.jar:$LIBDIR/mail.jar:$LIBDIR/saaj.jar
XMLPARSER_JAR=$LIBDIR/xercesImpl.jar:$LIBDIR/xmlParserAPIs.jar
CLASSPATH=$AXIS_JAR:$SUN_JAR:$SAMPLEBASEDIR/build:$SAMPLEBASEDIR
/source:$XMLPARSER_JAR:$LIBDIR/servlet.jar

```

The following sections describe the use of the Axis TCPMonitor utility to capture SOAP messages and the development process for following types of Actuate Information Delivery API applications:

- Writing a program that logs in to BIRT iHub System
- Catching a SOAP message with Axis TCPMonitor
- Writing a simple administration application
- Performing a search operation
- Writing a batch or transaction application
- Uploading a file
- Downloading a file
- Executing a report

The code examples and explanations in this chapter parallel the code examples and explanations in Chapter 12, “Developing Actuate Information Delivery API applications using Microsoft .NET.”

Writing a program that logs in to BIRT iHub System

A login operation authenticates a user in BIRT iHub System. A login operation involves the following actions:

- An IDAPI application sends a Login request to BIRT iHub System.
- BIRT iHub System. sends a Login response to the IDAPI application.

A Login request receives a Login response message from BIRT iHub System. When a Login request succeeds, the Login response message contains an AuthId, which is an encrypted, authenticated token. When a Login request fails, BIRT iHub System sends a Login response containing an error code and a description of the error.

The authentication ID in the Login response message remains valid for the current session. Any subsequent request that the application sends to BIRT iHub System must include the authentication ID in the message.

Each login operation uses the com.actuate.schemas classes to encode and decode the SOAP request and response messages. The following sections describe the SOAP messages, classes, and program interactions necessary to implement a successful login operation.

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. AcLogin class logs in to the BIRT iHub System to authenticate the user. If the login succeeds, the application prints a success message. If the login fails, the application prints a usage message.

```
import com.actuate.schemas.*;  
  
public class AcLogin {  
    public static final String usage =  
        "Usage:\n"+  
        "    java AcLogin [options]\n";  
  
    public static void main(String[ ] args) {  
        // set command line usage  
        Arguments.usage = usage;  
  
        // get command line arguments  
        Arguments arguments = new Arguments(args);  
  
        try {  
            // login to actuate server  
            AcController actuateControl = new  
                AcController(arguments.getURL( ));  
            actuateControl.setUsername(arguments.getUsername( ));  
            actuateControl.setPassword(arguments.getPassword( ));
```

```

        if (actuateControl.login( )) {
            System.out.println("Congratulations! You have
                successfully logged into BIRT iHub System as
                "+actuateControl.getUsername( ));
        }
        else {
            System.out.println("Please try again.\n Usage:\n"+
                " java AcLogin [options]\n");
        }
    }
    catch (Exception e) {
        e.printStackTrace( );
    }
}
}

```

About the auxiliary classes provided by the sample application

In the code example, the application makes use of the auxiliary classes provided by the BIRT iHub Integration Technology installation for the Apache Axis 2 client. These classes perform tasks common to most applications. The source code files for the auxiliary classes are in the source directory. The auxiliary classes are sample application components and are not part of the com.actuate.schemas package generated by the Actuate WSDL document:

- Arguments is an auxiliary class that analyzes the command line arguments to detect predefined options and enumerate any additional arguments. The following list describes the predefined options that can be specified at the command line:
 - serverURL
-h hostname specifies the SOAP endpoint. The default value, http://localhost:8000, is set in ActuateControl, another auxiliary class.
 - username
-u username specifies the user name. The default value, Administrator, is set in ActuateControl.
 - password
-p password specifies the password. The default value, "", is set in ActuateControl.
 - volume
-v volume specifies the target volume name. Actuate Release 11 requires a volume name in the SOAP header. For earlier and later releases, this specification is optional.

- `embeddedDownload`
`-e` turns on the download embedded option. When downloading a file, you can specify whether to embed the content in the response or transmit the file as an attachment. The default value, `False`, is set in Arguments.
- `printUsage`
`-?` prints the usage statement.
- `ActuateControl` is a controller class that handles routine interactions between the client application and the underlying `com.actuate.schemas` classes.
`ActuateControl` is a sample class. It is not a comprehensive implementation of all possible interactions. It implements the following essential parts of an IDAPI application:
 - Instantiates an `ActuateAPILocatorEx` object that implements the required Actuate SOAP header extensions and sets the BIRT iHub URL, as shown in the following code example:

```
public ActuateControl( ) throws
    MalformedURLException, ServiceException {
    actuateAPI = new ActuateAPILocatorEx( );
    setActuateServerURL(actuateServerURL);
}
```

 - Sets the endpoint for the proxy to the specified value of the volume URL, as shown in the following code example:

```
public void setActuateServerURL(String serverURL) throws
    MalformedURLException, ServiceException {
    if ((proxy == null) ||
        !serverURL.equals(actuateServerURL)) {
        if (serverURL != null)
            actuateServerURL = serverURL;
        System.out.println("Setting server to " +
            actuateServerURL);
        proxy =
            actuateAPI.getActuateSoapPort(
                new java.net.URL(actuateServerURL));
    }
}
```

 - Creates and configures an Axis Call object that sends the SOAP message to a volume, as shown in the following code example:

```
public org.apache.axis.client.Call createCall( ) throws
    ServiceException {
    org.apache.axis.client.Call call =
        (org.apache.axis.client.Call)
        actuateAPI.createCall( );
```

```

        call.setTargetEndpointAddress(this.actuateServerURL);
        return call;
    }
}

```

- ActuateAPIEx interface extends com.actuate.schemas.ActuateAPI. This interface defines the necessary Actuate SOAP header extensions and the Call object that returns the SOAP header element. ActuateAPIEx defines the following Actuate SOAP header extensions:
 - AuthId is a system-generated, encrypted String returned by BIRT iHub in a Login response. All requests, except a Login request, must have a valid AuthId in the SOAP header.
 - Locale specifies the language, date, time, currency and other conventions for BIRT iHub to use when returning the data to a client.
 - TargetVolume is an optional element that indicates the volume that receives the request.
 - ConnectionHandle supports keeping a connection open to view a persistent report. If ConnectionHandle is present in the SOAP header, the system routes subsequent viewing requests to the same View service that returned the ConnectionHandle.
 - DelayFlush tells BIRT iHub to write updates to the disk when the value is False.

The following example shows the code for the ActuateAPIEx interface:

```

public interface ActuateAPIEx extends
    com.actuate.schemas.ActuateAPI {
    public void setAuthId(java.lang.String authId);
    public void setLocale(java.lang.String locale);
    public void setTargetVolume(java.lang.String
        targetVolume);
    public void setConnectionHandle(java.lang.String
        connectionHandle);
    public void setDelayFlush(java.lang.Boolean delayFlush);

    public java.lang.String getAuthId();
    public java.lang.String getLocale();
    public java.lang.String getTargetVolume();
    public java.lang.String getConnectionHandle();
    public java.lang.Boolean getDelayFlush();

    public org.apache.axis.client.Call getCall();
}

```

- ActuateAPILocatorEx class extends com.actuate.schemas.ActuateAPILocator and implements the ActuateAPIEx interface. ActuateAPILocatorEx class creates the Call object and adds the Actuate SOAP header, as shown in the following code example:

```
public class ActuateAPILocatorEx
extends com.actuate.schemas.ActuateAPILocator
implements ActuateAPIEx {
...
public Call createCall( ) throws ServiceException {
    call = (org.apache.axis.client.Call) super.createCall( );
    if (null != authId)
        call.addHeader(new SOAPHeaderElement(null, "AuthId",
                                             authId));
    if (null != locale)
        call.addHeader(new SOAPHeaderElement(null, "Locale",
                                             locale));
    if (null != targetVolume)
        call.addHeader(
            new SOAPHeaderElement(null, "TargetVolume",
                                  targetVolume));
    if (null != connectionHandle)
        call.addHeader(
            new SOAPHeaderElement(
                null,
                "ConnectionHandle",
                connectionHandle));
    if (null != targetServer)
        call.addHeader(
            new SOAPHeaderElement(
                null,
                "TargetServer",
                targetServer));
    if (null != targetVolume)
        call.addHeader(
            new SOAPHeaderElement(null, "DelayFlush", delayFlush));
    return call;
}
```

Logging in to BIRT iHub System

In the example application, AcLogin class calls ActuateControl.login() method. This method instantiates the com.actuate.schemas.Login object, then sets the values for username and password, as shown in the following code:

```
public boolean login( ) {
    boolean success = true;
```

```
com.actuate.schemas.Login request =
    new com.actuate.schemas.Login( );
request.setPassword(password);
request.setUser(username);
```

ActuateControl.login() sets the AuthId to null, then uses the proxy to make a remote procedure call to BIRT iHub. If successful, the call returns a reference to com.actuate.schemas.LoginResponse object, as shown in the following code example:

```
...
try {
    actuateAPI.setAuthId(null);
    com.actuate.schemas.LoginResponse response =
        proxy.login(request);
    authenticationId = response.getAuthId();
    actuateAPI.setAuthId(authenticationId);
}
catch (java.rmi.RemoteException e) {
    // login failed
    success = false;
}
return success;
}
```

In the code example, the LoginResponse object contains the authentication ID returned by BIRT iHub. The application uses LoginResponse.getAuthId() to access the value, then calls ActuateAPIEx.setAuthId() to make the value available to the application to embed in the header of a subsequent SOAP request message.

Tutorial 15: Catching a SOAP message with Axis TCPMonitor

In this tutorial, you use AXIS TCPMonitor (tcpmon) utility to monitor the SOAP messages between an application and the volume. You configure TCPMonitor to listen at a port for an incoming message to the volume. You direct the client application to send a request message to the port to which TCPMonitor listens.

TCPMonitor intercepts the message, displays the SOAP message in the request panel, then resends the message to the volume. When the volume responds, TCPMonitor intercepts the message, displays the SOAP response in the response panel, and resends the message to the client application.

TCPMonitor logs each request-response message pair. You can view a message pair by choosing an entry row in the top panel. You can also remove an entry, edit and resend a message, and save a message pair to a file.

Figure 11-8 shows a request-response message pair in TCPMonitor.

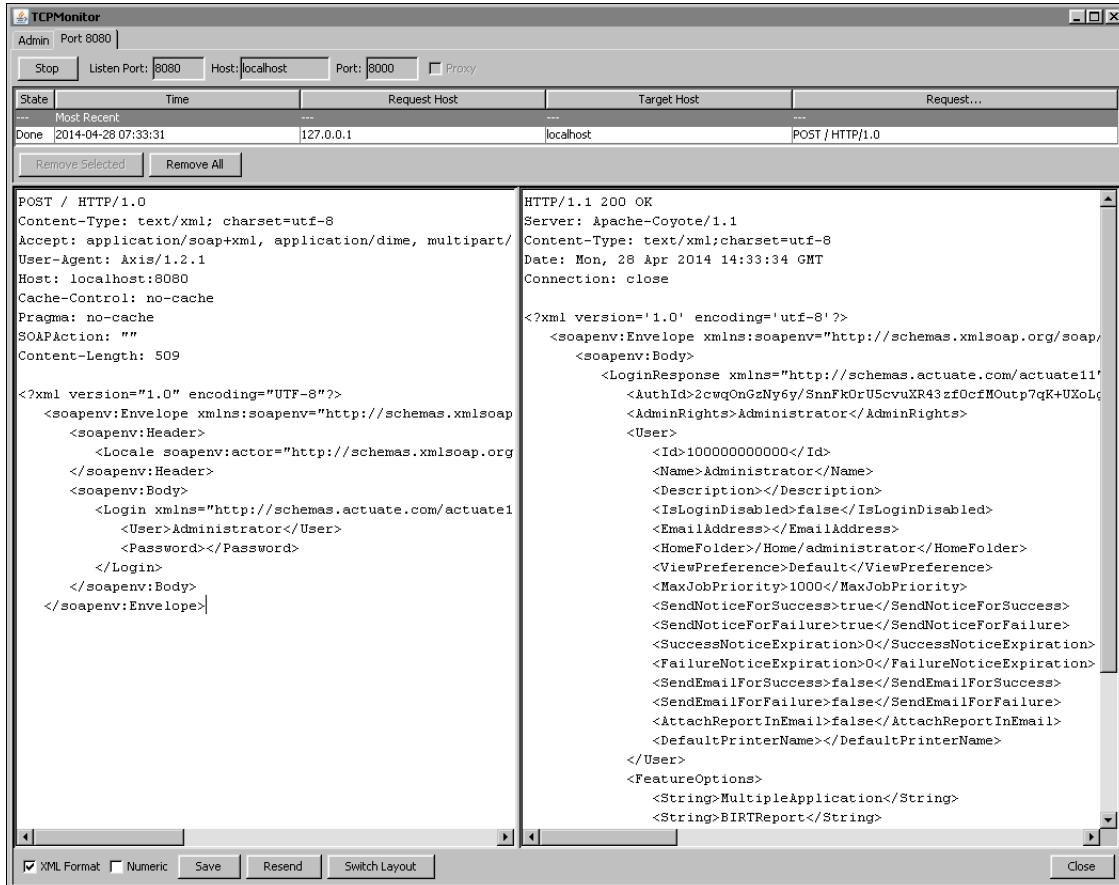


Figure 11-8 TCPMonitor displaying a request-response message pair

You perform the following tasks:

- Configure and run TCPMonitor.
- Redirect a client SOAP request to TCPMonitor.
- View a SOAP message in TCPMonitor.

To complete this tutorial, you need access to the following resources:

- Archive files in \Actuate3\ServerIntTech3\Web Services\Examples\Axis Client\lib, including:
 - com.actuate.schemas (idapi.jar)
 - org.apache.axis.utils.tcpmon.class (axis.jar)
 - Other related JAR files

Task 1: Configure and run TCPMonitor

This task uses the graphical interface of the TCPMonitor utility. This utility is in the org.apache.axis.utils package in axis.jar. You can also run TCPMonitor from the command line using the following syntax:

```
java org.apache.axis.utils.tcpmon [listenPort targetHost  
targetPort]
```

- 1 Open a command prompt window.
- 2 Add the axis client libraries to the CLASSPATH environment variable. At the command prompt, type:

```
set CLASSPATH=C:\Actuate3\ServerIntTech3\Web Services\Examples  
\\Axis Client\lib\*
```

Press Enter.

- 3 At the command prompt, type:

```
java org.apache.axis.utils.tcpmon
```

Press Enter.

TCPMonitor—Admin appears.

- 4 In TCPMonitor—Admin:

- In Listen Port #, type:

8080

- Select Listener.

- In Target Hostname, type:

localhost

- In Target Port #, type:

8000

TCPMonitor—Admin looks like the one in Figure 11-9.

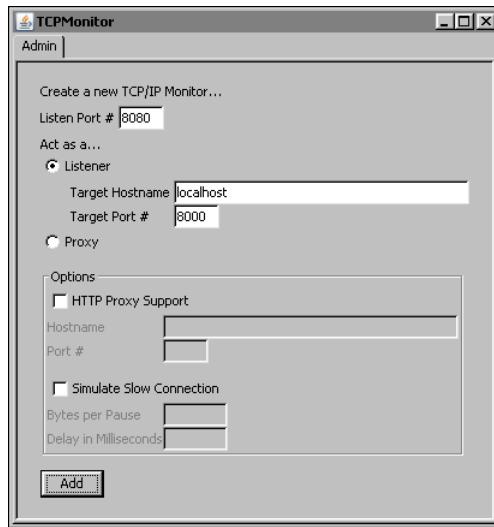


Figure 11-9 TCPMonitor—Admin showing the settings

- 5** Choose Add.

The Port 8080 tab appears.

If a firewall message appears about blocked ports, select the option to unblock the ports.

- 6** Choose Port 8080.

TCPMonitor—Port 8080 appears, as shown in Figure 11-10.

- 7** To view the SOAP messages in indented XML format, select XML Format.
8 To change the split pane layout to a left-right panel view instead of the default, top-bottom panel view, choose Switch Layout.

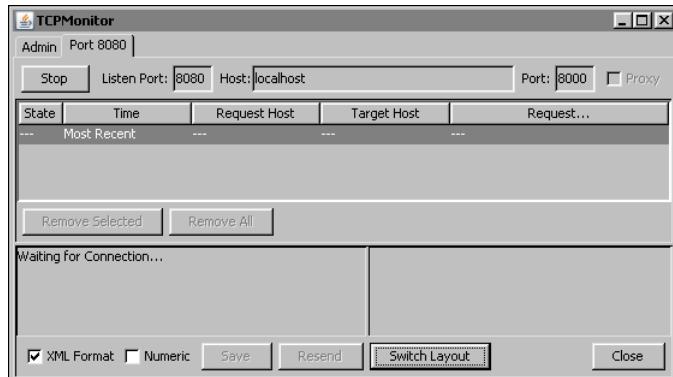


Figure 11-10 TCPMonitor—Port 8080

Task 2: Creating an IDAPI SOAP login application

In this task, you create a Java application to login to iHub using Java that generates a SOAP message. If you have access to the IDAPI training files, you can copy the necessary Java files from the IDAPI student DVD or download package.

- 1 Navigate to or create the \ActuateTraining3\IDAPI\source directory and create a new file called ActuateAPIEx.java. Open the file for editing and create an interface that extends com.actuate.schemas.ActuateAPI and includes the get and set methods for a SOAP header, as shown in the following code:

```
public interface ActuateAPIEx extends
    com.actuate.schemas.ActuateAPI {
    public void setAuthId(java.lang.String authId);
    public void setLocale(java.lang.String locale);
    public void setTargetVolume(java.lang.String targetVolume);
    public void setConnectionHandle(java.lang.String
        connectionHandle);
    public void setDelayFlush(java.lang.Boolean delayFlush);
    public java.lang.String getAuthId();
    public java.lang.String getLocale();
    public java.lang.String getTargetVolume();
    public java.lang.String getConnectionHandle();
    public java.lang.Boolean getDelayFlush();

    // @return Call object with the SOAP Header element set
    public org.apache.axis.client.Call getCall();
}
```

- 2 In the \ActuateTraining3\IDAPI\source directory, create a new file called ActuateAPILocatorEx.java. Open the file for editing and create an interface that extends com.actuate.schemas.ActuateAPILocator and implements the ActuateEx interface to construct the soap header after locating the iHub server, as shown in the following code:

```
import javax.xml.rpc.Call;
import javax.xml.rpc.ServiceException;
import org.apache.axis.message.SOAPHeaderElement;

public class ActuateAPILocatorEx
    extends com.actuate.schemas.ActuateAPILocator
    implements ActuateAPIEx {
    private java.lang.String authId = "";
    private java.lang.String locale = "en_US";
    private java.lang.String targetVolume;
    private java.lang.String targetServer;
    private java.lang.String connectionHandle;
```

```

private java.lang.Boolean delayFlush;
org.apache.axis.client.Call call = null;

/**
 * Constructor for ActuateAPILocatorEx
 */
public ActuateAPILocatorEx() {
    super();
}

/**
 * Create call object and add Actuate's SOAP Header
 */
public Call createCall() throws ServiceException {
    call = (org.apache.axis.client.Call) super.createCall();
    if (null != authId)
        call.addHeader(new SOAPHeaderElement(null, "AuthId",
                                             authId));
    if (null != locale)
        call.addHeader(new SOAPHeaderElement(null, "Locale",
                                             locale));
    if (null != targetVolume)
        call.addHeader(
            new SOAPHeaderElement(null, "TargetVolume",
                                  targetVolume));
    if (null != connectionHandle)
        call.addHeader(
            new SOAPHeaderElement(
                null,
                "ConnectionHandle",
                connectionHandle));
    if (null != targetServer)
        call.addHeader(
            new SOAPHeaderElement(
                null,
                "TargetServer",
                targetServer));
    if (null != targetVolume)
        call.addHeader(
            new SOAPHeaderElement(null, "DelayFlush",
                                  delayFlush));
    return call;
}

/**
 * Returns the authId.
 * @return java.lang.String
 */

```

```
public java.lang.String getAuthId() {
    return authId;
}

/**
 * Get previously created call object
 */
public org.apache.axis.client.Call getCall() {
    if (null == call) {
        try {
            createCall();
        } catch (ServiceException e) {
        }
    }
    return call;
}

/**
 * Returns the connectionHandle.
 * @return byte[]
 */
public String getConnectionHandle() {
    return connectionHandle;
}

/**
 * Returns the delayFlush.
 * @return java.lang.Boolean
 */
public java.lang.Boolean getDelayFlush() {
    return delayFlush;
}

/**
 * Returns the locale.
 * @return java.lang.String
 */
public java.lang.String getLocale() {
    return locale;
}

/**
 * Returns the targetVolume.
 * @return java.lang.String
 */
public java.lang.String getTargetVolume() {
    return targetVolume;
}
```

```

/**
 * Returns the targetServer
 * @return java.lang.String
 */
public java.lang.String getTargetServer(){
    return targetServer;
}

/**
 * Sets the authId.
 * @param authId The authId to set
 */
public void setAuthId(java.lang.String authId){
    this.authId = authId;
}

/**
 * Sets the connectionHandle.
 * @param connectionHandle The connectionHandle to set
 */
public void setConnectionHandle(java.lang.String
    connectionHandle) {
    this.connectionHandle = connectionHandle;
}

/**
 * Sets the delayFlush.
 * @param delayFlush The delayFlush to set
 */
public void setDelayFlush(java.lang.Boolean delayFlush) {
    this.delayFlush = delayFlush;
}

/**
 * Sets the locale.
 * @param locale The locale to set
 */
public void setLocale(java.lang.String locale) {
    this.locale = locale;
}

/**
 * Sets the targetVolume.
 * @param targetVolume The targetVolume to set
 */
public void setTargetVolume(java.lang.String targetVolume) {
    this.targetVolume = targetVolume;
}

```

```

    /**
     * Sets the targetServer
     * @param targetServer The targetServer to set
     */
    public void setTargetServer(java.lang.String targetServer) {
        this.targetServer = targetServer;
    }
}

```

- 3** In the \ActuateTraining3\IDAPI\source directory, create a new file called Arguments.java. Open the file for editing and create a class to parse command line arguments, including the server URL, as shown in the following code:

```

import java.util.Vector;
import java.util.Enumeration;
import java.util.NoSuchElementException;

public class Arguments{
    // default server settings
    String serverURL = null;
    String username = null;
    String password = null;
    boolean embeddedDownload = false;

    Vector arguments = new Vector();
    Enumeration enumeration;

    Arguments(String args[]){
        try {
            // check each argument if it's '-h http://something'
            // then take them as option
            for (int i = 0; i < args.length; i++){
                if (args[i].startsWith("-")){
                    char c = args[i].charAt(1);
                    switch (c){
                        case 'h' :
                        case 'H' :
                            serverURL = args[++i];
                            break;
                        case 'u' :
                        case 'U' :
                            username = args[++i];
                            break;
                        case 'p' :
                        case 'P' :
                            password = args[++i];
                            break;
                    }
                }
            }
        } catch (NoSuchElementException e) {
            e.printStackTrace();
        }
    }
}

```

```

        case 'e' :
        case 'E' :
            embeddedDownload = true;
            break;
        case '?' :
            printUsage();
            System.exit(0);
            break;
    }
} else {
    arguments.add(args[i]);
}
}
enumeration = arguments.elements();
}
catch (Exception e){
    printUsage();
    System.exit(0);
}
}

String getArgument(){
    String result = null;
    try{
        result = (String) enumeration.nextElement();
    }
    catch (NoSuchElementException e){
        printUsage();
        System.exit(0);
    }
    return result;
}

String getOptionalArgument(String defaultValue){
    String result = null;
    try {
        result = (String) enumeration.nextElement();
    }
    catch (NoSuchElementException e){
    }
    return (result != null) ? result : defaultValue;
}

static String usage = "java ClassName [options] ...\\n";

```

```

static String commonUsage =
    "Common options are: \n"+
    "    -h hostname      SOAP endpoint, default 'http://"+
    "        localhost:8000'\n"+
    "    -u username       specify username, default
        'Administrator'\n"+
    "    -p password      specify password, default ''\n"+
    "    -? print this usage\n"+
    "\n";

static void printUsage(){
    System.out.println(usage);
    System.out.println(commonUsage);
}

String getURL(){
    return serverURL;
}

String getUsername(){
    return username;
};

String getPassword(){
    return password;
};

/**
 * Returns the embeddedDownload.
 * @return boolean
 */
public boolean isEmbeddedDownload(){
    return embeddedDownload;
}
}

```

- 4 In the \ActuateTraining3\IDAPI\source directory, create a new file called AcController.java. Open the file for editing and create a class to control the client work flow by calling methods from the Arguments and ActuateEx classes, as shown in the following code:

```

import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.actuate.schemas.*;

```

```

public class AcController implements java.io.Serializable {
    // control variables
    public String authenticationId = null;
    public String username = "Administrator";
    private String password = "";

    // server settings
    public String actuateServerURL = "http://localhost:8000/";

    // proxy operation
    public ActuateSoapBindingStub proxy;
    public ActuateAPIEx actuateAPI;

    /**
     * No-argument Constructor
     */
    public AcController() throws MalformedURLException,
        ServiceException {
        actuateAPI = new ActuateAPILocatorEx();
        setActuateServerURL(actuateServerURL);
    }

    /**
     * Constructor accepting serverURL as a parameter
     */
    public AcController(String serverURL)
        throws MalformedURLException, ServiceException {
        actuateAPI = new ActuateAPILocatorEx();
        setActuateServerURL(serverURL);
    }

    /**
     * Create a new call object that can be used to send SOAP
     * message to actuate server
     * @return Call
     * @throws ServiceException
     */
    public org.apache.axis.client.Call createCall() throws
        ServiceException {
        org.apache.axis.client.Call call =
            (org.apache.axis.client.Call) actuateAPI.createCall();
        call.setTargetEndpointAddress(this.actuateServerURL);
        return call;
    }

    /**
     * Login to actuate server with username and password
     * Return true if login success
     */

```

```

public boolean login() {
    boolean success = true;
    com.actuate.schemas.Login request = new
        com.actuate.schemas.Login();
    request.setPassword(password);
    request.setUser(username);
    try {
        actuateAPI.setAuthId(null);
        com.actuate.schemas.LoginResponse response =
            proxy.login(request);
        authenticationId = response.getAuthId();
        actuateAPI.setAuthId(authenticationId);
    } catch (java.rmi.RemoteException e) {
        // login failed
        success = false;
    }
    return success;
}

/**
 * Sets the ActuateServerURL
 * @param serverURL The actuate server url to set
 */
public void setActuateServerURL(String serverURL)
    throws MalformedURLException, ServiceException {
    if ((proxy == null) ||
        !serverURL.equals(actuateServerURL)) {
        if (serverURL != null)
            actuateServerURL = serverURL;
        System.out.println("Setting server to " +
            actuateServerURL);
        proxy = (ActuateSoapBindingStub)
            actuateAPI.getActuateSoapPort(
                new java.net.URL(actuateServerURL));
    }
}

/**
 * Sets the password.
 * @param password The password to set
 */
public void setPassword(String password) {
    if (password == null)
        password = "";
    this.password = password;
}

```

```

    /**
     * Sets the username.
     * @param username The username to set
     */
    public void setUsername(String username) {
        if (username == null)
            return;
        this.username = username;
    }

    /**
     * Returns the password.
     * @return String
     */
    public String getPassword() {
        return password;
    }

    /**
     * Returns the username.
     * @return String
     */
    public String getUsername() {
        return username;
    }
}

```

- 5 In the \ActuateTraining3\IDAPI\source directory, create a new file called AcLogin.java. Open the file for editing and create a class that uses the IDAPI AcController.login method to connect with iHub and return a message, as shown in the following code:

```

/* This program demonstrates a login to a volume as
Administrator: outputs a success message confirming the login
or, if an error occurs, a usage statement to the console.
*/
import com.actuate.schemas.*;

public class AcLogin {
    public static final String usage = "Usage:\n"+
        "java AcLogin [options]\n";

    public static void main(String[] args) {
        // set command line usage
        Arguments.usage = usage;
        // get command line arguments
        Arguments arguments = new Arguments(args);
        try {
            // login to actuate server

```

```
AcController actuateControl = new
    AcController(arguments.getURL());
actuateControl.setUsername(arguments.getUsername());
actuateControl.setPassword(arguments.getPassword());
if (actuateControl.login()) {
    System.out.println("Congratulations! You have
        successfully logged into Encyclopedia volume as
        "+actuateControl.getUsername());
} else {
    System.out.println("Please try again.\n Usage:\n"+
        "java AcLogin [options]\n");
}
}
catch (Exception e) {
    e.printStackTrace();
}
}
```

6 Save and close all files.

Task 3: Redirect a client SOAP request to TCPMonitor

In this task, you run a Java application from the command line, redirecting the program to send a volume request to the port where TCPMonitor listens. After you type each command at the command prompt, press Enter.

- ## 1 Open another command prompt window.

- 2** At the command prompt, type:

```
cd C:\ActuateTraining3\IDAPI
```

- 3** At the command prompt, type:

```
mkdir build
```

- 4 Add the necessary Axis libraries from your Server Integration Technology installation directory and the new classes to the classpath using a command similar to the following:

```
set CLASSPATH=C:\Actuate3\ServerIntTech3\Web Services  
    \Examples\Axis Client\lib\*;C:\ActuateTraining3  
    \IDAPI;C:\ActuateTraining3\IDAPI\source;C:\ActuateTraining3  
    \IDAPI\build;
```

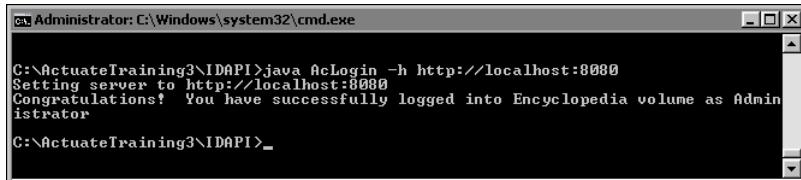
- 5** Compile the specified Java application by typing the following command:

```
javac -d build source/AcLogin.java source  
    /AcController.java source/Arguments.java source  
    /ActuateAPIEx.java source/ActuateAPILocatorEx.java
```

- 6** Run the Java application by typing the following command:

```
java AcLogin -h http://localhost:8080
```

The Java application writes the message shown in Figure 11-11 to the command prompt window when you successfully log in to the volume.



```
C:\ActuateTraining3\IDAPI>java AcLogin -h http://localhost:8080
Setting server to http://localhost:8080
Congratulations! You have successfully logged into Encyclopedia volume as Administrator
C:\ActuateTraining3\IDAPI>-
```

Figure 11-11 Message indicating that the login operation succeeded

Task 4: View a SOAP message in TCPMonitor

In this task, you examine the details of the SOAP request and response messages resulting from a successful volume login operation.

- 1** In the left panel of TCPMonitor, shown in Figure 11-12, examine the following elements of the SOAP login request message:
 - In HTTP header, notice that SOAPAction contains a set of double quotation marks. The volume uses this setting to identify a SOAP message.
 - In the SOAP envelope, examine the SOAP and XML Schema namespace URIs.
 - In the SOAP header, notice the locale tag.
 - In the SOAP body, examine the login tag with references to the <http://schemas.actuate.com/actuate11> namespace, the <User> attribute, and the <Password> attribute.
- 2** In the right panel of TCPMonitor, shown in Figure 11-12, examine the following parts of the body of the SOAP login response message from the volume:
 - Examine the <ACTU>LoginResponse> tag.
 - Examine the <AuthId> tag. A subsequent request for the current session must send this encrypted, authenticated token.
 - Notice the <AdminRights> tag. This tag indicates that the user is the Administrator.
 - Notice the <FeatureOptions> tag. This tag indicates available iHub features as attributes in an eight-element string array.

The screenshot shows the TCPMonitor application interface. At the top, there are buttons for 'Stop', 'Listen Port: 8080', 'Host: localhost', 'Port: 8000', and 'Proxy'. Below this is a table with columns 'State', 'Time', 'Request Host', 'Target Host', and 'Request...'. A single row is listed: 'Done' at '2014-04-28 07:33:31', 'Request Host' is '127.0.0.1', 'Target Host' is 'localhost', and 'Request...' is 'POST / HTTP/1.0'. Below the table are two large text panes. The left pane contains the SOAP request:

```

POST / HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/
User-Agent: Axis/1.2.1
Host: localhost:8080
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 509

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
<soapenv:Header>
<Locale soapenv:actor="http://schemas.xmlsoap.org/
</soapenv:Header>
<soapenv:Body>
<Login xmlns="http://schemas.actuate.com/actuate1
    <User>Administrator</User>
    <Password></Password>
</Login>
</soapenv:Body>
</soapenv:Envelope>

```

The right pane contains the SOAP response:

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Mon, 28 Apr 2014 14:33:34 GMT
Connection: close

<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
<soapenv:Body>
<LoginResponse xmlns="http://schemas.actuate.com/actuate11
    <AuthId>2cwqOnGzNy6y/SnnFkOrU5cvuXR43zf0cfHOutp7qK+UXoLc
    <AdminRights>Administrator</AdminRights>
    <User>
        <Id>100000000000</Id>
        <Name>Administrator</Name>
        <Description></Description>
        <IsLoginDisabled>false</IsLoginDisabled>
        <EmailAddress></EmailAddress>
        <HomeFolder>/Home/administrator</HomeFolder>
        <ViewPreference>Default</ViewPreference>
        <MaxJobPriority>1000</MaxJobPriority>
        <SendNoticeForSuccess>true</SendNoticeForSuccess>
        <SendNoticeForFailure>true</SendNoticeForFailure>
        <SuccessNoticeExpiration>0</SuccessNoticeExpiration>
        <FailureNoticeExpiration>0</FailureNoticeExpiration>
        <SendEmailForSuccess>false</SendEmailForSuccess>
        <SendEmailForFailure>false</SendEmailForFailure>
        <AttachReportInEmail>false</AttachReportInEmail>
        <DefaultPrinterName></DefaultPrinterName>
    </User>
    <FeatureOptions>
        <String>MultipleApplication</String>
        <String>BIRTReport</String>
    </FeatureOptions>

```

At the bottom of the window, there are buttons for 'XML Format' (checked), 'Numeric', 'Save', 'Resend', and 'Switch Layout'. On the far right, there is a 'Close' button.

Figure 11-12 TCPMonitor displaying the SOAP login response message

Writing a simple administration application

A simple administration application typically involves a create operation for one of the following BIRT iHub objects:

- User
- Folder
- UserGroup

To perform an administration operation that acts on an existing object in BIRT iHub System, such as a select, update, or delete operation, you must apply a search condition to the operation. To perform an administration operation that contains a set of administration operations requests, submit the request as a batch or transaction.

The following sections describe the techniques for building an application that performs a simple administration operation that creates a user in the volume.

Creating a user

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. The application class, AcCreateUser, logs in to as Administrator and creates a user.

AcCreateUser.createUser() method performs the following tasks.

- Instantiates a com.actuate.schemas.User object using ActuateControl.newUser() to set the username, password, and home folder

```
public class AcCreateUser {  
    ...  
    public static void createUser(String userName, String  
        homeFolder) throws RemoteException {  
        // create a user with password same as userName  
        User user = actuateControl.newUser(userName,  
            userName, homeFolder);
```

User is a complex data type that represents user attributes.

- Sets additional view preference, notification, e-mail, and job priority options in the User object

```
// Set user view preference to DHTML  
// (defaults to DHTML in standard configuration)  
user.setViewPreference(UserViewPreference.DHTML);  
// set notice options  
user.setSendNoticeForSuccess(new Boolean(true));  
user.setSendNoticeForFailure(new Boolean(true));  
// set email options  
user.setSendEmailForSuccess(new Boolean(true));  
user.setSendEmailForFailure(new Boolean(false));  
// create fake email address UserName@localhost  
user setEmailAddress(userName + "@" + "localhost")  
// assign job priority  
user.setMaxJobPriority(new Long(1000));
```

- Calls ActuateControl.createUser(), passing the reference to the User object

```
// create the user  
actuateControl.createUser(user);  
System.out.println("User " + userName  
    + ", view preferences, send notice and email features,  
    plus job priority privileges created.");  
}  
}
```

About ActuateControl.createUser()

ActuateControl.createUser() performs the following tasks:

- Instantiates a com.actuate.schemas.CreateUser object

```
public com.actuate.schemas.AdministrateResponse
    createUser(com.actuate.schemas.User user)
    throws RemoteException {
    com.actuate.schemas.CreateUser createUser =
        new com.actuate.schemas.CreateUser();
```

The CreateUser operation is only available to a user with the Administrator user group.

- Passes the reference to the User object, containing the username, password, and home folder, and other settings, to the CreateUser object

```
createUser.setUser(user);
```

- Sets IgnoreDup to True

```
createUser.setIgnoreDup(Boolean.TRUE);
```

If the value of IgnoreDup is True, a volume ignores a duplicate request to create the user and does not report an error. If the value of IgnoreDup is False, a volume ignores the duplicate request and reports the error. The default value is False. A volume always rejects a duplicate request regardless of the IgnoreDup setting.

- Instantiates an AdminOperation object

```
com.actuate.schemas.AdminOperation adminOperation =
    new com.actuate.schemas.AdminOperation();
```

An AdminOperation represents a single unit of work within an Administrate request. The list of attributes in the com.actuate.schemas. AdminOperation class lists the possible administration operations that BIRT iHub can perform within the scope of an Actuate Information Delivery API request.

- Passes the reference to the CreateUser object to AdminOperation.setCreateUser()

```
adminOperation.setCreateUser(createUser);
```

- Calls ActuateControl.runAdminOperation(), returning a reference to the AdministrateResponse object that contains the volume response

```
return runAdminOperation(adminOperation);
}
```

```
}
```

About ActuateControl.runAdminOperation()

ActuateControl.runAdminOperation() is an overloaded method that can assemble and run a single administration operation or an array of administration operations. The method that runs a single administration operation performs the following tasks:

- Instantiates an Administrate object

```
public com.actuate.schemas.AdministratResponse  
runAdminOperation(  
    com.actuate.schemas.AdminOperation adminOperation) {  
    com.actuate.schemas.Administrat administrate =  
        new com.actuate.schemas.Administrat();
```

Administrat is not an operation on its own. It is a mechanism for assembling the set of operations that the application is requesting BIRT iHub to perform. An Administrat request can be a composite operation and consist of multiple AdminOperation objects.

- Calls administrate.setAdminOperation() to construct the AdminOperation array, adding the AdminOperation object as an element

```
administrat.setAdminOperation(  
    new com.actuate.schemas.AdminOperation[ ] {  
        adminOperation});
```

The com.actuate.schemas.Administrat object is an array of AdminOperation objects that can create, update, or delete an item or items in the volume. An Actuate Information Delivery API application must submit even a single AdminOperation request in an Administrat object as an array of AdminOperations. The AdminOperation array can contain one or more elements.

- Makes the remote call to the Actuate SOAP port using proxy.Administrat()

```
com.actuate.schemas.AdministratResponse administratResponse =  
    null;  
try {  
    administratResponse = proxy.administrat(administrat);
```

- Handles a RemoteException

```
} catch (java.rmi.RemoteException e) {  
    org.apache.axis.AxisFault l_fault =  
        org.apache.axis.AxisFault.makeFault(e);  
    System.out.println(l_fault.getFaultString());  
    System.out.println(l_fault.getFaultCode().toString());  
    org.w3c.dom.Element[ ] l_details =  
        l_fault.getFaultDetails();  
}
```

- Returns a reference to the com.actuate.schemas.AdministrateResponse object
- ```
return administrateResponse; }
```

---

## Tutorial 16: Writing an application that creates a user

In this tutorial, you build an application that logs in to a volume as Administrator and creates a user. The program demonstrates the following administration operations to create a user:

- Creates a user name and home folder
- Sets view preference, notification, e-mail, and job priority options
- Calls ActuateControl.CreateUser()

Figure 11-13 shows the output to the command prompt window when the user creation operation succeeds.

```
Administrator: C:\Windows\system32\cmd.exe
C:\ActuateTraining3\IDAPI>java AcCreateUser -h http://localhost:8080 testUser /home
Setting server to http://localhost:8080
Creating user testUser
User testUser. view preferences, send notice and email features, plus job priority
privileges created.
C:\ActuateTraining3\IDAPI>
```

**Figure 11-13** Message indicating that the user creation operation succeeded

To complete this tutorial, you need access to the following application classes and related files from Tutorial 15:

- Arguments.java
- ActuateController.java
- ActuateAPIEx.java
- ActuateAPILocatorEx.java

### Task 1: How to write the application that creates a volume user

In this task, you create the AcCreateUser application. If you did not create Arguments.java, ActuateAPIEx.java, and ActuateAPILocatorEx.java in Tutorial 15: “Catching a SOAP message with Axis TCPMonitor,” you must also create these classes.

- 1 Navigate to or create the \ActuateTraining3\IDAPI\source directory. Create a new file called \ActuateTraining11\IDAPI\source\AcCreateUser.java and open the file for editing. Add import statements for com.actuate.schemas and

java.rmi.RemoteException, and create an AcCreateUser class definition and main method, as shown in Listing 11-4.

#### **Listing 11-4**    AcCreateUser

---

```
import java.rmi.RemoteException;
import com.actuate.schemas.*;
public class AcCreateUser {
 // set command line usage
 public static final String usage =
 "Usage:\n"+"
 java AcCreateUser [options] userName homeFolder \n";
 public static ActuateControl actuateControl;public static
 void createUser(String userName, String homeFolder) throws
 RemoteException {
 // create a normal user with password same as userName
 User user = actuateControl.newUser(userName, userName,
 homeFolder);
 // Set user view preference to DHTML (defaults to DHTML in
 // standard configuration)
 // optionally -
 user.setViewPreference(ViewPreference.Default);
 user.setViewPreference(UserViewPreference.DHTML);
 // set notice options
 user.setSendNoticeForSuccess(new Boolean(true));
 user.setSendNoticeForFailure(new Boolean(true));
 // set email options
 user.setSendEmailForSuccess(new Boolean(true));
 user.setSendEmailForFailure(new Boolean(true));
 // create fake email address UserName@localhost
 user setEmailAddress(userName + "@" + "localhost");
 // assign job priority
 user.setMaxJobPriority(new Long(1000));
 // create the user
 actuateControl.createUser(user);
 System.out.println("User " + userName
 + ", view preferences, send notice and email features,
 plus job priority privileges created.");
 }
 public static void main(String[] args) {
 String userName;
 String homeFolder = "/home";
```

```

// set command line usage
Arguments.usage = usage;

// get command line arguments
Arguments arguments = new Arguments(args);
userName = arguments.getArgument();
homeFolder = arguments.getOptionalArgument(homeFolder) +
"/" + userName;

try {
 actuateControl = new ActuateControl(arguments.getURL());

 // Login
 actuateControl.setUsername(arguments.getUsername());
 actuateControl.setPassword(arguments.getPassword());
 actuateControl.login();

 // create user
 createUser(userName, homeFolder);

}
catch (Exception e) {
 e.printStackTrace();
}
}
}
}

```

- To instantiate a User object and use ActuateControl.newUser( ) to set the userName, a password that is the same as the user name, and homeFolder attributes, type:

```
User user = actuateControl.newUser(userName, userName,
homeFolder);
```

- To set the viewer preference, type:

```
user.setViewPreference(UserViewPreference.DHTML);
```

- To set the notification options, type:

```
user.setSendNoticeForSuccess(new Boolean(true));
user.setSendNoticeForFailure(new Boolean(true));
```

- To set the e-mail notification options, type:

```
user.setSendEmailForSuccess(new Boolean(true));
user.setSendEmailForFailure(new Boolean(true));
```

- To set the e-mail address, type:

```
user.setEmailAddress(userName + "@" + "localhost");
```

- 6 To assign the job priority, type:

```
user.setMaxJobPriority(new Long(1000));
```

- 7 To make the call to create the user using ActuateControl.CreateUser(), type:

```
actuateControl.createUser(user);
```

- 8 To create an output message, type:

```
System.out.println("User " + userName + ", view preferences,
send notice and email features, plus job priority
privileges created.");
```

- 2 In the \ActuateTraining3\IDAPI\source directory, open the file called AcController.java, created in Tutorial 15: “Catching a SOAP message with Axis TCPMonitor,” for editing. If AcController.java does not exist, create a new file called AcController.java as instructed in Tutorial 15. Add a method to create a user as shown in Listing 11-5.

#### **Listing 11-5    AcController with a createUser method**

---

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.actuate.schemas.*;

public class AcController implements java.io.Serializable {
 // control variables
 public String authenticationId = null;
 public String username = "Administrator";
 private String password = "";

 // server settings
 public String actuateServerURL = "http://localhost:8000/";

 // proxy operation
 public ActuateSoapBindingStub proxy;
 public ActuateAPIEx actuateAPI;

 /**
 * No-argument Constructor
 */
 public AcController() throws MalformedURLException,
 ServiceException {
 actuateAPI = new ActuateAPILocatorEx();
```

```

 setActuateServerURL(actuateServerURL) ;
 }

 /**
 * Constructor accepting serverURL as a parameter
 */
 public AcController(String serverURL)
 throws MalformedURLException, ServiceException {
 actuateAPI = new ActuateAPILocatorEx();
 setActuateServerURL(serverURL);
 }

 /**
 * Create a new call object that can be used to send SOAP
 * message to actuate server
 * @return Call
 * @throws ServiceException
 */
 public org.apache.axis.client.Call createCall() throws
 ServiceException {
 org.apache.axis.client.Call call =
 (org.apache.axis.client.Call) actuateAPI.createCall();
 call.setTargetEndpointAddress(this.actuateServerURL);
 return call;
 }

 /**
 * Login to actuate server with username and password
 * Return true if login success
 */
 public boolean login() {
 boolean success = true;
 com.actuate.schemas.Login request = new
 com.actuate.schemas.Login();
 request.setPassword(password);
 request.setUser(username);
 try {
 actuateAPI.setAuthId(null);
 com.actuate.schemas.LoginResponse response =
 proxy.login(request);
 authenticationId = response.getAuthId();
 actuateAPI.setAuthId(authenticationId);
 } catch (java.rmi.RemoteException e) {
 // login failed
 success = false;
 }
 return success;
 }
}

```

```

 /**
 * Sets the ActuateServerURL
 * @param serverURL The actuate server url to set
 */
 public void setActuateServerURL(String serverURL)
 throws MalformedURLException, ServiceException {
 if ((proxy == null) ||
 !serverURL.equals(actuateServerURL)) {
 if (serverURL != null)
 actuateServerURL = serverURL;
 System.out.println("Setting server to " +
 actuateServerURL);
 proxy = (ActuateSoapBindingStub)
 actuateAPI.getActuateSoapPort(
 new java.net.URL(actuateServerURL));
 }
 }

 /**
 * Sets the password.
 * @param password The password to set
 */
 public void setPassword(String password) {
 if (password == null)
 password = "";
 this.password = password;
 }

 /**
 * Sets the username.
 * @param username The username to set
 */
 public void setUsername(String username) {
 if (username == null)
 return;
 this.username = username;
 }

 /**
 * Returns the password.
 * @return String
 */
 public String getPassword() {
 return password;
 }
}

```

```

 /**
 * Returns the username.
 * @return String
 */
 public String getUsername() {
 return username;
 }

 /**
 * Send a single administrate message to create a user
 * @param user
 * @return AdministrateResponse
 * @throws RemoteException
 */
 public com.actuate.schemas.AdministratResponse createUser(
 com.actuate.schemas.User user)
 throws RemoteException {
 System.out.println("Creating user " + user.getName());

 com.actuate.schemas.CreateUser createUser =
 new com.actuate.schemas.CreateUser();
 createUser.setUser(user);
 createUser.setIgnoreDup(Boolean.TRUE);

 com.actuate.schemas.AdminOperation adminOperation =
 new com.actuate.schemas.AdminOperation();
 adminOperation.setCreateUser(createUser);

 return runAdminOperation(adminOperation);
 }
}

```

**3** Save and close all files.

## Task 2: How to compile and run the CreateUser application

In this task, you compile and run the CreateUser application from the command prompt. The run command uses the following syntax:

```
java AcCreateUser [options] userName homeFolder
```

- 1** Open a command prompt window.
- 2** At the command prompt, type:

```
cd C:\ActuateTraining11\IDAPI
```

- 3** Add the axis client libraries, build path, and source path to the CLASSPATH environment variable. At the command prompt, type:

```
set CLASSPATH=C:\Actuate3\ServerIntTech3\Web Services\Examples
Axis Client\lib*;C:\Program Files\Actuate11\ServerIntTech
Web Services\Examples\Axis Client\build;C:\Program Files
Actuate11\ServerIntTech\Web Services\Examples\Axis Client
\source;
```

Press Enter.

- 4** To compile the Java application, type:

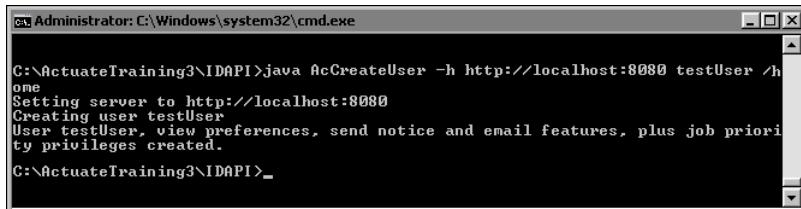
```
javac -d build source/AcCreateUser.java source
/AcController.java
```

If you did not compile Arguments.java, ActuateAPIEx.java, and ActuateAPILocatorEx.java in Tutorial 15: Catching a SOAP message with Axis TCPMonitor, you must also compile these classes.

- 5** To run the Java application, type:

```
java AcCreateUser -h http://localhost:8000 testUser /Home
```

The Java application writes the following messages, shown in Figure 11-14, to the command prompt when a volume login succeeds and the volume creates the user.



```
C:\Administrator:C:\Windows\system32\cmd.exe

C:\ActuateTraining3\IDAPI>java AcCreateUser -h http://localhost:8000 testUser /Home
Setting server to http://localhost:8000
Creating user testUser
User testUser, view preferences, send notice and email features, plus job priority privileges created.
C:\ActuateTraining3\IDAPI>_
```

**Figure 11-14** Message indicating that login succeeds and a new user is created

### Task 3: How to verify the creation of the user

In this task, you use the iHub administration tool to examine the results of the actions you completed in the earlier tasks of this tutorial.

- 1** Open BIRT iHub Visualization Console.
- 2** Log in as Administrator.
- 3** In Visualization Console, choose Administrator→iHub Administration. Users appears.
- 4** Verify that testUser exists in the list of Users, as shown in Figure 11-15.
- 5** Choose testUser. Edit testUser appears.

| Users                                                            |                                                                                | User Groups   |                        | Help |  | administrator |
|------------------------------------------------------------------|--------------------------------------------------------------------------------|---------------|------------------------|------|--|---------------|
|                                                                  |                                                                                |               |                        |      |  |               |
| <h2>Users</h2>                                                   |                                                                                |               |                        |      |  |               |
| <b>USER GROUPS</b><br><input type="text" value="Search Groups"/> | <input type="button" value="Actions"/> <input type="button" value="Add User"/> |               |                        |      |  | 1-4 of 4      |
| All                                                              |                                                                                | Name          | Email                  |      |  |               |
| group1                                                           |                                                                                | Administrator |                        |      |  |               |
| group2                                                           |                                                                                | testUser      | testUser@localhost.com |      |  |               |
| group3                                                           |                                                                                | user1         |                        |      |  |               |
|                                                                  |                                                                                | user2         |                        |      |  |               |

**Figure 11-15** The list of users showing testUser

- 6 Verify that correct values appear in E-mail address and Home folder, as shown in Figure 11-16.

### Edit testUser

|                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>General</b><br>Assign User Groups<br>Assign Licenses | *Name: <input type="text" value="testUser"/><br>Description: <input type="text"/><br>Password: <input type="password"/><br>Confirm Password: <input type="password"/><br>*Email: <input type="text" value="testUser@localhost.com"/><br>Home Folder: <input type="text" value="/Home/testUser"/> Example:/Home/administrator<br>Default Dashboard: <input type="text"/><br>Example:/dashboard/mydashboard.dashboard<br>Language: <input type="button" value="English (United States)"/><br>Time zone: <input type="button" value="Americas/Los Angeles"/> |
|                                                         | <input type="button" value="Cancel"/> <input type="button" value="Save"/>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Figure 11-16** The general property values for testUser

## Performing a search operation

Many operations support acting on one or more items in a volume. To target the items on which to act, you must apply a search condition to the operation. The com.actuate.schemas library contains many special classes for setting a search condition and implementing a search for an item in a volume.

The Information Delivery API provides three sets of parameters that support searching for the data on which an operation acts. Typically, these parameters apply to operations that select, update, move, copy, or delete volume items. The parameters are:

- Id or IdList
- Name or NameList
- Search

### Using com.actuate.schemas.SelectFiles

Use com.actuate.schemas.SelectFiles to retrieve file properties using the ID or name of a single file or folder, or a list of files or folders, in a volume. SelectFiles can recursively search all subfolders in a directory. To restrict a search to the items in a single directory, not including subfolders, use GetFolderItems. SelectFiles does not retrieve file or folder content.

SelectFiles supports three types of searches:

- Use Name or Id to retrieve a single file or folder.
- Use NameList or IdList to retrieve a list of files or folders in a directory.
- Use Search to retrieve all files or folders that match a given condition.

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. In the example application, AcSelectFiles.searchByCondition( ), calls ActuateControl.selectFiles( ), specifying a search condition, and performing the following operations:

- Specifies a search condition for the file type using the wildcard, \*, to target all files that contain the character, R, as the first character in the file extension.

```
public class AcSelectFiles {
 ...
 public static String fileType = "R*";
```

A wildcard is a character used in a search or conditional expression that matches one or more literal characters. Actuate wildcards include the ones in the following list:

- ? matches any single one- or two-byte character.

- # matches any ASCII numeric character [0-9].
- \* matches any number of characters.

The wildcard expression in the example targets files in BIRT iHub such as a BIRT executable file with the file extension RPTDESIGN, and a BIRT document file with the file extension RPTDOCUMENT.

- Instantiates a FileCondition object, setting the condition to match on FileField.FileType using the wildcard expression.

```
public static void searchByCondition() {
 System.out.println("\nThis example demonstrates search by
 file type using search condition:\n");
 com.actuate.schemas.FileCondition fileCondition =
 new com.actuate.schemas.FileCondition();
 fileCondition.setField(com.actuate.schemas.FileField
 .FileType);
 fileCondition.setMatch(fileType);
```

- Instantiates a FileSearch object, setting the search to the properties specified in the FileCondition object.

```
com.actuate.schemas.FileSearch fileSearch =
 new com.actuate.schemas.FileSearch();
fileSearch.setCondition(fileCondition);
```

An application sets the search condition for a file using the FileSearch class. FileSearch is a complex data type that contains the list of properties to specify in a file search condition. An application can specify the search condition for a file using one or more of the following fields:

- Name
- FileType
- Description
- PageCount
- Size
- TimeStamp
- Version
- VersionName
- Owner

Use ArrayOfFileCondition to specify multiple search conditions.

- Sets up the fetch handle to process the results.

```
fileSearch.setFetchSize(new Integer(2));
```

```

System.out.println("Search fileType = " + fileType + "\n");
String fetchHandle = null;
int fetchCount = 1;
while (true) {
 fileSearch.setFetchHandle(fetchHandle);
}

```

- Makes the call to ActuateControl.selectFiles( ), obtaining the reference to the fetch handle from the SelectFilesResponse object.

```

com.actuate.schemas.SelectFilesResponse selectFilesResponse =
 actuateControl.selectFiles(fileSearch, null, null);
fetchHandle = selectFilesResponse.getFetchHandle();

```

A fetch handle indicates that the number of items in the result set exceeds the fetch size limit. A fetch handle returns as a parameter in the response to a Select or Get request, such as SelectFiles or GetFolderItems.

Use the fetch handle to retrieve more results from the result set. In the second and subsequent calls for data, you must specify the same search condition that you used in the original call. All Get and Select requests, except SelectFileType, support the use of a fetch handle.

- Continues processing until there are no more results, printing an appropriate output message.

```

System.out.println("\n Fetch Size = " +
 fileSearch.getFetchSize()
 + "; Fetch Count = " + fetchCount++ + "\n");
if (fetchHandle == null)
 break;
}
}

```

## Using ResultDef

Use a ResultDef parameter to specify the object properties to retrieve from a search. The properties you set in ResultDef depend on the type of item. For example, if the item is a file or folder, you can retrieve the file or folder name, ID, description, date of creation or last update, owner, and version name and number.

ActuateControl.selectFiles( ) specifies and retrieves a list of file properties using a ResultDef parameter by performing the following tasks:

- Sets up the ResultDef String array containing the list of file properties.

```

public com.actuate.schemas.SelectFilesResponse selectFiles(
 com.actuate.schemas.FileSearch fileSearch,
 String name,
 ArrayOfString nameList) {
}

```

```

com.actuate.schemas.ArrayOfString resultDef =
 newArrayOfString(
 new String[] {
 "Description",
 "FileType",
 "Id",
 "Name",
 "Owner",
 "PageCount",
 "Size",
 "TimeStamp",
 "UserPermissions",
 "Version",
 "VersionName" });

```

- Instantiates the SelectFiles object.

```

com.actuate.schemas.SelectFiles selectFiles =
 new com.actuate.schemas.SelectFiles();

```

- Selects files based on a file name, list of file names, or uses com.actuate.schemas.ArrayOfString as a ResultDef parameter to specify the file properties to retrieve.
- ```

selectFiles.setResultDef(resultDef);

■ If not null, sets the reference to either the search object, file name, or list of file
names, depending on which parameter ActuateControl.selectFiles( ) receives.

```

```

if (fileSearch != null)
    selectFiles.setSearch(fileSearch);
else if (name != null)
    selectFiles.setName(name);
else if (nameList != null)
    selectFiles.setNameList(nameList);

```

- Makes the com.actuate.schemas.SelectFiles call using the proxy.

```

com.actuate.schemas.SelectFilesResponse selectFilesResponse =
    null;
try {
    selectFilesResponse = proxy.selectFiles(selectFiles);

```

- Loops through the SelectFileResponse item list, printing the file names and list of properties.

```

com.actuate.schemas.ArrayOfFile itemList =
    selectFilesResponse.getItemList();
com.actuate.schemas.File[ ] files = itemList.getFile();

```

```

        if (files != null) {
            for (int i = 0; i < files.length; i++) {
                printFile(System.out, files[i]);
            }
        }
    } catch (RemoteException e) {
        System.out.println("error !!!");
        e.printStackTrace();
    }
    return selectFilesResponse;
}

```

The Java application writes the messages to a command prompt window when a volume login succeeds and the select files operation completes.

Writing a batch or transaction application

Actuate Information Delivery API supports batch and transaction administration operations. An IDAPI application uses a mixture of developer and com.actuate.schemas classes to implement a batch or transaction administration operation, including:

- Administrate
- AdminOperation
- Transaction
- TransactionOperation

The following sections explain the use of these administration operation classes in detail.

About batch and transaction operations

A batch application submits an array of administration operation requests to a volume using one composite Administrate message. An Administrate request can contain any number of AdminOperation requests in the batch.

An AdminOperation request can contain any number of Transaction requests. A Transaction request is a composite message that can contain any number of TransactionOperation requests. A TransactionOperation represents a single unit of work within a Transaction.

The default level of granularity for a transaction is an object. One operation run against one object is atomic. The use of an explicit Transaction tag in a composite Administrate message expands the transaction boundary to include multiple TransactionOperation requests.

To perform an administration operation that contains a set of requests, submit the request as a batch or transaction within one composite Administrate message. If a

batch request fails, the operations that complete successfully before the failed operation still take effect. If a transaction operation fails, none of the operations in the transaction take effect. BIRT iHub rolls all the work back, leaving the system in the state it was in just prior to the execution of the transaction operation.

Implementing a transaction-based application

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. In the example application, AcAddUsers_Tran.addUsers() performs a transaction-based administration operation to add multiple users. When the command line includes the optional ignoreDup argument, the program ignores errors if a user already exists.

AcAddUsers_Tran.addUsers() performs the following tasks:

- Defines a TransactionOperation array, dimensioning the array to the number of new users.

```
public class AcAddUsers_Tran {  
    ...  
    public static void addUsers( ) throws RemoteException {  
        // set up transaction operation array  
        TransactionOperation[ ] transactionOperations =  
            new TransactionOperation[numberOfUsers];  
  
        System.out.println("\nAdding users... \n");
```

- In addUsers(), within a loop, for each user:

- Instantiates a com.actuate.schemas.User object, setting the user name, password, home folder, and other options such as view preference, notification, and e-mail.

```
// begin loop to create a user transaction operation  
for (int i = 0; i < numberOfUsers; i++) {  
  
    // set User with user name, password, and home folder  
    com.actuate.schemas.User user = new  
        com.actuate.schemas.User();  
    user.setName(userName);  
    user.setPassword(password);  
    user.setHomeFolder(homeFolder);  
    ...
```

- Instantiates the CreateUser object, passing the reference to the current User object and setting IgnoreDup.

```
// create user  
com.actuate.schemas.CreateUser createUser =  
    new com.actuate.schemas.CreateUser( );
```

- ```

createUser.setUser(user);
createUser.setIgnoreDup(ignoreDup);

■ Instantiates an AdminOperation object, passing the reference to the
CreateUser object.

// set Administration operation for current user
com.actuate.schemas.AdminOperation adminOperation =
 new com.actuate.schemas.AdminOperation();
adminOperation.setCreateUser(createUser);

■ Instantiates a TransactionOperation object, passing the reference to the
current User object to complete the set up of the transaction operation.

// set transaction operation for current user
TransactionOperation transactionOperation =
 new TransactionOperation();
transactionOperation.setCreateUser(createUser);

■ Adds the reference to the current TransactionOperation object to the
TransactionOperation array.

 transactionOperations[i] = transactionOperation;
}

■ After the end of loop, instantiates a Transaction object, passing the reference to
the TransactionOperation array that contains the details of all the create user
operations.

// set up the transaction with the transaction operations
Transaction transaction = new Transaction();
transaction.setTransactionOperation(transactionOperations);

■ Instantiates an AdminOperation object, passing the reference to the
Transaction object to create the composite administration operation.

// set up Administration operation
AdminOperation adminOperation = new AdminOperation();
adminOperation.setTransaction(transaction);

■ Calls ActuateControl.runAdminOperation(), passing the reference to the
composite administration operation, and returning a reference to the
AdministateResponse object that contains the volume response.

// run Administration operation
if (null == actuateControl.runAdminOperation(adminOperation)) {

■ Prints an output message, indicating the success or failure of the transaction,
depending on whether the reference to the AdministateResponse object is
null or valid.

```

```

 System.out.println("Create user transaction
 failed.\n");
 }
 else {
 System.out.println("Create user transaction succeeded.
 \n");
 }
}

```

## Uploading a file

To upload a file to a volume, you must identify the file to upload and the volume in which to place the file. You can also specify how to work with existing versions of the file you upload. Using Actuate's open server technology, you can upload third-party file types and native Actuate file types.

### About ways of uploading a file

When you upload a file, the content streams to the volume. You can stream a report with a SOAP message in two ways:

- Embed the file in the response.

In embedding a file, the application specifies the ContentLength in the HTTP header. If you use HTTP 1.0, you typically choose to embed the file.

- Send the file as a MIME attachment.

A MIME attachment transmits the contents of the file outside the boundary of the SOAP message.

A SOAP message with a MIME attachment consists of three parts:

- HTTP header
- Actuate SOAP message
- File attachment

The following example uses a MIME attachment and relevant Actuate Information Delivery API classes to build an application that uploads a file to a volume.

### Using com.actuate.schemas.UploadFile

Use the com.actuate.schemas.UploadFile class to upload a file to a volume. The file content streams to BIRT iHub as unchunked MIME attachment to the request. The UploadFile class contains the following attributes:

- NewFile is the com.actuate.schemas.NewFile object to upload.

- CopyFromLatestVersion is an array of Strings used to copy one or more of the following properties from the latest version of the file, if one exists, to the version of the file that you upload:
  - Description is a description of the file.
  - Permissions, the access control list (ACL) specifying the users and user groups that can access the file.
  - ArchiveRule specifies the autoarchive rules for the file, which determine how BIRT iHub ages the file and when the file expires.
- Content is the com.actuate.schemas.Attachment object that specifies the content Id, content type, content length, content encoding, locale, and content data.

## How to build an application that uploads a file

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. In this example application, AcUploadFile.uploadFile( ) performs the following operations:

- Instantiates a NewFile object, then sets the volume file location and name

```
public class AcUploadFile {
 ...
 public static void uploadFile (String locFileName, String
 encycFileName) throws RemoteException,
 ServiceException {
 // set new file information

 com.actuate.schemas.NewFile newFile =
 new com.actuate.schemas.NewFile();
 newFile.setName(encycFileName);
```

- Instantiates the Attachment object, then sets contentID attribute to the file location and name

```
// set MIME content ID (must be same as AttachmentPart)

com.actuate.schemas.Attachment content =
 new com.actuate.schemas.Attachment();
content.setContentId(locFileName);
```

- Instantiates the UploadFile object, then passes the references to the NewFile and Attachment objects

```
// set upload message

com.actuate.schemas.UploadFile request =
 new com.actuate.schemas.UploadFile();
request.setNewFile(newFile);
request.setContent(content);
```

- Sets up the data handler to read the file

```
// use input file as data source
javax.activation.DataHandler dhSource =
 new javax.activation.DataHandler(new
 javax.activation.FileDataSource(locFileName));
```
- Instantiates the org.apache.axis.attachments.AttachmentPart object to contain the data, passes the reference to the data handler, then sets contentID attribute to the file location and name

```
// set attachment in call
org.apache.axis.attachments.AttachmentPart attachmentPart =
 new org.apache.axis.attachments.AttachmentPart();
attachmentPart.setDataHandler(dhSource);
attachmentPart.setContentId(locFileName);
```
- Performs the UploadFile administration operation by making a call to ActuateControl.uploadFile()

```
// call upload file
com.actuate.schemas.UploadFileResponse response =
 actuateControl.uploadFile(request, attachmentPart);
}
```

ActuateControl.uploadFile() performs the following operations:

- Sets up org.apache.axis.client.Call object:
  - Obtains a reference to the Call object using createCall()

```
public com.actuate.schemas.UploadFileResponse uploadFile(
 com.actuate.schemas.UploadFile request,
 org.apache.axis.attachments.AttachmentPart
 attachmentPart)
throws RemoteException, RemoteException,
 ServiceException {
 org.apache.axis.client.Call call = createCall();
```
  - Calls addParameter() specifying the following UploadFile parameters:
    - Parameter name
    - XML datatype for the parameter
    - Java class for the parameter
    - Parameter mode, indicating whether it is ParameterMode.IN, OUT or INOUT

```

call.addParameter(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "UploadFile"),
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "UploadFile"),
 com.actuate.schemas.UploadFile.class,
 javax.xml.rpc.ParameterMode.IN);

```

- Sets the return type for the operation by specifying parameters that indicate the XML data type and Java class for UploadFileResponse

```

call.setReturnType(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "UploadFileResponse"));

```

- Sets the UseSOAPAction and SOAPAction URI parameters

```

call.setUseSOAPAction(true);
call.setSOAPActionURI("");

```

- Sets the encoding style URI to null to use the default, binary

```
call.setEncodingStyle(null);
```

- Sets org.apache.axis.AxisEngine.PROP\_DOMULTIREFS to False

```

call.setProperty(
 org.apache.axis.AxisEngine.PROP_DOMULTIREFS,
 Boolean.FALSE);

```

PROP\_DOMULTIREFS controls the serialization and deserialization processes and the way the client engine handles complex type parameters with multiple references.

- Sets org.apache.axis.client.Call.SEND\_TYPE\_ATTR to False

```

call.setProperty(
 org.apache.axis.client.Call.SEND_TYPE_ATTR,
 Boolean.FALSE);

```

SEND\_TYPE\_ATTR controls whether to send xsi type attributes.

- Sets the operation style to document

```
call.setOperationStyle("document");
```

- Converts the operation name String to a QName

```

call.setOperationName(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "UploadFile"));

■ Adds the MIME attachment

 // set the actual MIME attachment
 call.addAttachmentPart(attachmentPart);

■ Makes the call, passing the reference to the UploadFile request in an Object
array as required by the Apache Axis framework

Object resp = call.invoke(new Object[] { request });

■ If it occurs, handles a RemoteException, otherwise, casts the response into an
UploadFileResponse object and returns it to AcUploadFile.uploadFile()

if (resp instanceof java.rmi.RemoteException) {
 throw (java.rmi.RemoteException) resp;
}
else {
 try {
 return (com.actuate.schemas.UploadFileResponse) resp;
 }
 catch (java.lang.Exception e) {
 return (com.actuate.schemas.UploadFileResponse) org
 .apache
 .axis
 .utils
 .JavaUtils
 .convert(resp,
 com.actuate.schemas.UploadFileResponse.class);
 }
}
}

```

For more information about the Apache Axis 2 client Java-based framework for implementing a SOAP processor, see <http://ws.apache.org/axis/java>.

## Tutorial 17: Writing an application that uploads a file

In this tutorial, you write an application that uploads a file to an iHub volume. When you complete this tutorial, the file appears in the reports folder, as shown in Figure 11-17.

| Name               | Type        | Version # | Version Name | Size   | Modified          | Pages |
|--------------------|-------------|-----------|--------------|--------|-------------------|-------|
| Employee_Directory | BIRT Design | 3         |              | 136 KB | 7/24/2014 3:29 PM |       |

**Figure 11-17** The reports folder showing the finished file from this tutorial

To complete this tutorial, you need access to the following application classes and related files from Tutorial 15:

- Arguments.java
- AcController.java
- ActuateAPIEx.java
- ActuateAPILocatorEx.java

## Task 1: How to upload a file

In this task, you create a Java class, AcUploadFile, that uploads a file to an iHub volume, receives a response, and prints the name of the uploaded file. If you did not create Arguments.java, ActuateAPIEx.java, and ActuateAPILocatorEx.java in Tutorial 15: Catching a SOAP message with Axis TCPMonitor, you must also create these classes as well.

- 1 Navigate to or create the \ActuateTraining3\IDAPI\source directory. Create a new file called \ActuateTraining11\IDAPI\source\AcUploadFile.java and open the file for editing. Add import statements for javax.xml.rpc.ServiceException, java.io.File, and java.rmi.RemoteException, add an AcUploadFile class definition, and create the uploadClass and main methods, as shown in Listing 11-6.

**Listing 11-6** The AcUploadFile class

---

```

import javax.xml.rpc.ServiceException;
import java.io.File;
import java.rmi.RemoteException;

public class AcUploadFile {
 public static String usage =
 "java AcUploadFile [options] localFileName
 [encyclopediaFileName]\n"
 + "\n"
 + "Upload a file to the server\n";
 public static ActuateControl actuateControl;

```

```

/**
 * This example uploads a file to Actuate iHub volume
 * @throws RemoteException
 */
public static void uploadFile (String locFileName, String
 encycFileName) throws RemoteException, ServiceException {
 System.out.println("Uploading file: " + locFileName);
 System.out.println("As file : " + encycFileName);
 // set new file information
 com.actuate.schemas.NewFile newFile = new
 com.actuate.schemas.NewFile();
 newFile.setName(encycFileName);
 // set MIME content id (must be same as AttachmentPart)
 com.actuate.schemas.Attachment content = new
 com.actuate.schemas.Attachment();
 content.setContentId(locFileName);
 // set upload message
 com.actuate.schemas.UploadFile request = new
 com.actuate.schemas.UploadFile();
 request.setNewFile(newFile);
 request.setContent(content);
 // use input file as data source
 javax.activation.DataHandler dhSource = new
 javax.activation.DataHandler(new
 javax.activation.FileDataSource(locFileName));
 // set attachment in call
 org.apache.axis.attachments.AttachmentPart attachmentPart
 = new org.apache.axis.attachments.AttachmentPart();
 attachmentPart.setDataHandler(dhSource);
 attachmentPart.setContentId(locFileName);
 // call upload file
 com.actuate.schemas.UploadFileResponse response =
 actuateControl.uploadFile(request, attachmentPart);
 // print out the file id of the encyclopedia file
 System.out.println("Uploaded " + locFileName + " with
 fileid: " + response.getFileId());
}

public static void main(String[] args) {
 // upload settings
 String localFileName;
 String encyclopediaFileName;
 Arguments usage = usage;
 // get command line arguments
 Arguments arguments = new Arguments(args);
 localFileName = arguments.getArgument();
 if (!new File(localFileName).exists()){
 System.out.println("File not found: " + localFileName);
}

```

```
 return;
 }
 // get optional parameter
 encyclopediaFileName = arguments.getOptionalArgument(new
 File(localFileName).getName());
 try {
 actuateControl = new ActuateControl(arguments.getURL());
 // login to actuate server
 actuateControl.setUsername(arguments.getUsername());
 actuateControl.setPassword(arguments.getPassword());
 actuateControl.setVolume(arguments.getVolume());
 actuateControl.login();
 uploadFile(localFileName,encyclopediaFileName);
 }
 catch (Exception e) {
 e.printStackTrace();
 }
}
```

- 1 To instantiate the NewFile object, after the following line:

```
// set new file information
```

type:

```
com.actuate.schemas.NewFile newFile =
new com.actuate.schemas.NewFile();
newFile.setName(encycFileName);
```

- 2 To instantiate the Attachment object and specify the file name, after the following line:

```
// set MIME content id (must be same as AttachmentPart)
```

type:

```
com.actuate.schemas.Attachment content =
new com.actuate.schemas.Attachment();
content.setContentId(locFileName);
```

- 3 To instantiate the UploadFile object and specify the name and location of the file, after the following line:

```
// set upload message
```

type:

```
com.actuate.schemas.UploadFile request =
new com.actuate.schemas.UploadFile();
```

```
request.setNewFile(newFile);
request.setContent(content);
```

- 4 To set up the data handler using the input file as the data source, after the following line:

```
// use input file as data source
```

type:

```
javax.activation.DataHandler dhSource =
new javax.activation.DataHandler(new
javax.activation.FileDataSource(locFileName));
```

- 5 To embed the attachment in the response, after the following line:

```
// set attachment in call
```

type:

```
org.apache.axis.attachments.AttachmentPart attachmentPart =
new org.apache.axis.attachments.AttachmentPart();
attachmentPart.setDataHandler(dhSource);
attachmentPart.setContentId(locFileName);
```

- 6 To upload the file you receive in an Encyclopedia volume response, after the following line:

```
// call upload file
```

type:

```
com.actuate.schemas.UploadFileResponse response =
actuateControl.uploadFile(request, attachmentPart);
```

- 7 To print the name of the uploaded file, after the following line:

```
// print out the file id of the encyclopedia file
```

type:

```
System.out.println("Uploaded " + locFileName + " with fileId:
" + response.getFileId());
```

- 2 In the \ActuateTraining3\IDAPI\source directory, open the file called AcController.java, created in Tutorial 15: Catching a SOAP message with Axis TCPMonitor, for editing. If AcController.java does not exist, create a new file called AcController.java as instructed in Tutorial 15. Add a method to upload a file as shown in Listing 11-7.

#### **Listing 11-7     AcController with a uploadFile method**

---

```
import java.io.File;
```

```

import java.io.FileOutputStream;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.actuate.schemas.*;

public class AcController implements java.io.Serializable {
 // control variables
 public String authenticationId = null;
 public String username = "Administrator";
 private String password = "";

 // server settings
 public String actuateServerURL = "http://localhost:8000/";

 // proxy operation
 public ActuateSoapBindingStub proxy;
 public ActuateAPIEx actuateAPI;

 /**
 * No-argument Constructor
 */
 public AcController() throws MalformedURLException,
 ServiceException {
 actuateAPI = new ActuateAPILocatorEx();
 setActuateServerURL(actuateServerURL);
 }

 /**
 * Constructor accepting serverURL as a parameter
 */
 public AcController(String serverURL)
 throws MalformedURLException, ServiceException {
 actuateAPI = new ActuateAPILocatorEx();
 setActuateServerURL(serverURL);
 }

 /**
 * Create a new call object that can be used to send SOAP
 * message to actuate server
 * @return Call
 * @throws ServiceException
 */
 public org.apache.axis.client.Call createCall() throws
 ServiceException {
 org.apache.axis.client.Call call =
 (org.apache.axis.client.Call) actuateAPI.createCall();
 call.setTargetEndpointAddress(this.actuateServerURL);
 }
}

```

```

 return call;
 }

 /**
 * Login to actuate server with username and password
 * Return true if login success
 */
 public boolean login() {
 boolean success = true;
 com.actuate.schemas.Login request = new
 com.actuate.schemas.Login();
 request.setPassword(password);
 request.setUser(username);
 try {
 actuateAPI.setAuthId(null);
 com.actuate.schemas.LoginResponse response =
 proxy.login(request);
 authenticationId = response.getAuthId();
 actuateAPI.setAuthId(authenticationId);
 } catch (java.rmi.RemoteException e) {
 // login failed
 success = false;
 }
 return success;
 }

 /**
 * Sets the ActuateServerURL
 * @param serverURL The actuate server url to set
 */
 public void setActuateServerURL(String serverURL)
 throws MalformedURLException, ServiceException {
 if ((proxy == null) ||
 !serverURL.equals(actuateServerURL)) {
 if (serverURL != null)
 actuateServerURL = serverURL;
 System.out.println("Setting server to " +
 actuateServerURL);
 proxy = (ActuateSoapBindingStub)
 actuateAPI.getActuateSoapPort(
 new java.net.URL(actuateServerURL));
 }
 }

 /**
 * Sets the password.
 * @param password The password to set
 */
 public void setPassword(String password) {

```

```

 if (password == null)
 password = "";
 this.password = password;
 }

 /**
 * Sets the username.
 * @param username The username to set
 */
 public void setUsername(String username) {
 if (username == null)
 return;
 this.username = username;
 }

 /**
 * Returns the password.
 * @return String
 */
 public String getPassword() {
 return password;
 }

 /**
 * Returns the username.
 * @return String
 */
 public String getUsername() {
 return username;
 }

 public com.actuate.schemas.UploadFileResponse uploadFile(
 com.actuate.schemas.UploadFile request,
 org.apache.axis.attachments.AttachmentPart attachmentPart)
 throws RemoteException, RemoteException, ServiceException
 {
 org.apache.axis.client.Call call = createCall();

 call.addParameter(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate10",
 "UploadFile"),
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate10",
 "UploadFile"),
 com.actuate.schemas.UploadFile.class,
 javax.xml.rpc.ParameterMode.IN);

 call.setReturnType(
 new javax.xml.namespace.QName(

```

```
 "http://schemas.actuate.com/actuate10",
 "UploadFileResponse"));

call.setUseSOAPAction(true);
call.setSOAPActionURI("");
call.setEncodingStyle(null);
call.setProperty(
 org.apache.axis.AxisEngine.PROP_DOMULTIREFS,
 Boolean.FALSE);
call.setProperty(
 org.apache.axis.client.Call.SEND_TYPE_ATTR,
 Boolean.FALSE);
call.setOperationStyle("document");
call.setOperationName(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate10",
 "UploadFile"));

// set the actual MIME attachment
call.addAttachmentPart(attachmentPart);

Object resp = call.invoke(new Object[] { request });

if (resp instanceof java.rmi.RemoteException) {
 throw (java.rmi.RemoteException) resp;
} else {
 try {
 return (com.actuate.schemas.UploadFileResponse) resp;
 } catch (java.lang.Exception e) {
 return (com.actuate.schemas.UploadFileResponse)
 org.apache.axis.utils.JavaUtils.convert(resp,
 com.actuate.schemas.UploadFileResponse.class);
 }
}
}
```

## **Task 2: How to compile and run the application to**

## **upload a file**

uses the following syntax:

```
java AcUploadFile -h http://localhost:8000 <localFileName>
```

## 1 Open a command prompt window

- 2** At the command prompt, type:

```
cd C:\ActuateTraining11\IDAPI
```

- 3** Add the axis client libraries, build path, and source path to the CLASSPATH environment variable. At the command prompt, type:

```
set CLASSPATH=C:\Actuate3\ServerIntTech3\Web Services\Examples
\Axis Client\lib*;C:\Program Files\Actuate11\ServerIntTech
\Web Services\Examples\Axis Client\build;C:\Program Files
\Actuate11\ServerIntTech\Web Services\Examples\Axis Client
\source;
```

Press Enter.

- 4** To compile the Java application, type:

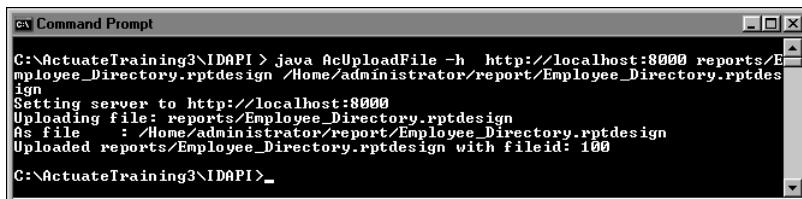
```
javac -d build source/AcUploadFile.javasource/AcController.java
```

If you did not compile Arguments.java, ActuateAPIEx.java, and ActuateAPILocatorEx.java in Tutorial 15: Catching a SOAP message with Axis TCPMonitor, you must also compile these classes.

- 5** To run the Java application, type:

```
java AcUploadFile -h http://localhost:8000 reports
/Employee_Directory.rptdesign /Home/administrator
/report/Employee_Directory.rptdesign
```

The Java application writes the following messages, shown in Figure 11-18, to the command prompt window when an Encyclopedia volume login succeeds and the file upload occurs.



The screenshot shows a Windows Command Prompt window titled 'Command Prompt'. The window contains the following text output:

```
C:\ActuateTraining3\IDAPI > java AcUploadFile -h http://localhost:8000 reports/Employee_Directory.rptdesign /Home/administrator/report/Employee_Directory.rptdesign
Setting server to http://localhost:8000
Uploading file: reports/Employee_Directory.rptdesign
As file : /Home/administrator/report/Employee_Directory.rptdesign
Uploaded reports/Employee_Directory.rptdesign with fileid: 100
C:\ActuateTraining3\IDAPI>_
```

**Figure 11-18** Message indicating that login succeeds and a file is uploaded

#### How to verify the file upload

In this task, you use the Visualization Platform to view the results of the upload file operation.

- 1** Open iHub Visualization Platform.

- 2** Log in as Administrator.

The files and folders belonging to administrator appear.

- 3** Choose /report.

- 4 Verify that /report contains the new version of Employee\_Directory, as shown in Figure 11-19.

| Name               | Type        | Version # | Version Name | Size   | Modified          | Pages |
|--------------------|-------------|-----------|--------------|--------|-------------------|-------|
| Employee_Directory | BIRT Design | 3         |              | 136 KB | 7/24/2014 3:29 PM |       |

**Figure 11-19** The contents of the report folder showing Employee\_Directory

## Downloading a file

To download a file from a volume, identify the file and indicate whether to embed the content in the response or use chunked transfer encoding. In HTTP 1.0, you must embed the entire file in the response and send it in a long, uninterrupted file stream. In HTTP 1.1, you can send the file in smaller chunks, which increases the efficiency of the file transfer. Although the volume supports both methods, BIRT iHub messages typically use chunked transfer encoding.

The following example application uses chunked transfer encoding and relevant com.actuate.schemas classes to build an application that downloads a file from a volume.

### Using com.actuate.schemas.DownloadFile

The com.actuate.schemas.DownloadFile class downloads a file from a volume to the client. You can choose to embed the file content in the response or send it to the client as an attachment.

The DownloadFile class contains the following list of attributes:

- FileName or FileId is a String specifying the ID or name of the file to download.  
Specify either FileId or FileName.
- DecomposeCompoundDocument is a Boolean indicating whether to download a compound document as one file or multiple attachments.  
If the DecomposeCompoundDocument value is False, you can download the file as a single file. If the value is True, and the file is a compound document, the volume splits the file into attachments containing the atomic elements of the compound document such as fonts and images. A decomposed document is not in a viewable format. The default value is False.
- DownloadEmbedded is a Boolean indicating whether to embed the content in the response or use chunked transfer encoding.

- FileProperties is a String array specifying the file properties to return.

## How to build an application that downloads a file

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. In the example application, AcDownloadFile\_Chunked class, downloads a file from the volume, using the chunked option, and saves the file in the user `~\temp` directory.

The AcDownloadFile\_Chunked class performs the following operations:

- Instantiates an org.apache.axis.client.Service object.

```
public class AcDownloadFile_Chunked {
 Service service = new Service();
```

- In AcDownloadFile\_Chunked.main(), defines an attribute for the file name and initializes the downloadEmbedded flag to False.

```
public static void main(String[] args) {
 // download settings
 String filename;
 Boolean downloadEmbedded = Boolean.FALSE;
```

- Gets the command line argument, specifying the file name.

```
Arguments arguments = new Arguments(args);
filename = arguments.getArgument();
```

- Creates a client Call object, using ActuateControl.createCall().

```
try {
 // create a call object
 org.apache.axis.client.Call call =
 actuateControl.createCall();
```

The ActuateControl.createCall() method performs the following tasks:

- Uses ActuateAPI.createCall() to create a client Call object that can send a SOAP message to BIRT iHub
- Casts the Call object as an org.apache.axis.client.Call object as required by the Apache Axis framework
- Sets the target BIRT iHub URL
- Returns the Call object to AcDownloadFile\_Chunked.main()

The following example shows the code for ActuateControl.createCall():

```
public org.apache.axis.client.Call createCall() throws
 ServiceException {
 org.apache.axis.client.Call call =
 (org.apache.axis.client.Call) actuateAPI.createCall();
```

```

 call.setTargetEndpointAddress(this.actuateServerURL);
 return call;
 }
}

```

- AcDownloadFile\_Chunked.main() sets up the Call parameters using a local method, prepareDownloadFileCall(), passing in the reference to the Call object.

```
prepareDownloadFileCall(call);
```

The prepareDownloadFileCall() method sets up the org.apache.axis.client.Call parameters as shown in the following code:

```

public static void prepareDownloadFileCall(
 org.apache.axis.client.Call call) {
 call.addParameter(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "DownloadFile"),
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "DownloadFile"),
 com.actuate.schemas.DownloadFile.class,
 javax.xml.rpc.ParameterMode.IN);
 call.setReturnType(
 new javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "DownloadFileResponse"));
 call.setUseSOAPAction(true);
 call.setSOAPActionURI("");
 call.setEncodingStyle(null);
 call.setProperty(
 org.apache.axis.AxisEngine.PROP_DOMULTIREFS,
 Boolean.FALSE);
 call.setProperty(snew javax.xml.namespace.QName(
 "http://schemas.actuate.com/actuate11",
 "DownloadFile"));
}

```

- Next, AcDownloadFile\_Chunked.main() sets up the DownloadFile request object, specifying the downloadEmbedded flag and file name.

```

// set download message
com.actuate.schemas.DownloadFile request =
 new com.actuate.schemas.DownloadFile();
request.setDownloadEmbedded(downloadEmbedded);
request.setFileName(filename);

```

- Invokes the Call object, passing the reference to the DownloadFile request in an Object array.

```
Object resp = call.invoke(new Object[] {request});
```

- Uses org.apache.axis.client.Call.getMessageContext( ) to obtain the reference to the org.apache.axis.MessageContext object.

```
MessageContext messageContext = call.getMessageContext();
```

- Uses MessageContext.getResponseMessage( ) to obtain the response message.

```
Message message = messageContext.getResponseMessage();
```

- Uses an Iterator to get the attachments and stream the attachment parts to the Axis default location for the downloaded file in user ~/temp directory.

```
Iterator iterator = message.getAttachments();
while (iterator.hasNext()) {
 AttachmentPart attachmentPart =
 (AttachmentPart) iterator.next();
 try {
 filename =
 attachmentPart.getDataHandler().getName();
 System.out.println("Attachment saved at " +
 filename);
 }
 catch (SOAPException e) {
 e.printStackTrace();
 }
}
catch (Exception e) {
 e.printStackTrace();
}
```

The Java application writes the messages to the command prompt window when the download succeeds. The path and file name in the output message provide the file name and location information for the attachment.

## Executing a report

An ExecuteReport operation generates a synchronous report from an executable file. The executable file can be an Actuate BIRT file type or an external executable file type.

An ExecuteReport request identifies the input file. To save the output, you indicate the output file's destination by including a full path in the Name parameter for RequestedOutputFile.

An ExecuteReportResponse returns a volume-generated ObjectId and the status of the report. For a persistent report, the ObjectId is valid until a user deletes the report. For a transient report, the ObjectId is a temporary identifier that lasts for a configurable period of time.

ExecuteReportResponse also returns a ConnectionHandle for a persistent report. Subsequent requests for the same report must use this ConnectionHandle. The ConnectionHandle remains valid throughout the session.

## Understanding the structure of an ExecuteReport application

An ExecuteReport application typically uses a mix of developer and com.actuate.schemas classes to implement an ExecuteReport operation in a volume. The following application derives from the code in the BIRT iHub Integration Technology example applications for the Apache Axis 2 client. In the example application, AcExecuteReport, performs these tasks:

- Instantiates the Arguments class, gets the input file (.rptdesign) and output file (.rptdocument) names as command line arguments, and passes these command line arguments to the constructor
- Instantiates the ActuateController class, getting a server URL from Arguments, if specified, at the command line
- Uses methods defined in the controller class to send a request to execute a report using the com.actuate.schemas.ExecuteReport and ExecuteReportResponse classes

The AcExecuteReport class looks like the following example:

```
public class AcExecuteReport {
 public static ActuateControl actuateControl;
 public static void main(String[] args) {
 // download settings
 String inputFileName;
 String outputFileName;
 Arguments arguments = new Arguments(args);
 inputFileName = arguments.getArgument();
 outputFileName = arguments.getArgument();
 try {
 actuateControl =
 new ActuateControl(arguments.getURL());
 ...
 actuateControl.setInputFileName(inputFileName);
 actuateControl.setOutputFileName(outputFileName);
 actuateControl.executeReport();
 }
 }
}
```

```

 }
 catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

`ActuateControl.executeReport()` performs the following tasks:

- Sets up an `ExecuteReport` object, specifying the job name, input file name, and output file flag
- Sets up a `NewFile` object to receive the requested output file
- Calls the proxy to submit the request and receives the response
- Outputs a status message

`ActuateControl.executeReport()` looks like the following example:

```

public void executeReport() throws RemoteException {
 com.actuate.schemas.ExecuteReport executeReport =
 new com.actuate.schemas.ExecuteReport();
 executeReport.setJobName(jobName);
 executeReport.setInputFileName(inputFileName);
 boolean bSaveOutputFile = (!outputFileName.equals(" "));
 executeReport.setSaveOutputFile(
 new Boolean(bSaveOutputFile));
 if (bSaveOutputFile) {
 com.actuate.schemas.NewFile requestedOutputFile =
 new com.actuate.schemas.NewFile();
 requestedOutputFile.setName(outputFileName);
 executeReport.setRequestedOutputFile(requestedOutputFile);
 }
 com.actuate.schemas.ExecuteReportResponse executeReportResponse
 = proxy.executeReport(executeReport);
 System.out.println("Status " + executeReportResponse.getStatus(
));
}

```

## Using SelectJavaReportPage

`SelectJavaReportPage` retrieves a specific page from an Actuate BIRT report specifying a page number, a page range, a component, or other search criteria. The following sample application, `AcSelectJavaReportPage`, selects a single page and saves the result in the specified directory by performing the following tasks:

- Instantiates the `Arguments` class, passing any command line arguments to the constructor

- Instantiates the ActuateController class, getting a server URL from Arguments, if specified in the command line
- Uses methods defined in the controller class to send a request to select a page using com.actuate.schemas.SelectJavaReportPage, SelectJavaReportPageResponse, ViewParameter, ObjectIdentifier, PageIdentifier, NameValuePair, and ArrayOfNameValuePair classes
- Instantiates the SelectJavaReportPage class and specifies the SelectJavaReportPage operation by performing the following tasks:
  - Sets up an ObjectIdentifier object, specifying the name and type of the file to view
  - Sets up a PageIdentifier object, specifying the page number to view and the page view mode
  - Sets up the references to the ObjectIdentifier and PageIdentifier objects in the SelectJavaReportPage object
  - Sets up the view properties used by the BIRT viewer using NameValuePair and ArrayOfNameValuePair classes

The view properties include the following settings:

- ❑ **SVGFlag**  
Specifies whether to use Scalable Vector Graphics (SVG), an XML language used to describe two-dimensional graphics, such as vector graphics shapes, images, and text. This flag is used by the chart engine to determine if SVG chart output is supported.
- ❑ **ContextPath**  
Specifies a context path relative to the root directory of the web server.
- ❑ **MasterPage**  
Determines whether to use the master page, which defines the layout for the pages of a report.
- ❑ **enableMetaData**  
Enables the client to retrieve metadata required to communicate with the service.
- ❑ **displayGroupIcon**  
Enables the display of group icons for viewing related items.
- ❑ **displayFilterIcon**  
Enables the display of filter icons for viewing items using options a user selects.
- ❑ **viewMode**  
Specifies whether the report displays in HTML or PDF.

- Makes the remote call to the Actuate SOAP port using proxy.selectJavaReportPage( ), returning a reference to the com.actuate.schemas.SelectJavaReportPageResponse object
  - Sets up the download directory using mkdir( )
  - Saves the result in the download directory
  - To obtain an image on a page in the document, the application performs the following tasks:
    - Gets the page reference
    - Calls Attachment.getContentData( ) to obtain the page content and sets up a ByteArrayInputStream
    - Calls getEmbed( ), passing in the references to ByteArrayInputStream and ActuateControl objects
- getEmbed( ) performs the following tasks:
- ❑ Sets up an InputStreamReader to iteratively read pattern chunks, line by line, to decode the input stream
  - ❑ Compiles the generic image file name, specified in a regular expression, into a pattern instance, which allows the application to use a Matcher object to identify an embedded image
  - ❑ Uses Matcher.find( ) to scan a chunk to locate a sequence matching the specified pattern
  - ❑ Calls returnId( ) to get the image ID
  - ❑ Sets up a GetJavaReportEmbeddedComponent object, specifying the file name, file type, and a name-value pair indicating the image component ID
  - ❑ Sets the response connection handle, the session ID of the object
  - ❑ Calls ActuateControl.GetJavaReportEmbeddedComponent( ) to get the embedded object
  - ❑ Saves the Attachment to the download directory

AcSelectJavaReportPage class looks like the following example:

```
public class AcSelectJavaReportPage {
 public static ActuateControl actuateControl;
 // download settings
 static String filename;
 static String downloadDirectory = "download";
 static String format = "Reportlet";
 static int pageNumber = 1;
 static com.actuate.schemas.SelectJavaReportPageResponse
```

```

 resp=null;
public static void main(String[] args)
{
 // get command line arguments
 Arguments arguments = new Arguments(args);
 filename = arguments.getArgument();
 pageNumber =
 Integer.parseInt(arguments.getOptionalArgument("1"));
 downloadDirectory =
 arguments.getOptionalArgument(downloadDirectory);

 ...
 actuateControl.selectPage(
 filename,format,pageNumber,downloadDirectory);
 com.actuate.schemas.SelectJavaReportPage
 acselectjavarptpage = new
 com.actuate.schemas.SelectJavaReportPage();

 com.actuate.schemas.ObjectIdentifier objId =
 new ObjectIdentifier();
 objId.setName(filename);
 objId.setType("rptdocument");
 acselectjavarptpage.setObject(objId);

 PageIdentifier pgId = new PageIdentifier();
 pgId.setPageNum(new Long(pageNumber));
 pgId.setViewMode(new Integer(1));

 acselectjavarptpage.setPage(pgId);
 acselectjavarptpage.setOutputFormat(format);
 acselectjavarptpage.setDownloadEmbedded(
 new Boolean(true));

 com.actuate.schemas.NameValuePair nvpair =
 new com.actuate.schemas.NameValuePair();
 nvpair0.setName("SVGFlag");
 nvpair0.setValue("false");

 com.actuate.schemas.NameValuePair nvpair1 =
 new com.actuate.schemas.NameValuePair();
 nvpair1.setName("ContextPath");
 nvpair1.setValue("/");

 com.actuate.schemas.NameValuePair nvpair2 =
 new com.actuate.schemas.NameValuePair();
 nvpair2.setName("MasterPage");
 nvpair2.setValue("true");

 com.actuate.schemas.NameValuePair nvpair3 =
 new com.actuate.schemas.NameValuePair();

```

```

nvpair3.setName("enableMetaData");
nvpair3.setValue("false");

com.actuate.schemas.NameValuePair nvpair4 =
 new com.actuate.schemas.NameValuePair();
nvpair4.setName("displayGroupIcon");
nvpair4.setValue("false");

com.actuate.schemas.NameValuePair nvpair5 =
 new com.actuate.schemas.NameValuePair();
nvpair5.setName("displayFilterIcon");
nvpair5.setValue("false");

com.actuate.schemas.NameValuePair nvpair6 =
 new com.actuate.schemas.NameValuePair();
nvpair6.setName("viewMode");
nvpair6.setValue("1");

com.actuate.schemas.NameValuePair npair[] =
 {nvpair,nvpair1,nvpair2,nvpair3,
 nvpair4,nvpair5,nvpair6};

ArrayOfNameValuePair arr = new ArrayOfNameValuePair();
arr.setNameValuePair(npair);
acselectjavarptpage.setViewProperties(arr);

resp = actuateControl.proxy.selectJavaReportPage(
 acselectjavarptpage);

// Save the result in download directory
new java.io.File(downloadDirectory).mkdir();

String firstAttachmentName =
 actuateControl.saveAttachment(resp.getPageRef(),
 downloadDirectory);
com.actuate.schemas.Attachment att = resp.getPageRef();

byte[] content = att.getContentData();
java.io.ByteArrayInputStream ba =
 new java.io.ByteArrayInputStream(content);

getembed(ba,actuateControl);
}

catch (Exception e)
{
 e.printStackTrace();
}

public static void getembed(
 java.io.ByteArrayInputStream ba,
 ActuateControl actuateControl) throws Exception {
Charset cs = Charset.forName("UTF-8");

```

```

 java.io.InputStreamReader isr =
 new java.io.InputStreamReader(ba,cs);

 Pattern p = Pattern.compile("__imageID=(\\s+\\.jpg)");
 java.io.BufferedReader br =
 new java.io.BufferedReader(isr);
 String chunk = null;

 while ((chunk = br.readLine()) != null) {
 Matcher m = p.matcher(chunk);
 String image_id = null;

 if(m.find()) {
 System.out.println("String read :" + chunk);
 String tmp = m.group();
 image_id = returnId(tmp);
 String decoded_image_id =
 java.net.URLDecoder.decode(image_id, "UTF-8");
 GetJavaReportEmbeddedComponent jrptComp =
 new GetJavaReportEmbeddedComponent();
 jrptComp.setDownloadEmbedded(new Boolean(true));
 ObjectIdentifier jobj = new ObjectIdentifier();
 jobj.setName(filename);
 jobj.setType("rptdocument");
 jrptComp.setObject(jobj);
 com.actuate.schemas.NameValuePair jpair =
 new com.actuate.schemas.NameValuePair();
 jpair.setName("compId");
 jpair.setValue(decoded_image_id);
 com.actuate.schemas.NameValuePair njpair[] =
 {jpair};
 ArrayOfNameValuePair jarr =
 new ArrayOfNameValuePair();
 jarr.setNameValuePair(njpair);
 jrptComp.setComponent(jarr);
 actuateControl.setConnectionHandle(
 resp.getConnectionHandle());
 GetJavaReportEmbeddedComponentResponse jresp =
 actuateControl.proxy.getJavaReportEmbeddedComponent(
 jrptComp);
 String imageName =
 actuateControl.saveAttachment(
 jresp.getEmbeddedRef(), downloadDirectory);
 System.out.println(imageName);
 }
 }
 }
}

```

```
public static String returnId (String str){
 int startPos;
 startPos = str.indexOf("-");
 String id = str.substring(startPos + 1);
 return id;
}
}
```

# 12

## **Developing Actuate Information Delivery API applications using Microsoft .NET**

This chapter contains the following topics:

- About the Microsoft .NET client
- About the Actuate Information Delivery API framework
- Developing Actuate Information Delivery API applications

---

## About the Microsoft .NET client

BIRT iHub System contains WSDL documents that define the Actuate web services schema for the Microsoft C# and Visual Basic development environments. The Microsoft .NET client provides a web services framework for building applications that communicate with BIRT iHub System, using SOAP messaging.

The Actuate Information Delivery API framework uses elements of the Microsoft .NET development platform to support the following features:

- Automatic generation of the Actuate Information Delivery API code library from an Actuate WSDL document  
The library contains the classes, including server proxies, that you can use to write an Actuate Information Delivery API application.
- A SOAP processor that provides automatic encoding and decoding of SOAP messages

A Microsoft .NET client solution containing example projects ships with BIRT iHub Integration Technology. The Microsoft .NET client is in the following directory:

`ACTUATE_HOME\ServerIntTech\Web Services\Examples\MS .NET Client`

The MS .NET Client directory contains the following subdirectories:

- Source  
Contains the example projects.
- Report  
Contains the report files used by the example projects.
- Download  
Stores the files or reports downloaded by the example projects. This directory is initially empty.
- Build  
After building an example project, there are two subdirectories in the build directory, debug and release. Each directory contains copies of the executable files for the project.

There are two solution description files in the MS .NET Client root directory:

- Server Proxy.sln  
Open the server proxy solution first, then build it to generate the proxy DLL. The build process puts the Server Proxy.dll and Server Proxy.pdb files in a build directory for other projects to share and reference.

- Examples.sln

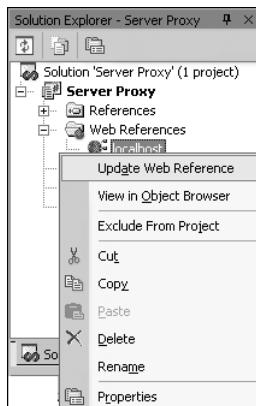
After you build the server proxy, open the examples solution and build it to compile the example projects. Copy the server proxy files, Server Proxy.dll and Server Proxy.pdb, to the /Debug and /Release subfolders of the example projects.

If changes occur in the WSDL interface, download the latest WSDL file from the BIRT iHub and replace the ActuateAPI.wsdl file in the server proxy solution at the following location:

```
ACTUATE_HOME\ServerIntTech\Web Services\Examples\MS .NET Client
 \source\Server Proxy\Web References\localhost
```

Rebuild the proxy, then rebuild all the examples. To update the WSDL file using the Microsoft .NET development tool, perform the following tasks:

- 1 In Solution Explorer, expand Web References.
- 2 Select localhost, right-click, then choose Update Web Reference, as shown in Figure 12-1.



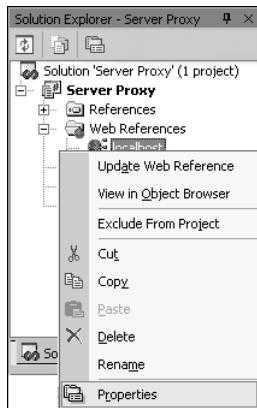
**Figure 12-1** Updating the web reference for localhost

This procedure updates the local copy of the WSDL file from an BIRT iHub configured to run at the following default location:

<http://localhost:8000/wsdl/v11/net/all>

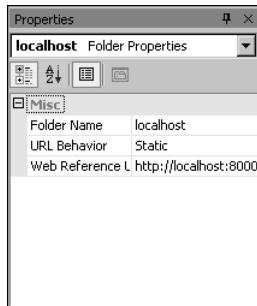
If you are using a BIRT iHub that listens at a different port on the local machine, change the URL property for the Web Reference, localhost. To update the URL property for the existing Web Reference, localhost, perform the following tasks:

- 1 In Solution Explorer, expand Web References.
- 2 Select localhost, right-click, then choose Properties, as shown in Figure 12-2.



**Figure 12-2** Changing the web reference for localhost

Properties appears, as shown in Figure 12-3.



**Figure 12-3** The Properties page for localhost

### 3 Change the Web Reference URL for localhost to the correct value.

Alternatively, you can delete the Web Reference, localhost, then add a new Web Reference that contains the correct URL property.

The ACTUATE\_HOME\ServerIntTech\Web Services\Examples\MS .NET Client \source\Server Proxy\Web References\localhost directory also contains the following files:

- **Reference.cs**  
Contains the Actuate Information Delivery API classes generated from the Actuate WSDL document
- **Reference.map**  
Contains the following references:
  - Namespace declaration, xsd, used as a prefix for every Actuate Information Delivery API SOAP message to determine the scope of the XML namespace. The following namespace declaration indicates that a message

is based on the World Wide Web Consortium XML schema initially published in 2001:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

- Specific instance of the XML schema. In order to use namespace attribute types in an XML document, you must define the xsi namespace, as shown in the following example:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

- Web services system. The following reference specifies the ActuateAPI.wsdl document for Actuate 11:

```
url="http://localhost:8000/wsdl/v11/net/all"
filename="ActuateAPI.wsdl"
```

The following sections describe how to build client applications that use the libraries provided by the Actuate Information Delivery API for development in a Microsoft .NET environment.

---

## About the Actuate Information Delivery API framework

The Actuate Information Delivery API framework contains the client-side bindings that the Actuate IDAPI application requires to implement SOAP processing. The SOAP processor serializes, or transforms a remote procedure call by the application into an XML-based SOAP message that asks BIRT iHub to perform a web service. The application sends the request across the network using the HyperText Transfer Protocol (HTTP) transport layer.

BIRT iHub receives the request and deserializes the SOAP message. BIRT iHub performs an appropriate action and sends a response, in the form of a SOAP message, back to the application.

The SOAP processor embedded in the Actuate Information Delivery API framework automates the serialization and deserialization of SOAP messages, relieving the developer of the necessity to program the application at this level.

The following sections describe these key elements of the framework as background information to provide the developer with an understanding of the way the Actuate Information Delivery API framework operates.

## Using a data type from a WSDL document to generate a C# class

When you generate the Actuate Information Delivery API source code, Microsoft .NET builds a C# class from each WSDL type definition.

For example, the Login type definition translates the following WSDL into the C# equivalent:

```
<s:complexType name="Login">
 <s:sequence>
 <s:element name="User" type="s:string" />
 <s:element minOccurs="0" name="Password"
 type="s:string" />
 <s:element minOccurs="0" name="EncryptedPwd"
 type="s:string" />
 <s:element minOccurs="0" name="Credentials"
 type="s:base64Binary" />
 <s:element minOccurs="0" name="Domain"
 type="s:string" />
 <s:element minOccurs="0" name="UserSetting"
 type="s:boolean" />
 <s:element minOccurs="0" name="ValidateUserGoups"
 type="s0:ArrayOfString" />
 </s:sequence>
</s:complexType>
```

The C# class receives the name that appears in the WSDL type definition. The class defines the attributes and data types for each WSDL element. Applying a System.Xml.Serialization class, such as XmlElementAttribute, specifies how the .NET framework serializes and deserializes a C# attribute.

The following code shows the C# class definition for the Login class:

```
[System.Xml.Serialization.XmlTypeAttribute(
Namespace="http://schemas.actuate.com/actuate11")]
public class Login {
 public string User;
 public string Password;
 public string EncryptedPwd;
 [System.Xml.Serialization.XmlElementAttribute(DataType=
 "base64Binary")]
 public System.Byte[] Credentials;
 public string Domain;
 public bool UserSetting;
 [System.Xml.Serialization.XmlIgnoreAttribute()]
 public bool UserSettingSpecified;
 [System.Xml.Serialization.XmlArrayItemAttribute(
 String", IsNullable=false)]
 public string[] ValidateUserGroups;
}
```

## Mapping the portType to a web service interface

The .NET framework uses the portType and binding in the WSDL document to create the web service interface. The web service interface specifies the input and output messages of the request-response pairs for an operation and the service name and port.

The following WSDL code shows the specification of the input and output messages, Login and LoginResponse, and the binding of this request-response pair to the login operation:

```
<wsdl:portType name="ActuateSoapPort">
 <wsdl:operation name="login">
 <wsdl:input message="tns:Login" />
 <wsdl:output message="tns:LoginResponse" />
 </wsdl:operation>
 ...
</wsdl:portType>
```

The following WSDL code shows the specification of the service and port names and the binding of the port to a machine address:

```
<wsdl:binding name="ActuateSoapBinding"
 type="tns:ActuateSoapPort">
 <soap:binding transport="http://schemas.xmlsoap.org/soap
 /http" style="document" />
</wsdl:binding>
<wsdl:service name="ActuateAPI">
 <wsdl:port name="ActuateSoapPort"
 binding="tns:ActuateSoapBinding">
 <soap:address location="http://ENL2509:8000" />
 </wsdl:port>
</wsdl:service>
```

An application uses this information to construct an interface to access the operations available from the service using a remote procedure call (RPC). In the following code example, taken from ActuateAPI class, the client application performs the following tasks:

- Sets up the SOAP message.
- The remote procedure call, login( ), submits a request, passing a Login object as a parameter and returns a LoginResponse object in response from the BIRT iHub defined by ActuateSoapPort in the web service interface.

```
[System.Web.Services.Protocols.SoapHeaderAttribute
 ("HeaderValue")]
```

---

# Developing Actuate Information Delivery API applications

The following sections describe the development of Actuate Information Delivery API applications:

- Writing a program that logs in to BIRT iHub System
- Writing a simple administration application
- Performing a search operation
- Writing a batch or transaction application
- Uploading a file
- Downloading a file

The code examples and explanations in this chapter parallel the code examples and explanations in Chapter 11, “Developing Actuate Information Delivery API applications using Java.”

## Writing a program that logs in to BIRT iHub System

A login operation authenticates a user in BIRT iHub System. A login operation involves the following actions:

- An IDAPI application sends a Login request to BIRT iHub System.
- BIRT iHub System sends a Login response to the IDAPI application.

A Login action receives a LoginResponse message from BIRT iHub System. When a Login action succeeds, the LoginResponse message contains an AuthId, which is an encrypted, authenticated token. When a Login action fails, BIRT iHub System sends a LoginResponse message containing an error code and a description of the error.

The authentication ID in the LoginResponse message remains valid for the current session. Any subsequent request that the application sends to a volume must include the authentication ID in the message. The following sections describe the SOAP messages, classes, and program interactions necessary to implement a successful login operation.

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. AcLogin class logs in to the volume to authenticate the user. If the login succeeds, the application prints a success message. If the login fails, the application prints a usage message. In AcLogin class, Main function performs the following operations:

- Specifies Actuate namespace and calls Usage function, which analyzes the command line arguments as shown in the following code:

```
using System;
using System.Web.Services.Protocols;
using Server_Proxy.localhost;
namespace Actuate
{
 class AcLogin
 {
 [STAThread]
 static void Main(string[] args)
 {
 string l_url;
 string l_userName;
 string l_password;
 string l_volumename;
 if (Usage(args, out l_url, out l_userName,
 out l_volumename, out l_password) == 0) return;
```

The following list describes the options that can be specified as command line arguments:

- serverURL  
-h specifies the SOAP endpoint. The default value is http://localhost:8000.
- username  
-u specifies the user name. The default value is Administrator.
- password  
-p specifies the password. The default value is "".
- volumename  
-v specifies the target volume name. Actuate Release 11 requires a volume name in the SOAP header. For earlier releases, this specification is optional.
- printUsage  
-? prints the usage statement. The default value is the following string:  
Usage: Login -h http://localhost:8000 -u username -p password  
-? help
- Creates an instance of the server proxy and constructs the SOAP header:  

```
ActuateAPI l_proxy = new ActuateAPI();
l_proxyHeaderValue = new Header();
l_proxy.Url = l_url;
```

AcLogin uses the following classes defined in References.cs to perform these operations:

- ActuateAPI class is a subclasses of System.Web.Services.Protocols.SapHttpClientProtocol that specifies the protocol for .NET to use at run time and defines the proxies used to make web service requests.
- Header class is a subclass of System.Web.Services.Protocols.SoapHeader that defines the following Actuate SOAP header extensions:
  - AuthId is a system-generated, encrypted string returned by BIRT iHub in a Login response. All requests, except a Login request, must have a valid AuthId in the SOAP header.
  - Locale specifies the language, date, time, currency, and other conventions for BIRT iHub to use when returning data to a client.
  - TargetVolume is an optional element that indicates the volume that receives the request.
  - ConnectionHandle supports keeping a connection open to view a persistent report. If ConnectionHandle is present in the SOAP header, the system routes subsequent viewing requests to the same View service that returned the ConnectionHandle.
  - DelayFlush tells BIRT iHub to write updates to the disk when the value is False.

- Instantiates the Login object and prepares the login parameters

```
Server_Proxy.localhost.Login l_req =
 new Server_Proxy.localhost.Login();
l_req.Password = l_password;
l_req.User = l_userName;
l_req.Domain = l_volumename;
l_req.UserSetting = true;
l_req.UserSettingSpecified = true;
```

- Sends the Login request, handling any exceptions by writing a usage statement to the console

```
LoginResponse l_res;
try
{
 l_res = l_proxy.login(l_req);
}
catch(SoapException e)
{
 Console.WriteLine("Please try again.\n");
 Usage: Login -h http://localhost:8000
 -u username -p password -v volumename -? help \n";
 PrintExceptionDetails((SoapException) e);
}
```

```
 return;
}
```

- Processes a successful Login response by writing a success message to the console and storing AuthId as a SOAP header value for use in the next request

```
Console.WriteLine("Congratulations! You have successfully
logged into BIRT iHub System as "+l_userName);
 // Store the authentication id
 l_proxy.HeaderValue.AuthId = l_res.AuthId;
}
```

AcLogin uses the Actuate Information Delivery API classes in References.cs, generated from the Actuate WSDL document. References.cs provides the following code definitions:

- Declares the proxy namespace, XML serialization, and web services protocols for the system

```
namespace Server_Proxy.localhost {
 using System.Diagnostics;
 using System.Xml.Serialization;
 using System;
 using System.Web.Services.Protocols;
 using System.ComponentModel;
 using System.Web.Services;
```

- Defines the ActuateAPI class, which specifies the bindings for the SOAP HTTP client protocol, extended Actuate SOAP header values, and proxy calls

```
[System.Web.Services.WebServiceBindingAttribute(
Name="ActuateSoapBinding",
Namespace="http://schemas.actuate.com/actuate11/wsdl")]
public class ActuateAPI
: System.Web.Services.Protocols.SoapHttpClientProtocol {
 public Header HeaderValue;
 public ActuateAPI() {
 this.Url = "http://ENL2509:8000";
 }
 /// <remarks/>
 [System.Web.Services.Protocols.SoapHeaderAttribute(
 ("HeaderValue"))]
 [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
 "", Use=System.Web.Services.Description.SoapBindingUse
 .Literal, ParameterStyle=
 System.Web.Services.Protocols.SoapParameterStyle.Bare)]
 [return:
 System.Xml.Serialization.XmlElementAttribute(
 "LoginResponse",
 Namespace="http://schemas.actuate.com/actuate11")]
}
```

```

public LoginResponse
 login([System.Xml.Serialization.XmlElementAttribute
 (Namespace="http://schemas.actuate.com/actuate11")] Login
 Login) {
 object[] results = this.Invoke("login", new object[] {
 Login});
 return ((LoginResponse)(results[0]));
}

```

## Writing a simple administration application

A simple administration application typically involves a create operation for one of the following BIRT iHub objects:

- User
- Folder
- UserGroup

To perform an administration operation that acts on an existing object in the volume, such as a select, update, or delete operation, you must apply a search condition to the operation. To perform an administration operation that contains a set of administration operations requests, submit the request as a batch or transaction.

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. The application class, AcCreateUser, logs in to a volume as Administrator and creates a user. AcCreateUser performs the following tasks:

- Instantiates a CreateUser object and prepares a create user operation
 

```

CreateUser l_createUser = new CreateUser();
l_createUser.IgnoreDup = true;
l_createUser.IgnoreDupSpecified = true;

```
- Instantiates a User object, setting the username, password, and home folder
 

```

l_createUser.User = new User();
l_createUser.User.Name = l_UserName;
l_createUser.User.Password = l_password;
l_createUser.User.HomeFolder = l_homeFolder;

```

User is a complex data type that represents user attributes.
- Sets additional view preference, notification, e-mail, and job priority options in the User object
 

```

l_createUser.User.ViewPreference = UserViewPreference.DHTML;
l_createUser.User.ViewPreferenceSpecified = true;
l_createUser.User.SendNoticeForSuccess = true;

```

```

l_createUser.User.SendNoticeForSuccessSpecified = true;
l_createUser.User.SendNoticeForFailure = true;
l_createUser.User.SendNoticeForFailureSpecified = true;
l_createUser.User.SendEmailForSuccess = true;
l_createUser.User.SendEmailForSuccessSpecified = true;
l_createUser.User.SendEmailForFailure = true;
l_createUser.User.SendEmailForFailureSpecified = true;
l_createUser.User.EmailAddress = (l_createUserName + "@" +
 "localhost");
l_createUser.User.MaxJobPriority = 1000;
l_createUser.User.MaxJobPrioritySpecified = true;

```

- Instantiates an AdminOperation object and assigns the reference to the CreateUser object to it

```

AdminOperation l_createUserOpt = new AdminOperation();
l_createUserOpt.Item = l_createUser;

```

- Assembles the create user request in an AdminOperation array

```

AdminOperation[] l_adminRequest = new AdminOperation[1];
l_adminRequest[0] = l_createUserOpt;

```

- Makes an administrate request using the proxy, passing in the reference to the AdminOperation array

```

try {
 l_proxy.administrate(l_adminRequest);
}
catch(SoapException e) {
 PrintExceptionDetails(e);
 return;
}

```

## Performing a search operation

Many operations support acting on one or more items in a volume. To target the items on which to act, you must apply a search condition to the operation. The Actuate Information Delivery API library contains many special classes for setting a search condition and implementing a search for an item in a volume.

The Actuate Information Delivery API provides three sets of parameters that support searching for the data on which an operation acts. Typically, these parameters apply to operations that select, update, move, copy, or delete volume items. The parameters are:

- Id or IdList
- Name or NameList
- Search

Use SelectFiles to retrieve file properties using the ID or name of a single file or folder, or a list of files or folders, in a volume. SelectFiles can recursively search all subfolders in a directory. To restrict a search to the items in a single directory, not including subfolders, use GetFolderItems. SelectFiles does not retrieve file or folder content. SelectFiles supports three types of searches:

- Use Name or Id to retrieve a single file or folder.
- Use NameList or IdList to retrieve a list of files or folders in a directory.
- Use Search to retrieve all files or folders that match a given condition.

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. In the example application, AcSelectFiles specifies a search condition and performs the following operations:

- Specifies a search condition for the file type using the wildcard, \*, to target all files that contain the character, R, as the first character in the file extension

```
namespace Actuate
{
 class AcSelectFiles
 {
 ...
 string fileType = "R*";
```

A wildcard is a character used in a search or conditional expression that matches one or more literal characters. Actuate wildcards include the ones in the following list:

- ? matches any single one- or two-byte character.
- # matches any ASCII numeric character [0-9].
- \* matches any number of characters.

The wildcard expression in the example targets files in BIRT iHub such as a BIRT executable file with the file extension RPTDESIGN, and a BIRT document file with the file extension RPTDOCUMENT.

- Instantiates a FileCondition object, setting the condition to match on FileField.FileType using the wildcard expression

```
static void SearchByCondition (string fileType)
{
 Server_Proxy.localhost.SelectFiles l_req = new
 SelectFiles();
 FileCondition l_fileCondition = new FileCondition();
 l_fileCondition.Field = FileField.FileType;
 l_fileCondition.Match = fileType;
```

- Instantiates a FileSearch object, setting the search to the properties specified in the FileCondition object

```
FileSearch l_fileSearch = new FileSearch();
l_fileSearch.Item = l_fileCondition;
```

An application sets the search condition for a file using the FileSearch class. FileSearch is a complex data type that contains the list of properties to specify in a file search condition.

An application can specify the search condition for a file using one or more of the following fields:

- Name
- FileType
- Description
- PageCount
- Size
- TimeStamp
- Version
- VersionName
- Owner

Use ArrayOfFileCondition to specify multiple search conditions.

- Specifies the fetch handle and SelectFilesResponse object for processing the results

```
l_fileSearch.FetchSize = 1;
l_fileSearch.FetchSizeSpecified = true;
SelectFilesResponse l_res;
```

- Makes the selectFiles( ) proxy call, obtaining the reference to the SelectFilesResponse object

```
do
{
 try
 {
 l_res = l_proxy.selectFiles(l_req);
 }
 catch(Exception e)
 {
 PrintExceptionDetails(e);
 }
}
```

```

 return;
 }
...

```

A fetch handle indicates that the number of items in the result set exceeds the fetch size limit. A fetch handle returns as a parameter in the response to a Select or Get request, such as SelectFiles or GetFolderItems.

Use the fetch handle to retrieve more results from the result set. In the second and subsequent calls for data, you must specify the same search condition that you used in the original call. All Get and Select requests, except SelectFileType, support the use of a fetch handle.

- Continues processing until there are no more results, printing an appropriate output message

```

File[] l_fileList = (File[]) l_res.Item;
if (l_fileList != null)
{
 for(int i = 0; i < l_fileList.Length; i++)
 {
 Console.WriteLine();
 Console.WriteLine("Item " + i + " Id:" +
 l_fileList[i].Id);
 Console.WriteLine("Item " + i + " Name:" +
 l_fileList[i].Name);
 Console.WriteLine("Item " + i + " Owner:" +
 l_fileList[i].Owner);
 Console.WriteLine("Item " + i + " Description:" +
 l_fileList[i].Description);
 Console.WriteLine("Item " + i + " File Type:" +
 l_fileList[i].FileType);
 Console.WriteLine("Item " + i + " File Size:" +
 l_fileList[i].Size);
 }
}
l_fileSearch.FetchHandle = l_res.FetchHandle;
} while(l_res.FetchHandle != null);

```

## Writing a batch or transaction application

Actuate Information Delivery API supports batch and transaction administration operations. An IDAPI application uses a mixture of developer and API classes to implement a batch or transaction administration operation, including:

- Administrate
- AdminOperation

- Transaction
- TransactionOperation

The following sections explain the use of these administration operation classes in detail.

## About batch and transaction operations

A batch application submits an array of administration operation requests to a volume using one composite Administrate message. An Administrate request can contain any number of AdminOperation requests in the batch.

An AdminOperation request can contain any number of Transaction requests.

A Transaction request is a composite message that can contain any number of TransactionOperation requests. A TransactionOperation represents a single unit of work within a Transaction.

The default level of granularity for a transaction is an object. One operation run against one object is atomic. The use of an explicit Transaction tag in a composite Administrate message expands the transaction boundary to include multiple TransactionOperation requests.

To perform an administration operation that contains a set of requests, submit the request as a batch or transaction within one composite Administrate message. If a batch request fails, the operations that complete successfully before the failed operation still take effect. If a transaction operation fails, none of the operations in the transaction take effect. BIRT iHub rolls all the work back, leaving the system in the state it was in just prior to the execution of the transaction operation.

## Implementing a transaction-based application

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. In the example application, AcAddUser\_Tran performs a transaction-based administration operation to add multiple volume users. When the command line includes the optional ignoreDup argument, the program ignores errors if a user already exists.

AcAddUsers\_Tran.addUsers( ) performs the following tasks:

- Defines a TransactionOperation array, dimensioning the array to the number of new users

```
namespace Actuate
{
 class AcAddUser_Tran
 {
 ...
 TransactionOperation[] l_transOperation =
 new TransactionOperation[numberOfUsers];
```

- Within a loop, for each user:

- Instantiates the CreateUser object and sets IgnoreDup

```
CreateUser l_createUser = new CreateUser();
l_createUser.IgnoreDup = true;
l_createUser.IgnoreDupSpecified = true;
```

- Instantiates a User object, passing the reference to the CreateUser object, and setting the user name, password, home folder, and other options such as view preference, notification, and e-mail

```
for (int i = 0; i < numberOfUsers; i++)
{
 l_createUser.User = new User();
 l_createUser.User.Name = l_tranUserName;
 l_createUser.User.Password = l_tranPassword;
 l_createUser.User.HomeFolder = l_tranHomeFolder;
 ...
}
```

- Instantiates an AdminOperation object, passing the reference to the CreateUser object

```
AdminOperation l_createUserOpt = new AdminOperation();
l_createUserOpt.Item = l_createUser;
```

- Instantiates a TransactionOperation object, passing the reference to the CreateUser object to complete the set up of the transaction operation

```
l_transOperation[i] = new TransactionOperation();
l_transOperation[i].Item = l_createUser;
```

- After the end of the loop, instantiates an AdminOperation object, passing the reference to the Transaction object to create the composite administration operation

```
AdminOperation l_adminOperation = new AdminOperation();
l_adminOperation.Item = l_transOperation;
```

- Instantiates an AdminOperation array, passing the reference to the AdminOperation object to complete the assembly of the administration operation request

```
AdminOperation[] l_adminRequest = new AdminOperation[1];
l_adminRequest[0] = l_adminOperation;
```

- Makes the administrate proxy call, passing the reference to the administration operation array, handling any Exception that occurs

```
try
{
 l_proxy.administrat(e(l_adminRequest);
}
```

```
 catch(Exception e)
 {
 Console.WriteLine("Create user transaction failed.\n");
 PrintExceptionDetails(e);
 return;
 }
}
```

- Prints an output message, if the operation succeeds

```
Console.WriteLine("Create user transaction succeeded.\n");
```

## Uploading a file

To upload a file to a volume, you must identify the file to upload and the volume in which to place the file. You can also specify how to work with existing versions of the file you upload. Using Actuate's open server technology, you can upload third-party file types and native Actuate file types.

### About ways of uploading a file

When you upload a file, the content streams to the volume. You can stream a report with a SOAP message in two ways:

- Embed the file in the response.

In embedding a file, the application specifies the ContentLength in the HTTP header. If you use HTTP 1.0, you typically choose to embed the file.

- Send the file as a MIME attachment.

A MIME attachment transmits the contents of the file outside the boundary of the SOAP message.

A SOAP message with a MIME attachment consists of three parts:

- HTTP header
- Actuate SOAP message
- File attachment

The following example uses a MIME attachment and relevant Actuate Information Delivery API classes to build an application that uploads a file to a volume.

## Using UploadFile

Use the UploadFile class to upload a file to a volume. The file content streams to BIRT iHub as an unchunked MIME attachment to the SOAP request. The UploadFile class contains the following attributes:

- NewFile is the NewFile object to upload.

- CopyFromLatestVersion is an array of strings used to copy one or more of the following properties from the latest version of the file, if one exists, to the version of the file that you upload:
  - Description is a description of the file.
  - Permissions, the access control list (ACL) specifying the users and user groups that can access the file.
  - ArchiveRule specifies the autoarchive rules for the file, which determine how BIRT iHub ages the file and when the file expires.
- Content is the Attachment object that specifies the content Id, content type, content length, content encoding, locale, and content data.

## Building an application that uploads a file

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. In this example application, AcUploadFile performs the following operations:

- Instantiates an ActuateAPIEx object for specifying the Actuate IDAPI SOAP header extension elements, such as AuthId

```
namespace Actuate
{
 class AcUploadFile
 {
 ...
 ActuateAPI l_proxy;

 ActuateAPIEx l_proxyEx= new ActuateAPIEx();
 l_proxyEx.Url = l_proxy.Url;
 l_proxyEx.HeaderValue = new Header();
 l_proxyEx.HeaderValue.AuthId =
 l_proxy.HeaderValue.AuthId;
 }
}
```

- Prepares the UploadFile request by instantiating UploadFile, NewFile, and Attachment objects, and specifying the file name, content type, and content ID

```
UploadFile l_req = new UploadFile();
l_req.NewFile = new NewFile();
l_req.NewFile.Name = l_encFileName;
l_req.Content = new Attachment();
l_req.Content.ContentType = "binary";
l_req.Content.ContentId = "Attachment";
```

- Opens the file for upload by constructing a FileStream object and passing the reference to the ActuateAPIEx object, handling any exception by outputting a message to the console

```

try
{
 ActuateAPIEx.UploadStream =
 new FileStream(l_localFileName, FileMode.Open);
}
catch(Exception e)
{
 Console.WriteLine("Cannot open the file" + e.Message);
 return;
}

```

- Performs the UploadFile administration operation by making an upload file proxy call and closing the file stream after the operation completes

```

UploadFileResponse l_res = null;
try
{
 l_res = l_proxyEx.uploadFile(l_req);
}
catch(Exception e)
{
 PrintExceptionDetails(e);
}
Console.WriteLine("Uploaded " + l_localFileName + " with
 file id: "
 + l_res.FileId);
ActuateAPIEx.UploadStream.Close();

```

## Downloading a file

To download a file from a volume, identify the file and indicate whether to embed the content in the response or use chunked transfer encoding. In HTTP 1.0, you must embed the entire file in the response and send it in a long, uninterrupted file stream. In HTTP 1.1, you can send the file in smaller chunks, which increases the efficiency of the file transfer. Although the volume supports both methods, BIRT iHub messages typically use chunked transfer encoding.

The following example application uses chunked transfer encoding and relevant com.actuate.schemas classes to build an application that downloads a file from a volume.

### Using DownloadFile

The DownloadFile class downloads a file from a volume to the client. You can choose to embed the file content in the response or send it to the client as an attachment.

The DownloadFile class contains the following list of attributes:

- FileName or FileId is a string specifying the ID or name of the file to download.  
Specify either FileName or FileId.
- DecomposeCompoundDocument is a Boolean indicating whether to download a compound document as one file or multiple attachments.  
If the DecomposeCompoundDocument value is False, you can download the file as a single file. If the value is True, and the file is a compound document, the volume splits the file into attachments containing the atomic elements of the compound document such as fonts and images. A decomposed document is not in a viewable format. The default value is False.
- DownloadEmbedded is a Boolean indicating whether to embed the content in the response or use chunked transfer encoding.
- FileProperties is a string array specifying the file properties to return.

## Building an application that downloads a file

The following application derives from the code in the BIRT iHub Integration Technology example applications for the Microsoft .NET client. In the example application, AcDownloadFile\_Chunked class, downloads a file from the volume, using the chunked option, and saves the file in the specified directory.

The AcDownloadFile\_Chunked class performs the following operations:

- Instantiates an ActuateAPIEx object for specifying the Actuate IDAPI SOAP header extension elements, such as AuthId

```
namespace Actuate
{
 class AcDownloadFile_Chunked
 {
 ...
 ActuateAPI l_proxy;

 ActuateAPIEx l_proxyEx= new ActuateAPIEx();
 l_proxyEx.Url = l_proxy.Url;
 l_proxyEx.HeaderValue = new Header();
 l_proxyEx.HeaderValue.AuthId =
 l_proxy.HeaderValue.AuthId;
 }
}
```

- Prepares the DownloadFile request by instantiating DownloadFile and DownloadFileResponse objects, then specifying the file name, item type, and option to download an embedded file

```
DownloadFile l_req = new DownloadFile();
l_req.Item = l_filename;
```

```
l_req.ItemElementName = ItemChoiceType34.FileName;
l_req.DownloadEmbedded = false;
DownloadFileResponse l_res;
```

- Performs the DownloadFile administration operation by making the download file proxy call and handling any exceptions

```
try
{
 l_res = l_proxyEx.downloadFile(l_req);
}
catch(Exception e)
{
 PrintExceptionDetails(e);
 return;
}
```

- Saves the downloaded file to the specified location and closes the file stream

```
FileStream l_fileStream = new FileStream(l_directory + "\\" +
 l_filename.Substring(l_filename.LastIndexOf("/") + 1),
 FileMode.Create);
((MemoryStream)
 ActuateAPIEx.DownloadStream).WriteTo(l_fileStream);
l_fileStream.Close();
```



# 13

# Actuate Information Delivery API operations

This chapter provides reference documentation for Actuate Information Delivery API operation classes listed in alphabetical order. Each entry includes a general description of the operation and request-response elements.

## IDAPI operations quick reference

Table 13-1 lists the Actuate IDAPI Operations. For more information about IDAPI Operation WSDL definitions, see Chapter 9, “Understanding the Information Delivery API and schema.”

**Table 13-1** IDAPI operations summary

| Operation                                 | Description                                      |
|-------------------------------------------|--------------------------------------------------|
| <a href="#">Administate</a>               | Performs file and user administrative operations |
| <a href="#">CallOpenSecurityLibrary</a>   | Executes an RSSE operation                       |
| <a href="#">CancelJob</a>                 | Cancels a job                                    |
| <a href="#">CancelReport</a>              | Cancels a report execute operation               |
| <a href="#">CreateParameterValuesFile</a> | Creates a parameter values file                  |
| <a href="#">CreateResourceGroup</a>       | Creates a resource group                         |
| <a href="#">CubeExtraction</a>            | Extracts data from a cube                        |

*(continues)*

**Table 13-1** IDAPI operations summary (continued)

| Operation                           | Description                                                           |
|-------------------------------------|-----------------------------------------------------------------------|
| DataExtraction                      | Extracts data from a non-cube resource or file                        |
| DeleteResourceGroup                 | Removes a resource group                                              |
| DownloadFile                        | Downloads a file                                                      |
| ExecuteReport                       | Generates a temporary report                                          |
| ExecuteVolumeCommand                | Executes predefined volume control commands                           |
| ExtractParameterDefinitionsFromFile | Extracts parameter definitions from a report file                     |
| ExportParameterDefinitionsToFile    | Exports parameter definitions from a report file                      |
| GetAllCounterValues                 | Returns the values for all counters                                   |
| GetBookmarks                        | Returns a list of bookmarks from a report                             |
| GetCapabilities                     | Returns a list of capabilities for a user group                       |
| GetCapabilitiesByCategory           | Returns a sorted list of capabilities for a group                     |
| GetCapabilityCategories             | Returns capability categories                                         |
| GetCounterValues                    | Returns the IDs, names, and values of a counter.                      |
| GetCubeMetaData                     | Returns cube result set schema details                                |
| GetDataExtractionFormats            | Returns a list of the output format and MIME types for each file type |
| GetDocumentConversionOptions        | Returns a list of the document conversion options for each file type  |
| GetFactoryServiceInfo               | Returns details about a Factory service                               |
| GetFactoryServiceJobs               | Returns a list of jobs pending and running for a Factory service      |
| GetFileACL                          | Returns the access control list for a file                            |
| GetFileCreationACL                  | Returns the access control list for a template                        |
| GetFileDetails                      | Returns details about a file                                          |
| GetFileTypeParameterDefinitions     | Returns parameters for the specified file type                        |
| GetFolderItems                      | Returns a list for folder contents                                    |
| GetJavaReportEmbeddedComponent      | Returns an embedded component, such as an image or graph              |
| GetJavaReportTOC                    | Returns the table of contents                                         |
| GetJobDetails                       | Returns the detail information for a job                              |
| GetMetaData                         | Returns metadata for the result set schema                            |
| GetNoticeJobDetails                 | Returns the details of a job notice                                   |

**Table 13-1** IDAPI operations summary (continued)

| Operation                           | Description                                        |
|-------------------------------------|----------------------------------------------------|
| GetPageCount                        | Returns the number of pages in a report file       |
| GetPageNumber                       | Returns the current page number                    |
| GetParameterPickList                | Returns a parameter pick list                      |
| GetReportParameters                 | Returns the list of report parameters              |
| GetResourceGroupInfo                | Returns the resource group details                 |
| GetResourceGroupList                | Returns a list of resource groups                  |
| GetServerResourceGroupConfiguration | Returns a resource group configuration             |
| GetSyncJobInfo                      | Returns information about an immediate job         |
| GetSystemMDSInfo                    | Returns Message Distribution service details       |
| GetSystemPrinters                   | Returns a list of printers                         |
| GetSystemServerList                 | Returns a list of servers in a cluster             |
| GetTOC                              | Obsolete                                           |
| GetUserExtendedProperties           | Returns the extended properties for a user         |
| GetUserGroupExtendedProperties      | Returns the extended properties for a user group   |
| GetUserGroupProductAccess           | Returns the product family access for a user group |
| GetUserLicenseOptions               | Returns a list of licensed options for a user      |
| GetUserPreference                   | Returns user preferences                           |
| GetUserPrinterOptions               | Returns user printing options                      |
| GetVolumeProperties                 | Returns volume properties                          |
| GrantUserGroupCapabilities          | Adds user group capabilities                       |
| GrantUserGroupProductAccess         | Adds user group product access                     |
| InstallApp                          | Installs an application                            |
| IsSSOEnabled                        | Determines whether single-sign-on is enabled       |
| Login                               | Authenticates a user                               |
| Ping                                | Pings the server                                   |
| PrintReport                         | Prints a document                                  |
| RevokeUserGroupCapabilities         | Removes user group capabilities                    |
| RevokeUserGroupProductAccess        | Removes user group product access                  |
| SaveTransientReport                 | Saves a report produced by an immediate job        |
| SelectFiles                         | Sets selected files                                |

*(continues)*

**Table 13-1** IDAPI operations summary (continued)

| Operation                           | Description                                                        |
|-------------------------------------|--------------------------------------------------------------------|
| SelectFileTypes                     | Sets selected file types                                           |
| SelectJavaReportPage                | Sets selected report page                                          |
| SelectJobNotices                    | Sets selected job notices                                          |
| SelectJobs                          | Sets selected jobs                                                 |
| SelectJobSchedules                  | Sets selected job schedules                                        |
| SelectUserGroups                    | Sets selected user groups                                          |
| SelectUsers                         | Sets selected users                                                |
| SetServerResourceGroupConfiguration | Sets the resource group configuration                              |
| SetUserExtendedProperties           | Sets extended properties for a user                                |
| SetUserGroupExtendedProperties      | Sets extended properties for a user group                          |
| SetUserPreference                   | Sets user preferences                                              |
| SubmitJob                           | Creates a job                                                      |
| SystemLogin                         | Authenticates the administrator user                               |
| UpdateResourceGroup                 | Refreshes the resource group list                                  |
| UploadFile                          | Copies a file to the volume                                        |
| VerifyAuthId                        | Checks for the existence of an AuthID                              |
| WaitForExecuteReport                | Returns the status of a cancel request for an immediate report job |

## Administate

Specifies the AdminOperation element which controls abilities to modify a volume. Only a volume administrator or a user in the Administrator role uses Administate operations.

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request element</b> | <b>AdminOperation</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|                        | Any sequence of <a href="#">CopyFile</a> , <a href="#">CreateFileType</a> , <a href="#">CreateFolder</a> , <a href="#">CreateUser</a> , <a href="#">CreateUserGroup</a> , <a href="#">DeleteFile</a> , <a href="#">DeleteFileType</a> , <a href="#">DeleteJob</a> , <a href="#">DeleteJobNotices</a> , <a href="#">DeleteJobSchedule</a> , <a href="#">DeleteUser</a> , <a href="#">DeleteUserGroup</a> , <a href="#">MoveFile</a> , <a href="#">Transaction</a> , <a href="#">UndeleteUser</a> , <a href="#">UpdateFile</a> , <a href="#">UpdateFileType</a> , <a href="#">UpdateJobSchedule</a> , <a href="#">UpdateOpenSecurityCache</a> , <a href="#">UpdateUser</a> , <a href="#">UpdateUserGroup</a> , or <a href="#">UpdateVolumeProperties</a> . Specifies a set of operations and their execution order for an Administate operation. AdminOperation is a set of complex data types which details how to create, delete, update, copy, and move items within a volume. Each AdminOperation element represents a single unit of work within an Administate operation. Only a volume administrator or a user in the |

Administrator role uses these operations. The valid AdminOperation values are listed and described in Table 13-2.

**Table 13-2** Administrative operations

| Value                   | Purpose                                                                                                                                  |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| CopyFile                | Copies a file or folder in the working directory to a specified target directory.                                                        |
| CreateFileType          | Creates a new file type in the volume.                                                                                                   |
| CreateFolder            | Creates a folder in a volume.                                                                                                            |
| CreateUser              | Creates a user in the volume.                                                                                                            |
| CreateUserGroup         | Creates a user group.                                                                                                                    |
| DeleteFile              | Deletes files or folders from the volume.                                                                                                |
| DeleteFileType          | Deletes file types from the volume.                                                                                                      |
| DeleteJob               | Deletes one or more jobs.                                                                                                                |
| DeleteJobNotices        | Deletes one or more job notices.                                                                                                         |
| DeleteJobSchedule       | Deletes a job schedule.                                                                                                                  |
| DeleteUser              | Deletes one or more users.                                                                                                               |
| DeleteUserGroup         | Deletes one or more user groups.                                                                                                         |
| MoveFile                | Moves a file or folder from the working directory to a specified target directory in the volume.                                         |
| Transaction             | A packaging mechanism for Administate operations. If a failure occurs anywhere in a transaction, all operations in the transaction fail. |
| UndeleteUser            | Restores a previously deleted user.                                                                                                      |
| UpdateFile              | Updates file or folder properties in the volume.                                                                                         |
| UpdateFileType          | Updates file type properties in the volume.                                                                                              |
| UpdateJobSchedule       | Updates a job schedule.                                                                                                                  |
| UpdateOpenSecurityCache | Flushes the volume's open security cache and retrieves new data from an external security source.                                        |
| UpdateUser              | Updates user properties.                                                                                                                 |
| UpdateUserGroup         | Updates user group properties.                                                                                                           |
| UpdateVolumeProperties  | Updates the properties of a specific volume.                                                                                             |

---

## CallOpenSecurityLibrary

Calls the Report Server Security Extension (RSSE) API AcRSSEPassThrough function or PassThrough message, which calls the RSSE for general purposes. The application then interprets the value AcRSSEPassThrough or PassThrough returns, along with the return code. The RSSE library registered with BIRT iHub determines the returned value.

**Request element**

**InputParameter**  
**String**. The input parameter string.

**Response elements**

**OutputParameter**  
**String**. The output parameter.

**ReturnCode**  
**Int**. The return code.

---

## CancelJob

Terminates a job.

**Request element**

**JobId**  
**String**. The id of the job to be cancelled.

**Response elements**

**Status**  
**CancelJobStatus**. The status of the cancelled job.

**ErrorDescription**

**String**. An error message regarding the cancelled job.

---

## CancelReport

Stops synchronous report execution. Synchronous report execution can be cancelled only after the connection handle is received. ConnectionHandle is a session ID of the object.

The job submitter can cancel a report. Only the system administrator can cancel or get the information about the report of another user.

**Request element**

**ObjectId**  
**String**. The object ID of the report to cancel.

**Response elements**

**Status**  
**String**. The status of the request. One of the following values:

- Succeeded

The synchronous report generation was successfully cancelled.

- Failed  
The request failed.
- InActive  
The synchronous report generation is complete and cannot be cancelled.

**ErrorDescription**

**String**. A description of any error that occurred.

---

## CloseInfoObject

Closes an information object.

|                        |                                                                                 |
|------------------------|---------------------------------------------------------------------------------|
| <b>Request element</b> | <b>DataFetchHandle</b><br><b>String</b> . The handle to the information object. |
|------------------------|---------------------------------------------------------------------------------|

---

## CreateDatabaseConnection

Obsolete since iHub Release 2.

---

## CreateOCache

Obsolete since iHub Release 2.

---

## CreateParameterValuesFile

Creates a report object value (.rov) file. To create the ROV based on specified parameters, specify ParameterList. To create the ROV based on an executable file, specify BasedOnFileName or BasedOnFileDialog. To create the ROV based on another ROV, specify ParameterFile. If you create the ROV based on either an executable or ROV, all parameters must be defined in the based-on file.

CreateParameterValuesFile ignores parameters not defined in the based-on file.

|                         |                                                                                                                                                          |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>BasedOnFileName</b><br><b>String</b> . The name of the executable file on which to base the ROV. Specify either BasedOnFileName or BasedOnFileDialog. |
|                         | <b>BasedOnFileDialog</b><br><b>String</b> . The ID of the executable file on which to base the ROV. Specify either BasedOnFileDialog or BasedOnFileName. |

---

## CreateQuery

### ParameterFile

[NewFile](#). The existing ROV on which to base the ROV.

### ParameterValueList

[ArrayOfParameterValue](#). The list of parameters on which to base the ROV.

### FileProperties

[ArrayOfString](#). The file properties to return.

### Response element

#### ParameterValuesFile

[File](#). The ROV attributes.

---

## CreateQuery

Obsolete since iHub Release 2.

---

## CreateResourceGroup

Creates a resource group and specifies its properties.

### Request elements

#### ResourceGroup

[ResourceGroup](#). The resource group details.

#### ResourceGroupSettingsList

[ArrayOfResourceGroupSettings](#). The resource group settings.

---

## CubeExtraction

Extracts data from a specified data cube object.

### Request elements

#### Object

[ObjectIdentifier](#). The ID of the object from which to extract the data.

#### Component

[ArrayOfNameValuePair](#). The name, display name, or ID, and the value of the component from which to retrieve the data.

#### Properties

[ArrayOfNameValuePair](#). The properties to retrieve.

#### Columns

[ArrayOfString](#). The list of column names.

#### FilterList

[ArrayOfDataFilterCondition](#). The list of available filters.

|                          |                                                                                                                                                                                         |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          | <b>SortColumnList</b><br><a href="#">ArrayOfDataSortColumn</a> . The list of columns on which to sort the query.                                                                        |
| <b>Response elements</b> | <b>ResultSetSchema</b><br><a href="#">ResultSetSchema</a> . The complex data type that describes the result set schema.                                                                 |
|                          | <b>DataExtractionRef</b><br>The reference to the complex data type that describes the object in the attachment <a href="#">Attachment</a> , and contains the attachment as binary data. |
|                          | <b>ConnectionHandle</b><br><a href="#">String</a> . The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.            |

## DataExtraction

Extracts data from a specified object. DataExtraction does not support extraction from multiple components. If multiple components are specified, DataExtraction only extracts the data of the last component.

|                          |                                                                                                                                                                                         |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b>  | <b>Object</b><br><a href="#">ObjectIdentifier</a> . The ID of the object from which to extract the data.                                                                                |
|                          | <b>Component</b><br><a href="#">ArrayNameValuePair</a> . The name, display name, or ID, and the value of the component from which to retrieve the data.                                 |
|                          | <b>Properties</b><br><a href="#">ArrayNameValuePair</a> . The properties to retrieve.                                                                                                   |
|                          | <b>Columns</b><br><a href="#">ArrayString</a> . The list of column names.                                                                                                               |
|                          | <b>FilterList</b><br><a href="#">ArrayOfDataFilterCondition</a> . The list of available filters.                                                                                        |
|                          | <b>SortColumnList</b><br><a href="#">ArrayOfDataSortColumn</a> . The list of columns on which to sort the query.                                                                        |
|                          | <b>StartRowNumber</b><br><a href="#">Int</a> . The row number from which to start data extraction.                                                                                      |
| <b>Response elements</b> | <b>ResultSetSchema</b><br><a href="#">ResultSetSchema</a> . The complex data type that describes the result set schema.                                                                 |
|                          | <b>DataExtractionRef</b><br><a href="#">Attachment</a> . The reference to the complex data type that describes the object in the attachment and contains the attachment as binary data. |

## DeleteDatabaseConnection

### **ConnectionHandle**

**String.** The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

---

## DeleteDatabaseConnection

Obsolete since iHub Release 2.

---

## DeleteOCache

Obsolete since iHub Release 2.

---

## DeleteResourceGroup

Deletes a resource group. You cannot delete a default resource group. If a scheduled job is assigned to a resource group that you delete, the job remains in a pending state when BIRT iHub runs the job. If job is running on a Factory assigned to a resource group that you delete, the job completes.

**Element** **Name**

**String.** The name of the resource group to delete.

---

## DownloadFile

Downloads a file from a volume. An attachment is included in the response packet, which refers to file content. The file content is streamed back using SOAP attachment.

**Request elements** **FileName**

**String.** The name of the file to download. Specify either FileName or FileId.

**FileId**

**String.** The ID of the file to download. Specify either FileId or FileName.

**DecomposeCompoundDocument**

**Boolean.** Specifies whether to download a compound document as one file or multiple attachments. If False, you can download the file as a single file. If True, and the file is a compound document, BIRT iHub splits the file into several attachments. The default value is False.

**DownloadEmbedded**

**Boolean.** Specifies whether to send the attachment embedded in the body of the SOAP message or in a separate MIME boundary. If True, the response contains the attachment as embedded data. If False, the attachment is in a separate MIME boundary. The default value is False.

**FileProperties**

**ArrayOfString.** The file properties to return.

**Response elements****File**

**File.** The file properties.

**Content**

**Attachment.** The downloaded file in an embedded or chunked file operation.

**ContainedFiles**

**ArrayOfAttachment.** The downloaded set of files in a decomposed compound document operation.

## DownloadTransientFile

Obsolete since iHub Release 2.

## ExecuteQuery

Obsolete since iHub Release 2.

## ExecuteReport

Triggers the execution of a report in synchronous mode. If you specify WaitTime, BIRT iHub sends a response within the specified time. Otherwise, BIRT iHub sends a response when the request is complete or, if progressive viewing is enabled, when the first page is complete. Specify FileType in the SOAP header for all execute, submit, and view IDAPI requests.

**Request elements****JobName**

**String.** The name of the job.

**InputFileName**

**String.** The name of the input executable file. Specify either InputFileName or InputFileId.

## ExecuteReport

### **InputFileDialog**

[String](#). The ID of the input executable file. Specify either InputFileDialog or InputFileName.

### **InputFile**

[Attachment](#). Specifies that the input executable file is an attachment in the response. Valid only if the value of SaveOutputFile is False.

### **OutputFormat**

[String](#). The display format for the output file.

### **SaveOutputFile**

[Boolean](#). Specifies whether to use the RequestedOutputFile setting. If False, BIRT iHub ignores RequestedOutputFile. If True and RequestedOutputFile is missing, BIRT iHub reports an error.

### **RequestedOutputFile**

[NewFile](#). The name to use for the output file. Required for persistent jobs.

### **ParameterValues**

[ArrayOfParameterValue](#). The array of parameter values choice with which to overwrite the default parameter values:

- ParameterValuesFileName  
The name of the report object value (.rov) file to create. If specified, BIRT iHub creates a persistent ROV. Otherwise, BIRT iHub creates a temporary ROV. Specify either ParameterValuesFileName or ParameterValuesId.
- ParameterValuesId  
The ID of the ROV to create. If specified, BIRT iHub creates a persistent ROV. Otherwise, BIRT iHub creates a temporary ROV. Specify either ParameterValuesId or ParameterValuesFileName.

### **IsBundled**

[Boolean](#). Specifies whether the report is bundled. If True, the report is bundled. If False, the report is not bundled. The default value is False.

### **OpenServerOptions**

[OpenServerOptions](#). The open server options to use. Valid options are:

- KeepWorkingSpace
- DriverTimeout
- PollingInterval

### **ProgressiveViewing**

[Boolean](#). Specifies whether to enable progressive viewing. True enables progressive viewing. The default value is True.

**WaitTime**

**Int.** The number of seconds BIRT iHub waits before sending a response to the report generation request. Use WaitTime to provide the ability to cancel a synchronous report generation request. The default value is 150 seconds.

**Response elements****Status**

**ExecuteReportStatus.** Indicates the execution status of a synchronous job. One of the following values:

- Done
- Failed
- FirstPage
- Pending

**ErrorDescription**

**String.** The description of the error. Returned if Status is Failed.

**OutputFileType**

**String.** Sets the file type for the output file.

**ObjectId**

**String.** The object ID of the report.

**ConnectionHandle**

**String.** The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

## ExecuteVolumeCommand

Executes predefined volume control commands.

**Request elements****VolumeName**

**String.** The volume on which to execute the commands.

**Command**

**String.** One or more of the following commands to execute:

- StartPartitionPhaseOut  
Starts the storage location phase out.
- StartArchive  
Starts an archive pass.
- StopArchive  
Stops an archive pass if one is currently running. If an archive pass is not currently running, this command returns a Failed status. This command is asynchronous. This means that the call returns without waiting for the archive

## ExtractParameterDefinitionsFromFile

pass to stop. To find out the status of the archive pass after sending this command, use GetVolumeProperties.

Only a user with the Operator or Administrator role can issue this command.

|                          |                                                                                                                                 |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>Response elements</b> | <b>Status</b>                                                                                                                   |
|                          | <a href="#">String</a> . The status of command execution. One of the following values:                                          |
|                          | <ul style="list-style-type: none"><li>■ Succeeded<br/>The command succeeded.</li><li>■ Failed<br/>The command failed.</li></ul> |

---

## ExtractParameterDefinitionsFromFile

Retrieves the parameter definitions from the specified file.

|                         |                                                                     |
|-------------------------|---------------------------------------------------------------------|
| <b>Request element</b>  | <b>Content</b>                                                      |
|                         | <a href="#">Attachment</a> . The parameter definitions to retrieve. |
| <b>Response element</b> | <b>ParameterList</b>                                                |

[ArrayOfParameterDefinition](#). The requested parameter definitions.

---

## ExportParameterDefinitionsToFile

Exports parameter definitions associated with the specified file to a new file.

|                         |                                                                                                                                                                                                                                                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>ParameterList</b>                                                                                                                                                                                                                                                                             |
|                         | <a href="#">ArrayOfParameterDefinition</a> . The parameters to export.                                                                                                                                                                                                                           |
|                         | <b>DownloadEmbedded</b>                                                                                                                                                                                                                                                                          |
|                         | <a href="#">Boolean</a> . Specifies whether to send the attachment embedded in the body of the SOAP message or in a separate MIME boundary. If True, the response contains the attachment as embedded data. If False, the attachment is in a separate MIME boundary. The default value is False. |

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| <b>Response element</b> | <b>Content</b>                                                   |
|                         | <a href="#">Attachment</a> . The exported parameter definitions. |

---

## FetchInfoObjectData

Obsolete since iHub Release 2.

---

## GetAllCounterValues

Retrieves the values of all counters.

|                         |                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>CounterInfoList</b><br><a href="#">ArrayOfCounterInfo</a> . The values of all counters. |
|-------------------------|--------------------------------------------------------------------------------------------|

---

## GetBookmarks

Retrieves the bookmarks in a report design.

|                         |                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------|
| <b>Request element</b>  | <b>Object</b><br><a href="#">ObjectIdentifier</a> . The object from which to retrieve the bookmarks. |
| <b>Response element</b> | <b>BookmarkList</b><br><a href="#">ArrayOfBookMark</a> . The array of bookmarks                      |

---

## GetCapabilities

Requests the array of capabilities available for a user group.

|                         |                                                                                                |
|-------------------------|------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>UserGroupName</b><br><a href="#">String</a> . The user group name.                          |
|                         | <b>UserGroupId</b><br><a href="#">String</a> . The user group ID.                              |
|                         | <b>UserName</b><br><a href="#">String</a> . The user name.                                     |
|                         | <b>UserId</b><br><a href="#">String</a> . The user ID.                                         |
|                         | <b>Categories</b><br><a href="#">ArrayOfString</a> . The Categories list.                      |
|                         | <b>DirectOnly</b><br><a href="#">Boolean</a> . Indicates whether capabilities are direct only. |
| <b>Response element</b> | <b>Capabilities</b><br>The array of Capabilities.                                              |

---

## GetCapabilitiesByCategory

Requests the array of capabilities available for a user group category.

## GetCapabilityCategories

|                         |                                                                               |
|-------------------------|-------------------------------------------------------------------------------|
| <b>Request element</b>  | <b>Category</b><br><a href="#">String</a> . The category name.                |
| <b>Response element</b> | <b>Capabilities</b><br><a href="#">ArrayOfString</a> . The capabilities list. |

---

## GetCapabilityCategories

Gets the list of capability categories.

|                         |                                                                           |
|-------------------------|---------------------------------------------------------------------------|
| <b>Request element</b>  | <b>GetCapabilityCategories</b><br>The GetCapabilityCategories request.    |
| <b>Response element</b> | <b>Categories</b><br><a href="#">ArrayOfString</a> . The categories list. |

---

## GetChannelACL

Obsolete since iHub Release 3.

---

## GetConnectionProperties

Retrieves the connection properties for a file, including user, role, or user group lists.

|                         |                                                                                               |
|-------------------------|-----------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>FileName</b><br><a href="#">String</a> . The name of the file.                             |
|                         | <b>FileId</b><br><a href="#">String</a> . The file ID of the file.                            |
|                         | <b>UserNames</b><br><a href="#">String</a> . The user names attached to the file.             |
|                         | <b>UserId</b><br><a href="#">String</a> . The user ID attached to the file.                   |
|                         | <b>RoleName</b><br><a href="#">String</a> . The role names attached to the file.              |
|                         | <b>RoleId</b><br><a href="#">String</a> . The role ID attached to the file.                   |
|                         | <b>UserGroup Names</b><br><a href="#">String</a> . The user group names attached to the file. |

|                         |                                                                                                                                                                |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UserGroupId</b>      | <b>String</b> . The user group ID attached to the file.                                                                                                        |
| <b>Effective</b>        | <b>Boolean</b> . The user group ID attached to the file.                                                                                                       |
| <b>Response element</b> | <b>ConnectionProperties</b><br><b>ArrayOfPropertyValue</b> . The list of user, role, and user groups attached to the file, and which properties are effective. |

---

## GetConnectionPropertyAssignees

Retrieves the users and user groups for a file.

|                          |                                                                                                |
|--------------------------|------------------------------------------------------------------------------------------------|
| <b>Request elements</b>  | <b>FileName</b><br><b>String</b> . The name of the file.                                       |
|                          | <b>FileId</b><br><b>String</b> . The file ID of the file.                                      |
| <b>Response elements</b> | <b>UserNames</b><br><b>ArrayOfString</b> . The user names associated with the file.            |
|                          | <b>RoleNames</b><br><b>ArrayOfString</b> . The roles names associated with the file.           |
|                          | <b>UserGroupNames</b><br><b>ArrayOfString</b> . The user group names associated with the file. |

---

## GetContent

Obsolete since iHub Release 2.

---

## GetCounterValues

Retrieves the IDs, names, and values of specific counters.

|                         |                                                                                                            |
|-------------------------|------------------------------------------------------------------------------------------------------------|
| <b>Request element</b>  | <b>CounterIDList</b><br><b>ArrayOfLong</b> . The list of counters for which to retrieve information.       |
| <b>Response element</b> | <b>CounterInfoList</b><br><b>ArrayOfCounterInfo</b> . The IDs, names, and values of the specified counters |

## GetCubeMetaData

Retrieves cube metadata describing a result set schema.

**Request elements**

**Object**

[ObjectIdentifier](#). The object from which to retrieve the metadata.

**Component**

[ArrayOfNameValuePair](#). The name, display name, or ID, and the value of the component from which to retrieve the metadata.

**Response element**

**ArrayOfResultSetSchema**

[ArrayOfResultSetSchema](#). The complex data type that represents an array of ResultSetSchema objects.

---

## GetCustomFormat

Obsolete since iHub Release 2.

---

## GetDatabaseConnectionDefinition

Obsolete since iHub Release 2.

---

## GetDatabaseConnectionParameters

Obsolete since iHub Release 2.

---

## GetDatabaseConnectionTypes

Obsolete since iHub Release 2.

---

## GetDataExtractionFormats

Retrieves a list of DataExtractionFormat objects for a specific file type. These objects consist of an output format and a mime type.

**Request elements**

**FileType**

[String](#). The name of the file type about which to retrieve information.

|                         |                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>DataExtractionFormats</b><br><a href="#">ArrayOfDataExtractionFormat</a> . The list of DataExtractionFormat objects. |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------|

---

## GetDocumentConversionOptions

Retrieves a list of DocumentConversionOptions. A document conversion option includes a file type, output format, mime type and parameter definitions.

|                         |                                                                                                                                                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>FileType</b><br><a href="#">String</a> . The file type about which to retrieve the conversion options. When FileType is not set, the response includes options for all the Java report documents. In this case, GetDocumentConversionOptions ignores OutputFormat. |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

When a FileType is set, GetDocumentConversionOptions returns the options for the conversion to the specified OutputFormat.

**OutputFormat**

[String](#). The display format about which to retrieve the conversion options. When OutputFormat is not specified, the response includes parameters for all output formats.

If the FileType or OutputFormat fields specify unsupported formats, GetDocumentConversionOptions returns a SOAP fault.

|                         |                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>ConversionOptions</b><br><a href="#">ArrayOfDocumentConversionOptions</a> . The list of DocumentConversionOptions. |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------|

---

## GetDynamicData

Obsolete since iHub Release 2.

---

## GetEmbeddedComponent

Obsolete since iHub Release 2.

---

## GetFactoryServiceInfo

Retrieves general information about a Factory service. The node name is specified in the TargetServer element of the SOAP header.

GetFactoryServiceInfo is available only to a BIRT iHub System administrator. To log in as a BIRT iHub System administrator, use SystemLogin.

## GetFactoryServiceInfo

To retrieve a list of servers for which you can obtain information, use GetSystemServerList.

**Request element** **GetFactoryServiceInfo**

Information about the Factory service.

**Response elements** **ServerName**

**String**. The node for which information is returned.

**PendingSyncJobs**

**Long**. The number of synchronous jobs in queue.

**SyncJobQueueSize**

**Long**. The maximum number of synchronous jobs allowed in the queue.

**RunningSyncJobs**

**Long**. The number of synchronous jobs currently running.

**RunningJobs**

**Long**. The total number of jobs currently running.

**SyncFactoryProcesses**

**Long**. The number of Factories reserved for running synchronous jobs.

**MaxFactoryProcesses**

**Long**. The maximum number of Factories that can run on the system.

**TransientReportCacheSize**

**Long**. The maximum disk space available for transient reports, in megabytes.

**PercentTransientReportCacheInUse**

**Long**. The currently used percentage of disk space available for transient reports.

**CurrentTransientReportTimeout**

**Long**. The number of minutes after which transient reports are deleted from the synchronous cache adjusted according to disk space currently available in the synchronous cache.

**TransientReportTimeout**

**Long**. Time after which transient reports are deleted from the synchronous cache, in minutes.

**SyncJobQueueWait**

**Long**. The maximum time a job remains in the synchronous queue, in seconds.

**MaxSyncJobRuntime**

**Long**. The maximum job execution time, in seconds.

---

## GetFactoryServiceJobs

Retrieves information about pending synchronous jobs and all running jobs on the node. The node is specified in the TargetServer element of the SOAP header.

For pending synchronous jobs, GetFactoryServiceJobs always returns the following information in addition to any values you specify:

- ConnectionHandle
- ObjectId
- IsTransient
- Volume

For running jobs, GetFactoryServiceJobs always returns the following information in addition to any values you specify:

- IsSyncJob
- Volume
- ConnectionHandle
- ObjectId
- IsTransient for synchronous jobs or JobId for asynchronous jobs

To retrieve all information, specify All.

GetFactoryServiceJobs is available only to a BIRT iHub System administrator. To log in as a BIRT iHub System administrator, use SystemLogin.

**Request elements****PendingSyncJobsResultDef**

[ArrayOfString](#). Requests the following information about pending synchronous jobs:

- Default  
ConnectionHandle, ObjectId, IsTransient, and Volume
- All  
All information
- ServerName  
The node on which the job originated
- Owner  
The name of the user who submitted the job
- ExecutableFileName  
The fully qualified name of the executable file

## GetFactoryServiceJobs

- ExecutableVersionNumber  
The fully qualified version number of the executable file
- ExecutableVersionName  
The fully qualified version name of the executable file
- SubmissionTime  
The time the job was submitted to the server
- QueueTimeout  
The number of seconds remaining before the job is deleted from the queue
- QueuePosition  
The job's position in the queue

## RunningJobsResultDef

[ArrayOfString](#). Requests the following information about all running jobs:

- Default  
IsSyncJob, Volume, ConnectionHandle, ObjectId, IsTransient for synchronous jobs or JobId for asynchronous jobs.
- All  
All information.
- ServerName  
The node on which the job originated.
- Owner  
The name of the user who submitted the job.
- ExecutableFileName  
The fully qualified name of the executable file.
- ExecutableVersionNumber  
The fully qualified version number of the executable file.
- ExecutableVersionName  
The fully qualified version name of the executable file.
- SubmissionTime  
The time the job was submitted to the server.
- StartTime  
The time at which the job execution started.
- ExecutionTimeout  
The number of seconds remaining before the job execution expires. Always zero (infinite) for asynchronous reports.

- **IsSyncFactory**  
True if the Factory is running synchronous jobs, False if the Factory is running asynchronous jobs.
- **FactoryPid**  
The Process ID of the Factory.

|                          |                                                                                                                                                                                                                                            |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Response elements</b> | <b>PendingSyncJobs</b><br><a href="#">ArrayOfPendingSyncJob</a> . Information about pending synchronous jobs on the node.<br><br><b>RunningJobs</b><br><a href="#">ArrayOfRunningJob</a> . Information about all running jobs on the node. |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## GetFileACL

Retrieves the ACL of the specified file or folder.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>FileId</b><br><a href="#">String</a> . The ID of the file or folder for which to retrieve the ACL. Specify either FileId or FileName.<br><br><b>FileName</b><br><a href="#">String</a> . The full name of the file or folder for which to retrieve the ACL. Specify either FileName or FileId.<br><br><b>GrantedUserId</b><br><a href="#">String</a> . The user ID.<br><br><b>GrantedUserName</b><br><a href="#">String</a> . The user name.<br><br><b>GrantedRoleId</b><br><a href="#">String</a> . The role ID.<br><br><b>GrantedRoleName</b><br><a href="#">String</a> . The role name.<br><br><b>GrantedUserGroupId</b><br><a href="#">String</a> . The user group ID.<br><br><b>GrantedRoleName</b><br><a href="#">String</a> . The user group name.<br><br><b>FetchSize</b><br><a href="#">Int</a> . The maximum number of records to retrieve and return in a result set. The default value is 500. |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## GetFileCreationACL

### FetchDirection

**Boolean.** If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

### CountLimit

**Int.** The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

### FetchHandle

**String.** Retrieves more items from the result set. In the second and subsequent calls for data, specify the same search criteria as in the original call.

#### Response elements

##### ACL

**ArrayOfPermission.** The ACL of the file or folder.

##### FetchHandle

**String.** Indicates that the number of items in the result set exceeds the FetchSize limit.

##### TotalCount

**Int.** The number of entries in the search result set.

---

## GetFileCreationACL

Retrieves the specified user FileCreationACL. The FileCreationACL is the template applied to all new files the user creates.

#### Request elements

##### CreatedByUserName

**String.** The name of the user whose template to retrieve. Specify either CreatedByUserName or CreatedById.

##### CreatedByUserId

**String.** The ID of the user whose template to retrieve. Specify either CreatedById or CreatedByUserName.

##### GrantedUserId

**String.** The user ID. Specify either GrantedUserId or GrantedUserName.

##### GrantedUserName

**String.** The user name. Specify either GrantedUserName or GrantedUserId.

##### GrantedRoleId

**String.** The role ID. Specify either GrantedRoleId or GrantedRoleName.

##### GrantedRoleName

**String.** The role name. Specify either GrantedRoleName or GrantedRoleId.

**FetchSize**

**Int.** The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

**Boolean.** If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

**Int.** The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

**String.** Retrieves more items from the result set. In the second and subsequent calls for data, specify the same search criteria as in the original call.

**Response elements****ACL**

**ArrayOfPermission.** The user ACL.

**FetchHandle**

**String.** Indicates that the number of items in the result set exceeds the FetchSize limit.

**TotalCount**

**Int.** The number of entries in the search result set.

## GetFileDetails

Retrieves the properties of the specified file.

**Request elements****FileName**

**String.** The full name of the file for which to retrieve properties. Specify either FileName or FileId.

**FileId**

**String.** The ID of the file for which to retrieve properties. Specify either FileId or FileName.

**ResultDef**

**ArrayOfString.** The properties to retrieve. The file properties are always returned. In addition, you can specify one or more of the following:

- **ACL**  
The access control list (ACL) of the file
- **ArchiveRules**  
The archive rules of the file

## GetFileTypeParameterDefinitions

- **AccessType**  
The access rights to the file, private or shared
- Response elements** **File**  
[File](#). The file properties.
- ACL**  
[ArrayOfPermission](#). The ACL. Returned only if ACL is specified in ResultDef.
- ArchiveRules**  
[ArrayOfArchiveRule](#). The archive rules. Returned only if ArchiveRules is specified in ResultDef.
- 

## GetFileTypeParameterDefinitions

Retrieves parameters of the specified file type on the BIRT iHub to which the user is logged in.

- Request element** **FileType**  
[String](#). The name of the file type for which to retrieve information.
- Response element** **ParameterList**  
[ArrayOfParameterDefinition](#). The list of parameters.
- 

## GetFolderItems

Retrieves all specified objects in a specified folder, such as all files or folders, a list of files or folders, or all users.

To search all files or folders that match the specified conditions, specify Search.

- Request elements** **FolderName**  
[String](#). The full path and the name of the folder from which to retrieve objects. Specify either FolderName or FolderId.
- FolderId**  
[String](#). The ID of the folder from which to retrieve objects. Specify either FolderId or FolderName.
- ResultDef**  
[ArrayOfString](#). The properties to retrieve. By default, the Id and Name are always returned. In addition, you can specify the following properties:
- Description
  - FileType
  - Owner
-

- PageCount
- Size
- TimeStamp
- Version
- VersionName
- UserPermissions

**LatestVersionOnly**

**Boolean.** Specifies whether only the latest version is returned. If True, only the latest version is returned. The default value is False.

**Search**

The search condition. If conditions apply to multiple fields, use ConditionArray. [FileSearch](#).

**Response elements****ItemList**

[ArrayOfFile](#). The objects matching the search criteria.

**FetchHandle**

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

**TotalCount**

[Int](#). The number of entries in the search result set.

## GetFormats

Obsolete since iHub Release 2.

## GetInfoObject

Obsolete since iHub Release 2.

## GetJavaReportEmbededComponent

Retrieves an embedded component in the document such as an image or a graph.

**Request elements****Object**

[ObjectIdentifier](#). The ID of the object from which to retrieve an embedded component in a document.

## GetJavaReportTOC

### **Component**

[ArrayOfNameValuePair](#). The name, display name, or ID, and the value of the component to retrieve.

### **Attributes**

[ArrayOfNameValuePair](#). Currently not used by BIRT reports.

### **DownloadEmbedded**

[Boolean](#). Specifies whether to send the attachment embedded in the body of the SOAP message or in a separate MIME boundary. If True, the response contains the attachment as embedded data. If False, the attachment is in a separate MIME boundary. The default value is False.

### **Response elements**

#### **EmbeddedRef**

[Attachment](#). Contains the following properties of the attachment:

- [MimeType](#)
- [ContentEncoding](#)
- [ContentLength](#)
- [Locale](#)

#### **ConnectionHandle**

[String](#). The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

---

## GetJavaReportTOC

Retrieves the table of contents (TOC) of the document.

### **Request elements**

#### **Object**

[ObjectIdentifier](#). The ID of the object from which to retrieve the TOC.

### **DownloadEmbedded**

[Boolean](#). Specifies whether to send the attachment embedded in the body of the SOAP message or in a separate MIME boundary. If True, the response contains the attachment as embedded data. If False, the attachment is in a separate MIME boundary. The default value is False.

### **Component**

[ArrayOfNameValuePair](#). The name, display name, or ID, and the value of the component from which to retrieve the TOC.

### **Recursive**

[Boolean](#). Specifies whether to search subfolders. If True, the search includes subfolders. The default value is False.

| Response elements | TocRef                                                                                                                                     |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
|                   | <b>Attachment</b> . The details of the attachment such as contentId, contentType, contentLength, contentEncoding, locale, and contentData. |

**ConnectionHandle**

**String**. The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

---

## GetJobDetails

Retrieves the properties of a specified job.

| Request elements | JobId                                                                |
|------------------|----------------------------------------------------------------------|
|                  | <b>String</b> . The ID of the job from which to retrieve properties. |

**ResultDef**

**ArrayOfString**. The properties to retrieve. You can specify the following properties:

- JobAttributes  
The general job properties
- InputDetail  
The job input parameters
- Schedules  
The job schedule information
- PrinterOptions  
The printer settings, if available
- NotifyUsers  
The names of users to receive notifications about the job
- NotifyGroups  
Obsolete since iHub Release 3
- NotifyChannels  
Obsolete since iHub Release 3
- DefaultOutputFileACL  
The output file ACL templates
- Status  
The job status
- ReportParameters

## GetJobDetails

The report parameters from the report parameters value file associated with the job

- **ResourceGroup**

The name of the resource group to which the job is assigned

**GroupingEnabled**

Obsolete since iHub Release 2.

**SupportedQueryFeatures**

Obsolete since iHub Release 2.

**Response elements**

**JobAttributes**

[JobProperties](#). The general job attributes.

**InputDetail**

[JobInputDetail](#). The job input parameters.

**Schedules**

[JobSchedule](#). The job schedule information.

**PrinterOptions**

[PrinterOptions](#). The job printer settings.

**NotifyUsers**

[ArrayOfString](#). The names of users to receive notifications about the job.

**NotifyGroups**

Obsolete since iHub Release 3.

**NotifyChannels**

Obsolete since iHub Release 3.

**DefaultOutputFileACL**

[ArrayOfPermission](#). The output file access control list (ACL) templates.

**Status**

[String](#). The job status.

**ReportParameters**

[ArrayOfParameterValue](#). The report parameters associated with the job.

**Query**

Obsolete since iHub Release 2.

**OutputFileAccessType**

[FileAccess](#). The access rights to the output file. One of the following values:

- **Private**

Only the owner of the file and an administrator can access the file.

- **Shared**

All users and roles specified in the access control list (ACL) for the file can access the file.

**WaitForEvent**

**Event**. An event that must complete before processing the response.

---

## GetMetaData

Retrieves the metadata describing a result set schema.

**Request elements****Object**

**ObjectIdentifier**. The object from which to retrieve the metadata.

**Component**

**ArrayOfNameValuePair**. The name, display name, or ID, and the value of the component from which to retrieve the metadata.

**Response element****ArrayOfResultSetSchema**

**ArrayOfResultSetSchema**. The complex data type that represents an array of ResultSetSchema objects.

---

## GetNoticeJobDetails

Retrieves the properties of the specified job notice.

**Request elements****JobId**

**String**. The ID of the job from which to retrieve properties.

**ResultDef**

**ArrayOfString**. The properties to retrieve. You can specify the following properties:

- InputDetail  
The job input parameters
- Schedules  
The job schedule information
- PrinterOptions  
The printer settings, if available
- NotifyUsers  
The names of users to receive notifications about the job
- NotifyGroups  
Obsolete since iHub Release 3

## GetNoticeJobDetails

- **NotifyChannels**  
Obsolete since iHub Release 3
- **DefaultOutputFileACL**  
The output file access control list (ACL) templates
- **Status**  
The job status
- **ReportParameters**  
The report parameters from the report object value (.rov) file associated with the job
- **ResourceGroup**  
The name of the resource group to which the job is assigned

### **NotifiedChannelId**

Obsolete since iHub Release 3.

### **NotifiedChannelName**

Obsolete since iHub Release 3.

### **GroupingEnabled**

Obsolete since iHub Release 2.

### **SupportedQueryFeatures**

Obsolete since iHub Release 2.

## **Response elements**

### **JobAttributes**

[JobProperties](#). The general job attributes.

### **InputDetail**

[JobInputDetail](#). The job input parameters.

### **Schedules**

[JobSchedule](#). The job schedule information.

### **PrinterOptions**

[JobPrinterOptions](#). The job printer settings.

### **NotifyUsers**

[ArrayOfString](#). The names of users to receive notifications about the job.

### **NotifyGroups**

Obsolete since iHub Release 3.

### **NotifyChannels**

Obsolete since iHub Release 3.

### **DefaultOutputFileACL**

[ArrayOfPermission](#). The output file access control list (ACL) templates.

**Status**

[String](#). The job status.

**ReportParameters**

[ArrayOfParameterValue](#). The report parameters from report object value (.rov) file associated with the job.

**OutputFileAccessType**

[FileAccess](#). The access rights to the output file. One of the following values:

- Private  
Only the owner of the file and an administrator can access the file.
- Shared  
All users and roles specified in the ACL for the file can access the file.

**WaitForEvent**

[Event](#). An event that must be completed before the response is processed.

## GetPageCount

Retrieves the number of pages in a report in a report generated by iHub.

**Request element****Object**

[ObjectIdentifier](#). The object from which to retrieve page count.

**Response elements****PageCount**

[String](#). The number of pages. If the report was uploaded to the volume and not generated by iHub, PageCount is 0.

**IsReportCompleted**

[Boolean](#). True if report generation is complete, False otherwise.

**ConnectionHandle**

[String](#). The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

## GetPageNumber

Retrieves the page number of a bookmark component in a report.

**Request elements****Object**

[ObjectIdentifier](#). The object from which to retrieve the page number.

## GetParameterPickList

### Component

[ArrayOfNameValuePair](#). The name, display name, or ID, and the value of the component from which to retrieve the page number. The page number is a bookmark value in a report.

### Response element

#### PageNumber

[String](#). The component element specifies ArrayOfNameValuePair as the data type. If multiple valid bookmarks are present inside a report, a response only returns the page number for first bookmark. Actuate IDAPI does not support getting the page number for multiple components. If you specify multiple components, the response only returns the page number for the first component.

---

## GetParameterPickList

Retrieves the parameters names from a pick list in a report.

### Request elements

#### Object

[ObjectIdentifier](#). The identifier of the object from which to retrieve the parameter pick list.

#### CascadingGroupName

[String](#). The cascading group name in the pick list containing the target list.

#### ParameterName

[String](#). The name of the parameter. Each parameter name is unique within a report even between execution and view parameters.

#### PrecedingParameterValues

[ArrayOfParameterValue](#). The values of the parameters that precede the specified parameter.

#### Filter

[String](#). A string prefix to be applied to the overall selection list.

#### startIndex

[Long](#). The index where the fetch operation starts.

#### FetchSize

[Long](#). The maximum number of records to retrieve and return in a result set. The default value is 500.

#### CountLimit

[Long](#). The Number of entries to be counted after FetchSize is reached. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

### Response elements

#### ParameterPickList

[ArrayOfNameValuePair](#). List of parameters.

**TotalCount**

**Long.** The number of parameters in the list.

## GetReportParameters

Retrieves report parameter values. GetReportParameters requires execute or read privileges when executed against a BIRT design, BIRT data design, or BIRT 360 dashboard file. When executed against a BIRT document file, GetReportParameters requires execute, secure read, or read privileges.

**Request elements****JobId**

**String.** The ID of the job from which to retrieve the parameter values. Specify JobId to retrieve the parameter values from the report file associated with the job.

**ReportFileId**

**String.** The ID of the file from which to retrieve the parameter values. Specify ReportFileId or ReportFileName to retrieve parameter values from an executable, document, or object value file, or a third-party compound storage file.

**ReportFileName**

**String.** The name of the file from which to retrieve the parameter values. Specify ReportFileName or Report FileId to retrieve parameter values from an executable, document, or object value file, or a third-party compound storage file.

**ReportParameterType**

**ReportParameterType.** Optional parameter type can include Execution, View, and All. If not specified, only execution parameters return to maintain backward compatibility.

**WithoutDynamicPickList**

**Boolean.** If set to True, parameters will be returned without a dynamic pick list.

**Response elements****ParameterList**

**ArrayOfParameterDefinition.** The list of parameter definition values such as group, cascading parent name, name, data type, default value, display name, help text, and so forth.

**ViewParameterList**

**ArrayOfParameterDefinition.** The list of view parameter definition values such as format, user agent, scaling factor, accept encoding, view operation, path information, embedded object path, redirect path, and PDF quality.

## GetResourceGroupInfo

Retrieves information about a specific resource group.

## GetResourceGroupList

|                          |                                                                                                                                                                                                                                                                                                                                      |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request element</b>   | <b>Name</b><br><a href="#">String</a> . The name of the resource group for which to retrieve information.                                                                                                                                                                                                                            |
| <b>Response elements</b> | <b>ResourceGroup</b><br><a href="#">ResourceGroup</a> . Contains the following information about the resource group:                                                                                                                                                                                                                 |
|                          | <ul style="list-style-type: none"><li>■ Name</li><li>■ Disabled</li><li>■ Description</li><li>■ Type</li><li>■ MinPriority<br/>Applies only to an asynchronous resource group</li><li>■ MaxPriority<br/>Applies only to an asynchronous resource group</li><li>■ Reserved<br/>Applies only to a synchronous resource group</li></ul> |
|                          | <b>ResourceGroupSettingsList</b><br><a href="#">ResourceGroupSettings</a> . Contains the following information about the resource group:                                                                                                                                                                                             |
|                          | <ul style="list-style-type: none"><li>■ ServerName</li><li>■ Activate</li><li>■ MaxFactory</li><li>■ FileTypes</li></ul>                                                                                                                                                                                                             |

---

## GetResourceGroupList

Retrieves a list of resource groups available to a BIRT iHub.  
GetResourceGroupList returns two lists, one for asynchronous resource groups and one for synchronous resource groups.

|                          |                                                                                                                                                                                 |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Response elements</b> | <b>AsyncResourceGroupList</b><br><a href="#">ArrayOfResourceGroup</a> . The list of available asynchronous resource groups and the properties of each of those resource groups. |
|                          | <b>SyncResourceGroupList</b><br><a href="#">ArrayOfResourceGroup</a> . The list of available synchronous resource groups and the properties of each of those resource groups.   |

**ViewResourceGroupList**

[ArrayOfResourceGroup](#). The list of available synchronous resource groups and the properties of each of those resource groups.

**WorkUnitTypes**

[ArrayOfString](#). The list of available work unit types and properties.

---

## GetSavedSearch

Obsolete since iHub Release 2.

---

## GetServerResourceGroupConfiguration

Retrieves information about resource groups available to the specified BIRT iHub.

**Request element****ServerName**

[String](#). The name of the BIRT iHub from which to retrieve information.

**Response element****ServerResourceGroupSettingList**

[ArrayOfServerResourceGroupSetting](#). Contains the following information about each resource group:

- ResourceGroupName
- Activate
- Type
- MaxFactory
- FileTypes

---

## GetStaticData

Obsolete since iHub Release 2.

---

## GetStyleSheet

Obsolete since iHub Release 2.

---

## GetSyncJobInfo

Retrieves information about a synchronous job.

GetSyncJobInfo is available only to a BIRT iHub System administrator. To log in as a BIRT iHub System administrator, use SystemLogin.

|                        |                                                                                                              |
|------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>Request element</b> | <b>ObjectId</b><br><a href="#">String</a> . The ID of the synchronous job for which to retrieve information. |
|------------------------|--------------------------------------------------------------------------------------------------------------|

|                          |                                                                                    |
|--------------------------|------------------------------------------------------------------------------------|
| <b>Response elements</b> | <b>Status</b><br><a href="#">String</a> . The status of the job. Valid values are: |
|--------------------------|------------------------------------------------------------------------------------|

- Pending
- Running
- Completed
- Failed

**ErrorDescription**

[String](#). The description of the error. Returned if Status is Failed.

**Pending**

[PendingSyncJob](#). The properties of the pending synchronous job.

**Running**

[RunningJob](#). The properties of the running synchronous job.

---

## GetSystemMDSInfo

Retrieves the names and properties of an MDS in a cluster or stand-alone server without authenticating the client. Use GetSystemMDSInfo to route requests to an alternate MDS if the one to which the client connects fails.

|                        |                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------|
| <b>Request element</b> | <b>OnlineOnly</b><br><a href="#">Boolean</a> . If True, information is retrieved only for an online MDS. |
|------------------------|----------------------------------------------------------------------------------------------------------|

|                         |                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>MDSInfoList</b><br><a href="#">ArrayOfMDSInfo</a> . The information about each MDS. |
|-------------------------|----------------------------------------------------------------------------------------|

---

## GetSystemPrinters

Retrieves all system printer information on the BIRT iHub to which the user is logged in.

If GetSystemPrinters cannot retrieve the printer information, for example if the printer is temporarily unavailable, and you specified the PrinterName, the response contains empty or incomplete attributes. If GetSystemPrinters cannot retrieve the printer information and you did not specify PrinterName, a SOAP fault is generated.

|                         |                                                                                                                                                                                                                                                |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>PrinterName</b><br><b>String</b> . The name of the printer for which to retrieve information. If not specified, information is retrieved for all printers configured for the volume.                                                        |
|                         | <b>GetAllPaperSizes</b><br><b>Boolean</b> . Indicates whether to retrieve all available paper sizes for the printer. If True, all paper sizes are retrieved. If False, only a subset of paper sizes are retrieved. The default value is False. |
| <b>Response element</b> | <b>Printers</b><br><b>ArrayOfPrinter</b> . The printer information.                                                                                                                                                                            |

---

## GetSystemServerList

Retrieves the list of BIRT iHubs and their states.

GetSystemServerList can retrieve information about cluster servers and online stand-alone servers. GetSystemServerList cannot retrieve information about stand-alone servers that are offline.

GetSystemServerList is available only to a BIRT iHub System administrator. To log in as a BIRT iHub System administrator, use SystemLogin.

|                         |                                                                                                                                                                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>ServerList</b><br><b>ArrayOfServerInformation</b> . Contains the following information about the server: <ul style="list-style-type: none"> <li>■ Server name</li> <li>■ Server state</li> <li>■ Error code of a failed server</li> <li>■ Error description of a failed server</li> </ul> |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

## GetSystemVolumeNames

Obsolete since iHub Release 2.

---

## GetTOC

Obsolete since iHub Release 2.

---

## GetUserExtendedProperties

Gets list of user extended properties, such as property name, ID, and product family.

**Request elements**

**PropertyName**

**String**. The extended user property name.

**PropertyNameList**

**ArrayOfString**. The list of extended user property names.

**UserName**

**String**. The user name.

**UserId**

**String**. The user ID.

**ProductFamily**

**String**. The product family.

**Response element**

**Properties**

**ArrayOfNameValuePair**. The list of name-value pairs specifying the user extended properties.

---

## GetUserGroupExtendedProperties

Gets list of user group extended properties, such as property name, user group name, ID, and product family.

**Request elements**

**PropertyName**

**String**. The user group extended property name.

**PropertyNameList**

**ArrayOfString**. The list of user group extended property names.

**UserGroupName**

**String**. The user group name.

**UserGroupId**

**String**. The user group ID.

|                         |                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------|
| <b>Response element</b> | <b>Properties</b><br><a href="#">ArrayOfNameValuePair</a> . The user group extended properties list. |
|-------------------------|------------------------------------------------------------------------------------------------------|

## GetUserGroupProductAccess

Gets list of product family access by user group name or ID.

|                         |                                                                                                    |
|-------------------------|----------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>UserGroupName</b><br><a href="#">String</a> . The user group name.                              |
|                         | <b>UserGroupId</b><br><a href="#">String</a> . The user group ID.                                  |
| <b>Response element</b> | <b>ProductFamilies</b><br><a href="#">ArrayOfString</a> . The list of accessible product families. |

## GetUserLicenseOptions

Retrieves the specified user license options.

|                         |                                                                                                                                                |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>UserId</b><br><a href="#">String</a> . The ID of the user for whom to retrieve the printer settings. Specify either UserId or UserName.     |
|                         | <b>UserName</b><br><a href="#">String</a> . The name of the user for whom to retrieve the printer settings. Specify either UserName or UserId. |
| <b>Response element</b> | <b>LicenseOptions</b><br><a href="#">ArrayOfString</a> . The license options list.                                                             |

## GetUserPreference

Gets list of user preferences, including preference name and product family.

|                         |                                                                        |
|-------------------------|------------------------------------------------------------------------|
| <b>Request elements</b> | <b>UserId</b><br><a href="#">String</a> . The user ID.                 |
|                         | <b>UserName</b><br><a href="#">String</a> . The user name.             |
|                         | <b>PreferenceName</b><br><a href="#">String</a> . The preference name. |

## GetUserPrinterOptions

### PreferenceNameList

[ArrayOfString](#). The preference name list.

### ProductFamily

[String](#). The product family.

### Response element

#### Preference

[ArrayOfNameValuePair](#). The user preferences option list.

---

## GetUserPrinterOptions

Retrieves the specified user printer settings.

If GetUserPrinterOptions cannot retrieve the printer information, for example if the printer is temporarily unavailable, and you specified the PrinterName, the response contains empty or incomplete attributes. If GetUserPrinterOptions cannot retrieve the printer information and you did not specify PrinterName, a SOAP fault is generated.

### Request elements

#### UserId

[String](#). The ID of the user for whom to retrieve the printer settings. Specify either UserId or UserName.

#### UserName

[String](#). The name of the user for whom to retrieve the printer settings. Specify either UserName or UserId.

#### PrinterName

[String](#). The name of the printer for which to retrieve the settings.

### Response element

#### PrinterOptions

[ArrayOfPrinterOptions](#). The printer settings.

---

## GetVolumeProperties

Retrieves properties of a specific volume.

### Request element

#### ResultDef

[ArrayOfString](#). The properties to retrieve. By default, VolumeProperties are always returned. In addition, you can specify the following properties:

- ArchiveLibrary
- AutoArchiveSchedule
- ExternalUserPropertyNames
- OnlineBackupSchedule

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
|                          | <ul style="list-style-type: none"> <li>■ TranslatedRoleNames</li> <li>■ PrinterOptions</li> <li>■ VolumeProperties</li> </ul> |
| <b>Response elements</b> | <b>VolumeProperties</b><br><b>Volume</b> . The volume properties.                                                             |
|                          | <b>TranslatedRoleNames</b><br><b>ExternalTranslatedRoleNames</b> . The translated role names.                                 |
|                          | <b>ExternalUserPropertyNames</b><br><b>ArrayOfString</b> . The external user property names list.                             |
|                          | <b>PrinterOptions</b><br><b>ArrayOfPrinterOptions</b> . The printer options.                                                  |
|                          | <b>AutoArchiveSchedule</b><br><b>JobSchedule</b> . The autoarchive schedule.                                                  |
|                          | <b>ArchiveLibrary</b><br><b>String</b> . The name of the archive application for the volume.                                  |
|                          | <b>ArchiveServiceCmd</b><br><b>String</b> . The archive service command for the volume.                                       |
|                          | <b>LicenseOptions</b><br><b>ArrayOfLicenseOption</b> . The license options for the volume.                                    |

## GrantUserGroupCapabilities

|                         |                                                                                           |
|-------------------------|-------------------------------------------------------------------------------------------|
|                         | Grants listed user group capabilities by name or ID.                                      |
| <b>Request elements</b> | <b>UserGroupName</b><br><b>String</b> . The user group name.                              |
|                         | <b>UserGroupId</b><br><b>String</b> . The user group ID.                                  |
|                         | <b>Capabilities</b><br><b>ArrayOfCapabilities</b> . The array of user group capabilities. |

|                         |                                           |
|-------------------------|-------------------------------------------|
| <b>Response element</b> | <b>GrantUserGroupCapabilitiesResponse</b> |
|                         | The response is empty.                    |

## GrantUserGroupProductAccess

Grants listed user group product family access by user group name or ID.

## InstallApp

|                         |                                                                                           |
|-------------------------|-------------------------------------------------------------------------------------------|
| <b>Request elements</b> | <b>UserGroupName</b><br><i>String</i> . The user group name.                              |
|                         | <b>UserGroupId</b><br><i>String</i> . The user group ID.                                  |
|                         | <b>ProductFamilies</b><br><i>ArrayOfString</i> . The list of accessible product families. |
| <b>Response element</b> | <b>GrantUserGroupProductAccessResponse</b><br>The response is empty.                      |

---

## InstallApp

Installs application using name, ZIP file path, and description.

|                         |                                                                    |
|-------------------------|--------------------------------------------------------------------|
| <b>Request elements</b> | <b>Name</b><br><i>String</i> . The application name.               |
|                         | <b>ZipFilePath</b><br><i>String</i> . The ZIP file path.           |
|                         | <b>Description</b><br><i>String</i> . The application description. |
| <b>Response element</b> | <b>FolderId</b><br><i>String</i> . The folder ID.                  |

---

## IsSSOEnabled

Determines whether Single-sign-on (SSO) is enabled.

|                          |                                                                      |
|--------------------------|----------------------------------------------------------------------|
| <b>Request elements</b>  | <b>RelayState</b><br><i>String</i> . The relay state.                |
|                          | <b>EntityID</b><br><i>String</i> . The entity ID.                    |
| <b>Response elements</b> | <b>Enabled</b><br><i>Boolean</i> . Indicates whether SSO is enabled. |
|                          | <b>DestinationURL</b><br><i>String</i> . The destination URL.        |
|                          | <b>SAMLRequest</b><br><i>String</i> . The SAML request.              |
|                          | <b>RelayState</b><br><i>String</i> . The relay state.                |

---

## Login

Authenticates a user to the iHub System.

**Request elements**

**User**

**String**. The name of the user to log in.

**Password**

**String**. The password of the user to log in. You must specify either Password or Credentials.

**EncryptedPwd**

**String**. The password in encrypted.

**Credentials**

**Base64Binary**. Extended credentials data. Used for Report Server Security Extension (RSSE) integration. You must specify Either Credentials or Password.

**Domain**

**String**. The volume to which to log in. In Release 11 and iHub 2, the Login message must specify the volume name using TargetVolume in the SOAP header and Domain in the SOAP message body. In iHub 3, specifying the volume name using TargetVolume and Domain in the SOAP message body are optional.

**UserSetting**

**Boolean**. Specifies whether the response includes detailed user information. If True, the response includes all user attributes. If False, the response does not include detailed user information. The default value is False.

**ValidateUserGroups**

**ArrayOfString**. Checks whether the user is in the specified user groups.

**RunAsUser**

**String**. Specifies the user name in the run-time environment.

**SAMLToken**

**String**. The SAML token.

**GetVolumeList**

**Boolean**. Indicates whether to get the volume list.

**CapabilityCategories**

**ArrayOfString**. The capability categories list.

**ClientVersion**

**String**. Specify the enumeration value ihub3 or actuate11.

**Response elements**

**AuthId**

**String**. The system-generated, encrypted, and authenticated token that the application uses in all subsequent requests.

## OpenInfoObject

### **AdminRights**

**String.** Returned if the user has either Administrator or Operator rights.

### **User**

**User.** All user attributes except the user password.

### **LoginVolume**

**String.** The login volume.

### **FeatureOptions**

**ArrayOfString.** The features available to the user:

- ReportGeneration
- PageSecureViewing  
BIRT Page Level Security option

### **ValidUserGroups**

**ArrayOfString.** The user roles from the ValidateRoles list. Does not return the user roles that ValidateRoles does not specify. For example, if ValidateRoles specifies Sales, Marketing, and Engineering and the user has Sales and Accounting roles, the response contains only Sales.

### **Volumes**

**ArrayOfString.** The specified volumes.

### **Capabilities**

The specified capabilities.

### **SAMLSessionIndex**

**String.** The specified SAML session index.

---

## OpenInfoObject

Obsolete since iHub Release 2.

---

## Ping

Tests whether a specific component of BIRT iHub is operational and retrieves other diagnostic information about the component.

|                         |                    |
|-------------------------|--------------------|
| <b>Request elements</b> | <b>Destination</b> |
|-------------------------|--------------------|

**String.** The component to test. Valid values are:

- MDS  
A Message Distribution service

- EE  
An Encyclopedia engine
- FS  
A Factory service
- VS  
A View service
- OSD  
An open server driver
- AIS  
An Actuate Integration service
- ACS  
Obsolete since iHub Release 3

**Action**

**String.** The optional action to take. Valid values are:

- Echo  
Echoes the data specified in Payload.
- ReadFile  
Opens the specified volume file, reads the file's contents, then closes the file. Applies only if the value of Destination is EE, FS, or VS. Ping returns the timing of the read operation. Specify the file name in FileName.
- WriteFile  
Creates a temporary file in a storage location, writes a specified number of bytes to the file, closes the file, then deletes the file. Applies only if the value of Destination is EE or FS. Ping returns the timing information for each step. Specify the storage location in PartitionName. Specify the number of bytes to read in NumBytes.
- Connect  
Connects to a data source. Specify the connection parameters in ConnectionProperties.

**Mode**

**String.** The level of detail to return. Valid values are:

- Normal  
Returns the names of components in the test path and the timestamps of the request entering and leaving each component. This is the default mode.
- Trace

Returns the timestamp of the request entering and leaving major subcomponents of the component being tested.

- Concise

Returns the elapsed time between a component's receipt of the request and the time the component sends a reply.

**Server**

**String**. Specifies which instance of a Factory service or View service to test. Applies only if the value of Destination is FS or VS. Use Server in conjunction with the ProcessID element. To test all available instances of the Factory or View service, specify an asterisk (\*). If not specified, the BIRT iHub load-balancing mechanism allocates an available instance of the requested service to respond to the request.

**ProcessID**

**String**. Specifies the process ID of the Factory or View service to test. Use in conjunction with the Server element. Applies only if the value of Destination is FS or VS.

**FileName**

**String**. If the value of Action is ReadFile, indicates the volume file to read. If the value of Destination is OSD, specifies the executable file to prepare for execution.

**PartitionName**

**String**. Specifies the name of the storage location on which to create the temporary file. Applies only if the value of Action is WriteFile.

**NumBytes**

**Long**. Specifies the number of bytes to read or write. Applies only if the value of Action is ReadFile or WriteFile. If NumBytes is not specified or 0, the default value of 10 KB is used.

**ConnectionProperties**

**ArrayOfParameterValue**. An array of property name and value pairs that specify the parameter values for establishing a data source connection. Applies only if the value of Action is Connect. To establish a connection, you must specify a property with a name DBType and a value that specifies the type of database. You must

also specify any other properties that the specific database interface requires. Table 13-3 lists the valid property names.

**Table 13-3** Valid connection properties

| Property name       | Applicable database interface                                        | Description                                                                                                                                                                   |
|---------------------|----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DBType              | All                                                                  | The type of database. Valid values are:<br><ul style="list-style-type: none"> <li>■ DB2, Informix, MSSQL</li> <li>■ ODBC, Oracle, Sybase, Progress, Progress SQL92</li> </ul> |
| DllPath             | DB2, Informix, MSSQL, ODBC, Oracle, Progress, Progress SQL92, Sybase | The name of the DLL providing the client database.                                                                                                                            |
| UserName            | DB2, Informix, MSSQL, ODBC, Oracle, Progress, Progress SQL92, Sybase | The database user name.                                                                                                                                                       |
| Password            | DB2, Informix, MSSQL, ODBC, Oracle, Progress, Progress SQL92, Sybase | The database password.                                                                                                                                                        |
| DataSource          | DB2, ODBC                                                            | The name of the data source.                                                                                                                                                  |
| ConnectionString    | ODBC                                                                 | Any additional text that ODBC needs to establish the connection.                                                                                                              |
| HostString          | Oracle                                                               | The Oracle server name for the connection.                                                                                                                                    |
| DatabaseEnvironment | Informix                                                             | The name of the database server, the database, or both database server and database to which to connect.                                                                      |
| DatabaseList        | Progress                                                             | The name of the database.                                                                                                                                                     |
| StartUpParameters   | Progress                                                             | The Progress Open Interface Broker parameters.                                                                                                                                |
| Database            | Progress SQL92                                                       | The name of the database.                                                                                                                                                     |
| Host                | Progress SQL92                                                       | The host computer name for a remote database. Not used for a local database. Required if you are connecting to a database running on a database server.                       |
| ServiceOrPort       | Progress SQL92                                                       | The database service name or port number on the database server. Not used for a local database. The port number is an unsigned 16-bit integer in the range 1–65535.           |

## PrintReport

### **Payload**

**String.** Specifies the payload data. Applies only if the value of Action is Echo. Payload is binary data attached to the request.

### **Response elements**

#### **Reply**

**String.** The Ping reply, in plain text format. The information depends on the value of Mode.

### **Payload**

**String.** If a value is specified for Payload in the request, the payload data as a string.

---

## PrintReport

Prints a document. PrintReport requests are always executed in asynchronous mode.

The document prints to the specified printer. If a printer is not specified, the document prints to the user default printer.

Both PrintReport and SubmitJob support printing. Use PrintReport to print an existing document. Use SubmitJob to schedule execution and printing from a design executable.

### **Request elements**

#### **JobName**

**String.** The job name.

#### **Priority**

**Int.** The job priority. Limited by the user Max job priority setting.

#### **InputFileName**

**String.** The name of the file to print. Specify either InputFileName or InputFileId.

#### **InputFileId**

**String.** The ID of the file to print. Specify either InputFileId or InputFileName.

#### **Schedules**

**JobSchedule.** The schedule for the print job. If not specified, the print request is sent immediately.

#### **PrinterOptions**

**JobPrinterOptions.** The job printer settings. The printer settings have the following precedence:

- Job printer settings
- User printer settings
- System printer settings

**NotifyUsersByName**

[ArrayOfString](#). The names of users to receive job completion notice. Specify either NotifyUsersByName or NotifyUsersById.

**NotifyGroupsByName**

Obsolete since iHub Release 3.

**NotifyChannelsByName**

Obsolete since iHub Release 3.

**NotifyUsersById**

[ArrayOfString](#). The IDs of users to receive the job completion notice. Specify either NotifyUsersById or NotifyUsersByName.

**NotifyGroupsById**

Obsolete since iHub Release 3.

**NotifyChannelsById**

Obsolete since iHub Release 3.

**SendSuccessNotice**

[Boolean](#). Specifies whether notices are sent if report printing succeeds.

**SendFailureNotice**

[Boolean](#). Specifies whether notices are sent if report printing fails.

**SendEmailForSuccess**

[Boolean](#). Specifies whether an e-mail is sent when report printing succeeds.

**SendEmailForFailure**

[Boolean](#). Specifies whether an e-mail is sent when report printing fails.

**OverrideRecipientPref**

[Boolean](#). Specifies whether recipient preferences are overridden.

**RecordSuccessStatus**

[Boolean](#). Specifies whether the job status is kept if report printing succeeds.

**RecordFailureStatus**

[Boolean](#). Specifies whether the job status is kept if report printing fails.

**RetryOptions**

[RetryOptions](#). Specifies how to retry printing if the previous attempt failed. Used only if Retryable is specified.

**Response element****JobId**

[String](#). The ID of the print job. Returned after the job is created.

## RevokeUserGroupCapabilities

Revokes list of user group capabilities using group name or ID.

**Request elements**

**UserGroupId**

[String](#). The user group ID.

**UserGroupName**

[String](#). The user group name.

**Capabilities**

[ArrayOfCapabilities](#). The array of user capabilities.

**Response element**

**RevokeUserGroupCapabilitiesResponse**

The revoke user group capabilities response. Data type undefined.

---

## RevokeUserGroupProductAccess

Revokes user group product access using group name, ID, and list of product families.

**Request elements**

**UserGroupId**

[String](#). The user group ID.

**UserGroupName**

[String](#). The user group name.

**ProductFamilies**

[ArrayOfString](#). The product families list.

**Response element**

**RevokeUserGroupProductAccessResponse**

The revoke user group product access response. Data type undefined.

---

## SaveSearch

Obsolete since iHub Release 2.

---

## SaveTransientReport

Saves a report to a specified file.

**Request elements**

**Object**

[ObjectIdentifier](#). The object identifier of the report.

**NewFile**

[NewFile](#). The file to which the report is saved.

**Response element**

**FileId**

[String](#). File ID of created report file.

---

## SearchReport

Obsolete since iHub Release 2.

---

## SelectChannels

Obsolete since iHub Release 2.

---

## SelectFiles

Retrieves information about a specified file.

To retrieve a single file or folder, specify Name or Id. To retrieve a list of files or folders, specify NameList or IdList. To search all file or folders that match specific condition, specify Search.

**Request elements**

**WorkingFolderId**

[String](#). The ID of the working folder in which to search for the file. Specify either WorkingFolderId or WorkingFolderName.

**WorkingFolderName**

[String](#). The absolute path and the name of the working folder in which to search for the file. Specify either WorkingFolderName or WorkingFolderId.

**Recursive**

[Boolean](#). Specifies whether to search subfolders. If True, the search includes subfolders. The default value is False.

**LatestVersionOnly**

[Boolean](#). Specifies whether to search all versions of the file. If True, the search includes only the latest version. The default value is False.

**ResultDef**

[ArrayOfString](#). The properties to retrieve.

**Search**

[FileSearch](#). The search condition. If conditions apply to multiple fields, use ConditionArray.

## SelectFileTypes

### NameList

[ArrayOfString](#). The name of the file or folder list to retrieve. Specify either IdList or NameList.

### IdList

[ArrayOfString](#). The ID of the file or folder list to retrieve. Specify either IdList or NameList.

### Name

[String](#). The name of a single file or folder to retrieve. Specify either Name or Id.

### Id

[String](#). The ID of a file or folder to retrieve. Specify either Id or Name.

### Content

[String](#). Specifies whether the file is embedded in or attached to the response. Valid values are:

- Embed  
The file is embedded.
- Attach  
The file is attached.

### Response elements

#### ItemList

[ArrayOfFile](#). The list of attached items that match the search criteria.

#### ContentItemList

[ArrayOfFileContent](#). The list of embedded items that match the search criteria.

#### FetchHandle

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

#### TotalCount

[Long](#). The number of entries in the search result set.

---

## SelectFileTypes

Searches file types for specified information.

To search a single file type, specify Name. To search a list of file types, specify NameList. To search file types matching the specified conditions, specify Search. File names are case sensitive, and file type extensions are stored in uppercase. Specify uppercase for all file type extensions, for example, use RPTDESIGN instead of rptdesign.

### Request elements

#### ResultDef

[ArrayOfString](#). The properties to retrieve.

**Search**

[ArrayOfString](#). The search conditions. If conditions apply to multiple fields, use ConditionArray.

**NameList**

[ArrayOfString](#). The list of file types to search.

**Name**

[String](#). The name of a single file type to search.

**Response element**

**FileTypes**

[ArrayOfFileType](#). The specified file types.

---

## SelectGroups

Obsolete since iHub Release 2.

---

## SelectJavaReportPage

Returns a report page formatted in the specified display format indicated by the Page or Component element.

**Request elements**

**Object**

[ObjectIdentifier](#). The ID of the object from which to select the report page.

**Page**

[PageIdentifier](#). The identifier of the page to retrieve.

**Component**

[ArrayNameValuePair](#). The name, display name, or ID, and the value of the component from which to retrieve the data.

**ViewParameterValues**

[ArrayOfParameterValue](#). Include view parameters if defined in the document. This feature is not currently supported for a BIRT report.

**OutputFormat**

[String](#). For a BIRT report, the output format can be HTML, rptdocument, or PDF. The default is rptdocument.

**ViewProperties**

[ArrayNameValuePair](#). Specifies the layout and contents of a report such as the name of a bookmark name, table of contents, or an object ID. ViewProperties is available to the BIRT render task as the java.util.Map object in the Engine ApplicationContext under the key ServerViewProperties.

## SelectJobNotices

### DownloadEmbedded

[Boolean](#). Specifies whether to send the attachment embedded in the body of the SOAP message or in a separate MIME boundary. If True, the response contains the attachment as embedded data. If False, the attachment is in a separate MIME boundary. The default value is False.

### Response elements

#### PageRef

[Attachment](#). Contains the following properties of the attachment:

- MimeType
- ContentEncoding
- ContentLength
- Locale

#### ConnectionHandle

[String](#). The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

#### OutputProperties

[ArrayOfNameValuePair](#). Applies only if Format is CSV. The properties to include in the search result are:

- EnableColumnHeaders  
If True, column headers are included in the search result. The default value is True.
- UseQuoteDelimiter  
If True, each data item in the search result is enclosed in double quotes (""). The default value is True.

---

## SelectJobNotices

Retrieves job notices matching the specified criteria.

### Request elements

#### ResultDef

[ArrayOfString](#). The properties to retrieve. You can specify the following properties:

- JobId
- JobName
- ActualHeadline
- CompletionTime
- ActualOutputFileId
- ActualOutputFileName

- VersionName
- OutputFileSize

**Search**

[JobNoticeSearch](#). The search conditions. If conditions apply to multiple fields, use ConditionArray.

**Response elements****JobNotices**

[ArrayOfJobNotice](#). The selected job notices.

**FetchHandle**

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

**TotalCount**

[Long](#). The number of entries in the search result set.

## SelectJobs

Selects all jobs matching the specified conditions. SelectJobs returns states and information for job instances.

In Release 11 or iHub, use SelectJobSchedules to obtain job schedule information. In Release 10 Service Pack 1 and earlier, use SelectJobs with JobCondition, specifying Scheduled, Expired, or Cancelled, to obtain designated job schedule information.

Release 11 Service Pack 4 and later support backward compatibility for all pre-Release 11 applications that send a SelectJobs request. However, all Release 11 and iHub applications must implement the new SelectJobSchedules API to request schedule information.

**Request elements****ResultDef**

[ArrayOfString](#). The properties to retrieve.

**Search**

[JobSearch](#). The search conditions. If conditions apply to multiple fields, use ConditionArray.

**IdList**

[ArrayOfString](#). The list of job IDs to search.

**Id**

[String](#). The ID of a single job to search.

**Response elements****Jobs**

[ArrayOfJobProperties](#). The selected jobs.

## SelectJobSchedules

### FetchHandle

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

### TotalCount

[Long](#). The number of entries in the search result set.

---

## SelectJobSchedules

Selects all scheduled jobs matching the specified criteria. SelectJobSchedules also retrieves information for expired and cancelled jobs.

In Release 11 or iHub, use SelectJobSchedules to obtain job schedule information. In Release 10 Service Pack 1 and earlier, use SelectJobs with JobCondition, specifying Scheduled, Expired, or Cancelled, to obtain designated job schedule information.

Release 11 Service Pack 4 and iHub support backward compatibility for all pre-Release 11 applications that send a SelectJobs request. However, all Release 11 and iHub applications must implement the new SelectJobSchedules API to request schedule information.

### Request elements

#### ResultDef

[ArrayOfString](#). The properties to retrieve.

#### Search

[JobScheduleSearch](#). The search conditions. If conditions apply to multiple fields, use ConditionArray.

#### IdList

[ArrayOfString](#). The list of job IDs to search.

#### Id

[String](#). The ID of a single job to search.

### Response elements

#### Jobs

[ArrayOfJobProperties](#). The selected jobs.

### FetchHandle

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

### TotalCount

[Long](#). The number of entries in the search result set.

---

## SelectPage

Obsolete since iHub Release 2.

---

## SelectRoles

Deprecated since iHub Release 3. Replaced by [SelectUserGroups](#).

---

## SelectUserGroups

Searches user groups for specified information.

To search a single user group, specify Name or Id. To search a list of user groups, specify NameList or IdList. To search user groups matching the specified conditions, specify Search.

**ResultDef**

[ArrayOfString](#). The properties to retrieve.

**Search**

[UserGroupSearch](#). The search conditions. If conditions apply to multiple fields, use ConditionArray. When using RSSE external registration, SelectUserGroups allows only one search condition.

In the following example, <Condition> and <AssignedToUserName> are search conditions:

```
<Search>
 <Condition>
 <Field>Name</Field>
 <Match>roleA</Match>
 </Condition>
 <ConditionArray xsi:nil="true"/>
 <ParentUserName xsi:nil="true"/>
 <ChildUserName xsi:nil="true"/>
 <WithRightsToChannelName xsi:nil="true"/>
 <AssignedToUserName>Administrator</AssignedToUserName>
 <ParentRoleId xsi:nil="true"/>
 <ChildRoleId xsi:nil="true"/>
 <WithRightsToChannelId xsi:nil="true"/>
 <AssignedToUserId xsi:nil="true"/>
</Search>
```

## SelectUsers

When using RSSE external registration, the previous search pattern is invalid, generating the following response:

```
<Description>
 <Message>The search pattern is too long or is incorrect.
 </Message>
 <Parameter1>More than one search condition is invalid under
 External Registration
 </Parameter1>
</Description>
```

### **IdList**

**ArrayOfString**. The list of user IDs to search. Specify either IdList or NameList.

### **NameList**

**ArrayOfString**. The list of user names to search. Specify either NameList or IdList.

### **Id**

**String**. The ID of the single user to search. Specify either Id or Name.

### **Name**

**String**. The name of the single user to search. Specify either Name or Id.

#### **Response elements**

### **Roles**

**ArrayOfRole**. The selected roles.

### **FetchHandle**

**String**. Indicates that the number of items in the result set exceeds the FetchSize limit.

### **TotalCount**

**Long**. The number of entries in the search result set.

---

## SelectUsers

Searches users for specified information.

To search a single user, specify Name or Id. To search a list of users, specify NameList or IdList. To search users matching the specified conditions, specify Search.

#### **Request elements**

### **ResultDef**

**ArrayOfString**. The properties to retrieve.

### **Search**

**UserSearch**. The search conditions. If conditions apply to multiple fields, use ConditionArray. When using RSSE external registration, SelectUsers allows only one search condition.

**IdList**

[ArrayOfString](#). The list of user IDs to search. Specify either IdList or NameList.

**NameList**

[ArrayOfString](#). The list of user names to search. Specify either NameList or IdList.

**Id**

[String](#). The ID of the single user to search. Specify either Id or Name.

**Name**

[String](#). The name of the single user to search. Specify either Name or Id.

**Response elements****Users**

[ArrayOfUser](#). The selected users.

**FetchHandle**

[String](#). Indicates that the number of items in the result set exceeds the FetchSize limit.

**TotalCount**

[Long](#). The number of entries in the search result set. Not used when querying a volume that uses Open Security. In this case, TotalCount returns Null. To retrieve the number of entries from a volume that uses Open Security, use an array length. For example, the following code returns Null:

```
com.actuate.schemas.SelectUsersResponse userSrchResponse =
 proxy.selectUsers(userSel);
com.actuate.schemas.ArrayOfUser userArr =
 userSrchResponse.getUsers();
com.actuate.schemas.User []user = userArr.getUser();
userSrchResponse.getTotalCount();
int noOfUsers1 = userSrchResponse.getTotalCount().intValue();
The following code returns the correct value:
com.actuate.schemas.SelectUsersResponse userSrchResponse =
 proxy.selectUsers(userSel);
com.actuate.schemas.ArrayOfUser userArr =
 userSrchResponse.getUsers();
com.actuate.schemas.User []user = userArr.getUser();
userSrchResponse.getTotalCount();
int noOfUsers2 = user.length;
```

## SetConnectionProperties

Sets the connection properties for a file based on user or role.

**Request elements****FileName**

[String](#). The name of the file.

## SetServerResourceGroupConfiguration

### Field

**String**. The file ID.

### UserName

**String**. User name for which to set property values.

### UserId

**String**. User ID for which to set property values.

### RoleName

**String**. The role name for which to set property values.

### RoleId

**String**. The role ID for which to set property values.

### UserGroupName

**String**. The user group name for which to set property values.

### UserGroupId

**String**. The user group ID for which to set property values.

### ConnectionProperties

**ArrayOfPropertyValue**. An array of name-value pairs containing the connection properties being set.

---

## SetServerResourceGroupConfiguration

Sets or updates properties of all resource groups on a BIRT iHub.

SetServerResourceGroupConfiguration is available only to a volume administrator or a user with an Administrator role.

### Request elements

#### ServerName

**String**. The name of the BIRT iHub.

#### ServerResourceGroupSettingList

**ArrayOfServerResourceGroupSetting**. Contains one or more of the following properties to set or update:

- Activate
- MaxFactory
- FileTypes

Cannot be set or updated for a default resource group

### Response element

#### SetServerResourceGroupConfigurationResponse

The SetServerResourceGroupConfigurationResponse. Data type undefined.

---

## SetUserExtendedProperties

Sets list of user extended properties using user name or ID and product family.

**Request elements** **UserName**  
[String](#). The user name.

**UserId**  
[String](#). The user ID.

**Properties**  
[ArrayOfNameValuePair](#). The user extended properties list.

**ProductFamily**  
[String](#). The product family.

**Response element** **SetUserExtendedPropertiesResponse**  
The SetUserExtendedPropertiesResponse. Data type undefined.

---

## SetUserGroupExtendedProperties

Sets list of user group extended properties using user group name or ID.

**Request elements** **UserGroupName**  
[String](#). The user group name.

**UserGroupId**  
[String](#). The user group ID.

**Properties**  
[ArrayOfNameValuePair](#). The user group extended property list.

**ProductFamily**  
[String](#). The product family.

**Response element** **SetUserGroupExtendedPropertiesResponse**  
The SetUserGroupExtendedPropertiesResponse. Data type undefined.

---

## SetUserPreference

Sets user preference using user name, ID, and product family.

**Request elements** **UserId**  
[String](#). The user ID.

**UserName**  
[String](#). The user name.

## SubmitJob

### Preference

[ArrayOfNameValuePair](#). The user preference list.

### ProductFamily

[String](#). The product family.

### Response element

#### SetUserPreferenceResponse

The SetUserPreferenceResponse. Data type undefined.

---

## SubmitJob

Generates and prints a report or information object in asynchronous mode. BIRT iHub sends a response after completing the request.

After generating a document in asynchronous mode, you can convert the document to one of the following formats:

- Advanced function printing (AFP)
- Comma-separated values (CSV)
- Excel XLS
- Excel XLSX
- PDF
- PostScript
- PowerPoint (PPT)
- PowerPoint (PPTX)
- PSV
- Tab-separated values (TSV)
- Word (DOC)
- Word (DOCX)

When converting a document, the properties of the converted file are the same as the original document properties.

BIRT iHub only supports converting document output for asynchronous generation. Conversion is not supported for the following types of document output:

- Synchronous generation
- Report bursting
- Page-level security

Both SubmitJob and PrintReport support printing. Use SubmitJob to schedule execution and printing from design executables. Use PrintReport to print an existing document.

**Request elements****JobName**

**String**. The name of the job.

**Headline**

**String**. The job headline.

**Priority**

**Int**. The job priority. Job priority is limited by the user Max job priority setting. Valid values are 0–1,000, where 1,000 is the highest priority.

**ResourceGroup**

**String**. The resource group to which to assign the job. Available only to a volume administrator or a user with the Administrator role.

**InputFileName**

**String**. The full path, name, and version number of the file to use as input. If RunLatestVersion is specified, the version number is ignored.

**InputFileId**

**String**. The ID of the file to use as input.

**RunLatestVersion**

**Boolean**. Specifies whether to run or print the most recent version of the executable file. If True, the latest version of the file is used. If True, the version number in InputFileName is ignored. The default value is True.

**RequestedOutputFile**

**NewFile**. The file name and extension for the output.

**Operation**

**String**. Specifies the type of task to perform, RunReport, RunAndPrintReport, ConvertReport, or PrintReport.

**ParameterValues**

**ArrayOfParameterValue**. A list of parameter values to use. Specify either ParameterValues, ParameterValueFileDialog, ParameterValueFileName.

**ParameterValueFileName**

**String**. The name of the parameter value file to use. Specify either ParameterValues, ParameterValueFileDialog, or ParameterValueFileName.

**ParameterValueFileDialog**

**String**. The ID of the report object values file to use. Specify either ParameterValues, ParameterValueFileDialog, or ParameterValueFileName.

### Schedules

**JobSchedule**. Specifies the schedule on which to run the report. If not specified, the job runs immediately.

### PrinterOptions

**ArrayOfPrinterOptions**. If the job is to be printed, specifies the job printer settings. If the job is not to be printed, PrinterOptions is ignored. The printer settings have the following precedence:

- Job printer settings
- User printer settings
- System printer settings

### NotifyUsersByName

**ArrayOfString**. The names of users to receive the job completion notice. Specify either NotifyUsersByName or NotifyUsersById.

### NotifyGroupsByName

Obsolete since iHub Release 3.

### NotifyChannelsByName

Obsolete since iHub Release 3.

### NotifyUsersById

**ArrayOfString**. The IDs of users to receive the job completion notice. Specify either NotifyUsersById or NotifyUsersByName.

### NotifyGroupsById

Obsolete since iHub Release 3.

### NotifyChannelsById

Obsolete since iHub Release 3.

### SendSuccessNotice

**Boolean**. Specifies whether success notices are sent if the job succeeds. Used only if OverrideRecipientPref is True. If SendSuccessNotice is True, notices are sent.

### SendFailureNotice

**Boolean**. Specifies whether failure notices are sent if the job fails. Used only if OverrideRecipientPref is True. If SendFailureNotice is True, failure notices are sent to specified users and groups if the job fails.

### SendEmailForSuccess

**Boolean**. Specifies whether e-mail notifications are sent if the job succeeds. Used only if OverrideRecipientPref is True. If True, e-mail notifications are sent to specified users and groups if the job succeeds. The default value is False.

### SendEmailForFailure

**Boolean**. Specifies whether e-mail notifications are sent to specified users and groups if the job fails. Used only if OverrideRecipientPref is True. If

SendEmailForFailure is True, e-mail notifications are sent. The default value is False.

**AttachReportInEmail**

**Boolean**. Specifies whether to attach a report to an e-mail completion notice. Used only if OverrideRecipientPref is True. If AttachReportInEmail is True, the output file is attached to the e-mail notification if the job succeeds. If False, only a link to the output file is sent. Specify the format for the attachment in the EmailFormat parameter. The default value is False.

**OverrideRecipientPref**

**Boolean**. Specifies whether e-mail notifications and output attachments are sent according to job settings or user settings. If True, e-mail notifications and output attachments are sent according to job settings. If False, e-mail notifications and output attachments are sent according to user settings. The default value is False. If False, the following elements are ignored:

- AttachReportInEmail
- SendEmailForSuccess
- SendEmailForFailure
- SendSuccessNotice
- SendFailureNotice

**EmailFormat**

**String**. Specifies the output format of the report attached to the e-mail notification. The following formats are supported:

- ExcelDisplay
- PDF
- rptdesign
- rptdocument

**RecordSuccessStatus**

**Boolean**. Specifies whether to keep the job status for successful jobs. If True, the job status is kept if job execution succeeds.

**RecordFailureStatus**

**Boolean**. Specifies whether to keep the job status for failed jobs. If True, the job status is kept if job execution fails.

**IsBundled**

**Boolean**. Specifies whether a report is bundled. If True, the report is bundled. If False, the report is not bundled. The default value is False.

## SystemLogin

### RetryOptions

**RetryOptions**. Specifies how to retry the job if the previous attempt failed. Used only if Retryable is specified.

### OpenServerOptions

**OpenServerOptions**. Contains the following open server options:

- **KeepWorkingSpace**

Specifies whether the workspace directory is removed after the job completes.

- **DriverTimeout**

The time for the driver to return from executing a job.

- **PollingInterval**

The time interval for the open server to get status messages. The minimum value is 10 seconds.

### KeepOutputFile

**Boolean**. Specifies whether the generated output file remains in the volume if the generation request succeeds but the printing request fails. Used if Operation is RunAndPrintReport. If True, the output file remains in the volume if the printing request fails. If False, the output file is deleted if the printing request fails. The default value is False.

### ConversionOptions

Obsolete since iHub Release 2.

### WaitForEvent

**Event**. An event that must be completed before the response is processed.

### DataACL

**ArrayOfString**. Specifies the access control list (ACL) restricting data privileges.

#### Response element

##### JobId

**String**. The job ID. Use the job ID to refer to the job in subsequent requests during the current session.

---

## SystemLogin

Logs the user in as the BIRT iHub administrator.

#### Request elements

##### SystemPassword

**String**. The password.

##### SystemPasswordEncryptLevel

**Long**. The encryption level of the SystemPassword. Valid values are:

- 0 - No encryption

- 1 - Two way encryption

- 2 - Hash encryption

The default is hash encryption.

**Response element****AuthId**

[String](#). The system-generated, encrypted authenticated string all subsequent requests use.

---

## UpdateDatabaseConnection

Obsolete since iHub Release 2.

---

## UpdateIOCache

Obsolete since iHub Release 2.

---

## UpdateResourceGroup

Updates resource group properties. You cannot update the resource group name, type, or the name of the BIRT iHub on which the resource group runs.

**Request elements****ResourceGroup**

[ResourceGroup](#). Contains one or more of the following properties to update:

- Disabled
- Description
- MinPriority
- MaxPriority
- Reserved

**ResourceGroupSettingsList**

[ArrayOfResourceGroupSettings](#). Contains one or more of the following properties to update:

- Activate
- MaxFactory
- FileTypes

---

## UploadFile

Uploads a file to a volume. You can upload the file as a MIME attachment or embed it in the request. To embed the file in the request, specify the ContentData element of the attachment.

<b>Request elements</b>	<b>NewFile</b> <a href="#">NewFile</a> . The file to upload.
	<b>CopyFromLatestVersion</b> <a href="#">ArrayOfString</a> . Copies one or more of the following properties from the latest version of the file, if one exists, to the version of the file that you upload:
	<ul style="list-style-type: none"><li>■ Description The description of the file</li><li>■ Permissions Access control list (ACL) specifying the users and roles that can access the file</li><li>■ ArchiveRule The autoarchive rules for the file</li></ul>
	<b>Content</b> <a href="#">Attachment</a> . The information about the file, such as the encoding the file uses and the data to upload.
<b>Response element</b>	<b>FileId</b> <a href="#">String</a> . The ID of the uploaded file.

---

## VerifyAuthId

Checks the validity of an AuthID string with a target volume.

<b>Request elements</b>	<b>AuthID</b> <a href="#">String</a> . An authentication ID.
	<b>Volume</b> <a href="#">String</a> . The volume with which to verify the AuthID.
	<b>UserName</b> <a href="#">String</a> . The user name.
<b>Response element</b>	<b>isValid</b> A boolean value for the validity of the AuthID with the target volume and user name.

---

## WaitForExecuteReport

Retrieves the status of the report generation request after receiving the Pending status. Send WaitForExecuteReport after sending CancelReport.

If progressive viewing is enabled, WaitForExecuteReport retrieves the status after the first page generates. Otherwise, WaitForExecuteReport waits until the report is complete.

If the current job status is Pending, WaitForExecuteReport waits for the report to generate.

**Request element**

**ObjectId**  
**String**. The ID of the job.

**Response elements**

**Status**  
**String**. The status of the request. One of the following values:

- Done
- Failed
- FirstPage

**ErrorDescription**

**String**. The description of the error. Returned if Status is Failed.

**OutputFileType**

**String**. The type of the output file.

**ObjectId**

**String**. The object ID of the report.

**ConnectionHandle**

**String**. The ID of the report. Supports viewing a report when the report is already in iHub System. Specified in the SOAP header.

## WaitForExecuteReport

# 14

## Actuate Information Delivery API data types

This chapter provides reference documentation for Actuate Information Delivery API data type classes the uses listed in alphabetical order. Each entry includes a general description of the data type and elements.

### IDAPI data types quick reference

Table 14-1 lists the Actuate IDAPI data types. For more information about IDAPI data type WSDL definitions, see Chapter 9, “Understanding the Information Delivery API and schema.”

**Table 14-1** IDAPI data type summary

Data Type	Description
AbsoluteDate	Date and run options for a job
acDouble	Hexadecimal double
acNull	Null value
AdminOperation	Controls the ability to create, delete, update, copy, and move items within a volume.
Arrays of data types	Arrays of singular data types
Attachment	An attachment as binary data

(continues)

**Table 14-1** IDAPI data type summary (continued)

Data Type	Description
Base64Binary	Standard XML base64Binary data type
BookMark	Bookmark metadata from a report document
Boolean	Primitive XML boolean data type
CancelJobStatus	Status of a cancel job request
Capabilities	User capabilities for specific tools
ColumnDetail	Type of data within a column
ColumnSchema	The schema of a column
CopyFile	A file or list of files to copy
CounterInfo	A counter object
CreateFileType	A file type to create
CreateFolder	A folder to create
CreateUser	A user to create
CreateUserGroup	A user group to create
CustomEvent	Details for a custom event
Daily	Enumerated value for daily setting
DatabaseConnectionDefinition	Details of a database connection
DataCell	Type of data within a data cell
DataExtractionFormat	Format of a file and its mime type
DataFilterCondition	Details of a filter condition
DataRow	Information from a data row
DataSchema	Details of a data schema by column
DataSortColumn	Details of a sorted data column
DataSourceType	The type of file in which a parameter exists
DataType	Details of a data type
Date	A date
DateTime	A date and time
Decimal	Primitive XML decimal data type
DeleteFile	Details of a file or folder to delete
DeleteFileType	Details of a file type to delete
DeleteJob	Details of a job to delete
DeleteJobNotices	Details of job notices to delete

**Table 14-1** IDAPI data type summary (continued)

Data Type	Description
DeleteJobSchedule	Details of a schedule to delete
DeleteUser	Details of a user to delete
DeleteUserGroup	Details of a user group to delete
DocumentConversionOptions	Conversion options of a file
Double	Primitive XML double data type
Event	Describes an event and its status
EventOptions	Describes polling and other options for an event
EventType	A type of event
ExecuteReportStatus	The status of a temporary report
ExternalTranslatedUserNames	The external name of an administrator user
ExternalTranslatedUserGroupNames	The external name of the Administrators user group
FieldDefinition	A scalar parameter
File	Describes a file
FileAccess	A file's access type
FileCondition	The field and condition for a file search
FileContent	A list of attached files or the content of embedded files
FileEvent	Information about file type events
FileField	Lists file fields upon which a search can be performed
FileSearch	A file search object
FileType	A file type
Header	The contents of a SOAP header
InfoObjectData	Data from a BIRT information object
InfoObjectDataFormat	An information object's data format
Int	Primitive XML integer data type
JobCondition	The field and condition to match for a job search
JobEvent	Information about job type events
<i>(continues)</i>	
JobField	Describes job fields
JobInputDetail	Job input and output files

**Table 14-1** IDAPI data type summary (continued)

Data Type	Description
JobNotice	A job notice
JobNoticeCondition	The field and condition for job notice field search
JobNoticeField	Describes notice fields
JobNoticeSearch	A notice search object
JobPrinterOptions	Describes job printer options
JobProperties	General job attributes
JobSchedule	Details about a job schedule
JobScheduleCondition	The field and condition for schedule field search
JobScheduleDetail	A schedule for running a job
JobScheduleField	Describes schedule fields
JobScheduleSearch	A schedule search object
JobSearch	A file search object
LicenseOption	A license option
Long	Primitive XML long data type
MDSInfo	Describes a Message Distribution service (MDS)
Monthly	Enumerated value for monthly setting
MoveFile	Moves files or folders to a new location
NameValuePair	A named piece of data and its value
NewFile	Describes a file
ObjectIdentifier	Describes object identifiers
PageIdentifier	Describes page numbers
ParameterDefinition	Defines a report parameter
ParameterValue	Value of a report parameter
PendingSyncJob	A job in the factory queue
Permission	A user or user group's privileges
Printer	Describes a printer
PrinterOptions	Describes printer options
PrivilegeFilter	A filter based on user or user group privileges
PropertyValue	A name-value pair
Repeat	A job frequency
ReportParameterType	Describes parameter types

**Table 14-1** IDAPI data type summary (continued)

Data Type	Description
ResourceGroup	Defines a resource group
ResourceGroupSettings	Describes resource group settings
ResultSetSchema	The schema for a result set
RetryOptions	Retry options after failed report generation or printing
RetryOptionType	Describes retry option types
RunningJob	Describes a running job
ScalarDataType	A scalar parameter type
ServerInformation	Describes a BIRT iHub server
ServerResourceGroupSetting	Describes the settings of a resource group
ServerState	Describes the current state of Actuate iHub
ServerStatusInformation	Describes the detailed status of Actuate iHub
ServerVersionInformation	Describes the version of Actuate iHub
Service	Simple XML service data type
Short	Primitive XML short data type
Stream	A streamed image
String	Primitive XML string data type
SystemType	Describes the network configuration of an iHub System
Time	Standard XML Time data type
Transaction	Packaging mechanism for Administrate operations
TypeName	The name of a data type
UndeleteUser	Operation that reverses a DeleteUser operation
UpdateFile	Describes an update of file or folder details
UpdateFileOperation	Operation to update file or folder details
UpdateFileOperationGroup	Sequence of operations to update file or folder details
UpdateFileType	Describes an update of a file type
UpdateFileTypeOperation	Operation to update a file type
UpdateFileTypeOperationGroup	Sequence of operations to update a file type

*(continues)*

**Table 14-1** IDAPI data type summary (continued)

Data Type	Description
UpdateJobSchedule	Describes an update of a job schedule
UpdateJobScheduleOperation	Operation to update a job schedule
UpdateJobScheduleOperationGroup	Sequence of operations to update a job schedule
UpdateOpenSecurityCache	Describes a security cache refresh operation
UpdateUser	Describes an update of a user
UpdateUserGroup	Describes an update of a user group
UpdateUserGroupOperation	Operation to update a user group
UpdateUserGroupOperationGroup	Sequence of operations to update a user group
UpdateUserOperation	Operation to update a user
UpdateUserOperationGroup	Sequence of operations to update a user
UpdateVolumeProperties	Describes an update of volume properties
UpdateVolumePropertiesOperation	Operation to update volume properties
UpdateVolumePropertiesOperationGroup	Sequence of operations to update volume properties
User	Describes a user
UserCondition	The field and condition to match for a user search
UserField	Describes the fields within a user element
UserGroup	Describes a user group
UserGroupCondition	The field and condition to match for a user group search
UserGroupField	Lists user group fields on which a search can be performed
UserGroupSearch	A user group search object
UserSearch	A user search object
VersioningOption	Options for handling the latest existing version
ViewParameter	Describes a viewing parameter
Volume	Describes a volume
Weekly	Enumerated value for weekly setting

---

## AbsoluteDate

A complex data type that includes a date and run options for a job.

**Elements** **RunOn**  
**String**. The date that a job is scheduled to run.

**OnceADay**  
**String**. The days a job is to be run.

**Repeat**  
**Repeat**. Repeats the job run during a set start and stop time.

---

## acDouble

A simple data type that represents a hexadecimal double.

---

## acNull

A simple data type that represents a null value.

---

## AdminOperation

Controls the ability to create, delete, update, copy, and move items within a volume. An AdminOperation request represents a single unit of work within an Administrate operation. Only a volume administrator or a user in the Administrator user group uses these operations.

**Elements** **Transaction**  
A packaging mechanism for Administrate operations. If a failure occurs anywhere in a transaction, all operations in the transaction fail.

**CreateUser**  
Creates a user in the volume.

**DeleteUser**  
Deletes one or more users.

**UndeleteUser**  
Undeletes one or more previously deleted users within the unit of work.

**UpdateUser**  
Updates user properties.

**CreateUserGroup**  
Creates a user group.

**DeleteUserGroup**  
Deletes one or more user groups.

**UpdateUserGroup**

Updates user group properties.

**CreateFileType**

Creates a new file type in BIRT iHub.

**DeleteFileType**

Deletes file types.

**UpdateFileType**

Updates file type properties in the volume.

**CreateFolder**

Creates a folder in a volume.

**DeleteFile**

Deletes files or folders from the volume.

**MoveFile**

Moves a file or folder from the working directory to a specified target directory in the volume.

**CopyFile**

Copies a file or folder in the working directory to a specified target directory.

**UpdateFile**

Updates file or folder properties in the volume.

**DeleteJob**

Deletes one or more jobs.

**DeleteJobNotices**

Deletes one or more job notices.

**DeleteJobSchedule**

Deletes a job schedule.

**UpdateJobSchedule**

Updates a job schedule.

**UpdateVolumeProperties**

Updates the properties of a specific volume.

**UpdateOpenSecurityCache**

Flushes the volume's open security cache and retrieves new data from an external security source.

---

## Aggregation

Obsolete since iHub Release 2.

---

## ArchiveRule

A complex data type that represents an archiving rule.

**Elements****FileType**

**String**. The file type. Cannot exceed 20 characters.

**NeverExpire**

**Boolean**. Optional. Specifies whether the object expires.

**ExpireDependentFiles**

**Boolean**. Optional. Specifies whether the object's dependent files expire when the object is expired.

**ArchiveOnExpiration**

**Boolean**. Optional. Specifies whether the object is archived before it is expired.

**ExpirationAge**

**Long**. Optional. The expiration age for the object. Use either this element or **ExpirationTime**, but not both.

**ExpirationTime**

**DateTime**. Optional. The expiration time for the object. Use either this element or **ExpirationAge**, but not both.

**IsInherited**

**Boolean**. Optional. Specifies whether the rule is inherited.

**InheritedFrom**

**Boolean**. Optional. The object from which the rule is inherited.

---

## Argument

Obsolete since iHub Release 2.

---

## Arrays of data types

Data type definitions can be grouped into arrays. Each array has a specific definition for that data type. IDAPI supports empty arrays having no entries.

## Arrays of data types

The schema for an array of a data type generally follows the following pattern:

```
<xsd:complexType name="ArrayOfX">
 <xsd:sequence>
 <xsd:element name="X" type="typens:X"
 maxOccurs="unbounded" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

In the above listing, X is the data type of object the array contains. For example, the XML for an array of UserGroup objects is:

```
<xsd:complexType name="ArrayOfUserGroup">
 <xsd:sequence>
 <xsd:element name="UserGroup" type="typens:UserGroup"
 maxOccurs="unbounded" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

The following data types have arrays defined in this manner:

- Aggregation
- ArchiveRule
- Argument
- Attachment
- Capabilities
- Channel
- ChannelCondition
- ColumnDefinition
- ColumnSchema
- Component
- ComponentIdentifier
- CounterInfo
- DataExtractionFormat
- DataFilterCondition
- DataRow
- DataSortColumn
- DocumentConversionOptions
- FieldDefinition
- File
- FileCondition
- JobProperties
- JobScheduleCondition
- JobScheduleDetail
- LicenseOption
- MDSInfo
- NameValuePair
- ParameterDefinition
- ParameterValue
- PendingSyncJob
- Permission
- Printer
- PrinterOptions
- PropertyValue
- Record
- ResourceGroup
- ResourceGroupSettings
- ResultSetSchema
- Role
- RoleCondition
- RunningJob

- FileContent
- FileType
- FilterCriteria
- Group
- GroupCondition
- Grouping
- IOCacheDBIndexConstraint
- JobCondition
- JobNotice
- JobNoticeCondition
- ServerInformation
- ServerResourceGroupSetting
- Service
- SortColumn
- String
- Time
- User
- UserCondition
- UserGroup
- UserGroupCondition

Some array definitions are different from the ones listed above. These arrays have a type definition for the element other than what appears in the array name. For example, the `ArrayOfDate` is defined as:

```
<xsd:complexType name="ArrayOfDate">
 <xsd:sequence>
 <xsd:element name="Date" type="typens:string"
 maxOccurs="unbounded" minOccurs="0" />
 </xsd:sequence>
</xsd:complexType>
```

In this definition, the element name is `Date`, but its type is defined as a string. The `ArrayOfDate` type is defined as an array of string elements. The arrays in this format are listed in Table 14-1, along with the associated element type.

**Table 14-1** Non-standard arrays

Array type	Element type
Date	string
String	string
Int	int
Component	ComponentType
Long	long

---

## ArrayOfAggregation

Obsolete since iHub Release 2.

## ArrayOfArchiveRule

ArrayOfArchiveRule is a complex data type that represents an array of [ArchiveRule](#) objects.

---

## ArrayOfArgument

Obsolete since iHub Release 2.

---

## ArrayOfAttachment

ArrayOfAttachment is a complex data type that represents an array of [Attachment](#) objects.

---

## ArrayOfBookMark

ArrayOfBookMark is a complex data type that represents an array of [BookMark](#) objects.

---

## ArrayOfCapabilities

ArrayOfCapabilities is a complex data type that represents an array of [Capabilities](#) objects.

---

## ArrayOfChannel

Obsolete since iHub Release 3.

---

## ArrayOfChannelCondition

Obsolete since iHub Release 3.

---

## ArrayOfColumnDefinition

Obsolete since iHub Release 2.

---

## ArrayOfColumnSchema

ArrayOfColumnSchema is a complex data type that represents an array of [ColumnSchema](#) objects.

---

## ArrayOfComponent

Obsolete since iHub Release 2.

---

## ArrayOfComponentIdentifier

Obsolete since iHub Release 2.

---

## ArrayOfCounterInfo

ArrayOfCounterInfo is a complex data type that represents an array of [CounterInfo](#) objects.

---

## ArrayOfDataExtractionFormat

ArrayOfDataExtractionFormat is a complex data type that represents an array of [DataExtractionFormat](#) objects.

---

## ArrayOfDataFilterCondition

ArrayOfDataFilterCondition is a complex data type that represents an array of [DataFilterCondition](#) objects.

## ArrayOfDataRow

ArrayOfDataRow is a complex data type that represents an array of [DataRow](#) objects.

---

## ArrayOfDataSortColumn

ArrayOfDataSortColumn is a complex data type that represents an array of [DataSortColumn](#) objects.

---

## ArrayOfDate

ArrayOfDate is a complex data type that represents an array of [Date](#) objects.

---

## ArrayOfDocumentConversionOptions

ArrayOfDocumentConversionOptions is a complex data type that represents an array of [DocumentConversionOptions](#).

---

## ArrayOfFieldDefinition

ArrayOfFieldDefinition is a complex data type that represents an array of [FieldDefinition](#) objects.

---

## ArrayOfFile

ArrayOfFile is a complex data type that represents an array of [File](#) objects.

---

## ArrayOfFileCondition

ArrayOfFileCondition is a complex data type that represents an array of [FileCondition](#) objects.

---

## ArrayOfFileContent

ArrayOfFileContent is a complex data type that represents an array of [FileContent](#) objects.

---

## ArrayOfFileType

ArrayOfFileType is a complex data type that represents an array of [FileType](#) objects.

---

## ArrayOfFilterCriteria

Obsolete since iHub Release 2.

---

## ArrayOfGroup

Obsolete since iHub Release 2.

---

## ArrayOfGroupCondition

Obsolete since iHub Release 2.

---

## ArrayOfGrouping

Obsolete since iHub Release 2.

---

## ArrayOfInt

ArrayOfInt is a complex data type that represents an array of [Int](#) objects.

---

## ArrayOfIOCacheDBIndexConstraint

Obsolete since iHub Release 2.

## ArrayOfJobCondition

ArrayOfJobCondition is a complex data type that represents an array of [JobCondition](#) objects.

---

## ArrayOfJobNotice

ArrayOfJobNotice is a complex data type that represents an array of [JobNotice](#) objects.

---

## ArrayOfJobNoticeCondition

ArrayOfJobNoticeCondition is a complex data type that represents an array of [JobNoticeCondition](#) objects.

---

## ArrayOfJobProperties

ArrayOfJobProperties is a complex data type that represents an array of [JobProperties](#) objects.

---

## ArrayOfJobScheduleCondition

ArrayOfJobScheduleCondition is a complex data type that represents an array of [JobScheduleCondition](#) objects.

---

## ArrayOfJobScheduleDetail

ArrayOfJobScheduleDetail is a complex data type that represents an array of [JobScheduleDetail](#) objects.

---

## ArrayOfLong

ArrayOfLong is a complex data type that represents an array of [Long](#) objects.

---

---

## ArrayOfLicenseOption

ArrayOfLicenseOption is a complex data type that represents an array of [LicenseOption](#) objects.

---

## ArrayOfMDSInfo

ArrayOfMDSInfo is a complex data type that represents an array of [MDSInfo](#) objects.

---

## ArrayOfNameValuePair

ArrayOfNameValuePair is a complex data type that represents an array of [NameValuePair](#) objects.

---

## ArrayOfParameterDefinition

ArrayOfParameterDefinition is a complex data type that represents an array of [ParameterDefinition](#) objects.

---

## ArrayOfParameterValue

ArrayOfParameterValue is a complex data type that represents an array of [ParameterValue](#) objects.

---

## ArrayOfPendingSyncJob

ArrayOfPendingSyncJob is a complex data type that represents an array of [PendingSyncJob](#) objects.

---

## ArrayOfPermission

ArrayOfPermission is a complex data type that represents an array of [Permission](#) objects.

## ArrayOfPrinter

ArrayOfPrinter is a complex data type that represents an array of [Printer](#) objects.

---

## ArrayOfPrinterOptions

ArrayOfPrinterOptions is a complex data type that represents an array of [PrinterOptions](#) objects.

---

## ArrayOfPropertyValue

ArrayOfPropertyValue is a complex data type that represents an array of [PropertyValue](#) objects.

---

## ArrayOfRecord

Obsolete since iHub Release 2.

---

## ArrayOfResourceGroup

ArrayOfResourceGroup is a complex data type that represents an array of [ResourceGroup](#) objects.

---

## ArrayOfResourceGroupSettings

ArrayOfResourceGroupSettings is a complex data type that represents an array of [ResourceGroupSettings](#) objects.

---

## ArrayOfResultSetSchema

ArrayOfResultSetSchema is a complex data type that represents an array of [ResultSetSchema](#) objects.

---

## ArrayOfRole

Deprecated since iHub Release 3. Replaced by [ArrayOfUserGroup](#).

---

## ArrayOfRoleCondition

Deprecated since iHub Release 3. Replaced by [ArrayOfUserGroupCondition](#).

---

## ArrayOfRunningJob

ArrayOfRunningJob is a complex data type that represents an array of [RunningJob](#) in the Factory.

---

## ArrayOfServerInformation

ArrayOfServerInformation is a complex data type that represents an array of [ServerInformation](#) objects.

---

## ArrayOfServerResourceGroupSetting

ArrayOfServerResourceGroupSetting is a complex data type that represents an array of [ServerResourceGroupSetting](#) objects.

---

## ArrayOfService

ArrayOfService is a complex data type that represents an array of [Service](#) objects.

---

## ArrayOfString

ArrayOfString is a complex data type that represents an array of [String](#) objects.

## ArrayOfUser

ArrayOfUser is a complex data type that represents an array of [User](#) objects.

---

## ArrayOfUserCondition

ArrayOfUserCondition is a complex data type that represents an array of [UserCondition](#) objects.

---

## ArrayOfUserGroup

ArrayOfUserGroup is a complex data type that represents an array of [UserGroup](#) objects.

---

## ArrayOfUserGroupCondition

ArrayOfUserGroupCondition is a complex data type that represents an array of [UserGroupCondition](#) objects.

---

## Attachment

A complex data type that describes the object in the attachment and contains the attachment as binary data.

**Elements** **ContentId**

[String](#). Maps to the attachment's MIME header. ContentId is required.

**ContentType**

[String](#). The type of file to upload, such as binary.

**ContentLength**

[Long](#). Optional. The length of the object.

**ContentEncoding**

[String](#). Optional. The encoding the object uses. Cannot exceed 10 characters.

**Locale**

[String](#). Optional. The object locale.

**ContentData**

[Base64Binary](#). Optional. The attachment as binary data. Use ContentData to embed the file in the request.

---

## Base64Binary

A standard XML base64Binary data type.

---

## BookMark

A complex data type that specifies the bookmark value, display name, and element type in a report document.

**Elements** **BookMarkValue**

[String](#). The bookmark value, such as the page number.

**DisplayName**

[String](#). The display name of the bookmark.

**ElementType**

[String](#). The element or component type.

**BookMarkType**

[String](#). Optional. The bookmark type.

---

## Boolean

A standard XML Boolean data type with a value of True or False.

---

## CancelJobStatus

A simple data type that represents the status of a request to cancel a synchronous report.

**Element** [String](#). One of the following values:**Succeeded**

The synchronous report generation was successfully cancelled.

**Failed**

The request to cancel a synchronous report failed.

## Capabilities

### InActive

The synchronous report generation is complete and cannot be cancelled.

---

## Capabilities

A complex data type that represents the tasks that a user is permitted to perform using a specified tool. The tasks for each tool are grouped into a category.

### Elements Category

[String](#). The name of a category, for example InteractiveViewer.

### CapabilityNames

[ArrayOfString](#). The list of tasks that the user is permitted to perform. The available capability names differ according to the category.

---

## Channel

Obsolete since iHub Release 3.

## ChannelCondition

Obsolete since iHub Release 3.

---

## ChannelField

Obsolete since iHub Release 3.

---

## ChannelSearch

Obsolete since iHub Release 3.

---

## ColumnDefinition

Obsolete since iHub Release 2.

---

---

## ColumnDetail

A complex data type that describes the type of data within a column.

**Elements** **name**

**String**. The name of the column.

**type**

**TypeName**. The type of data within the column.

**displayName**

**String**. The display name of the column.

---

## ColumnSchema

A complex data type that describes the schema of a column.

**Elements** **Name**

**String**. The column name.

**Alias**

**String**. Optional. User-defined name for column.

**DataType**

**Int**. Optional. The data type of the column.

**TypeName**

**String**. The name of the data type.

**Label**

**String**. Optional. The column label.

**Visibility**

**Boolean**. Optional. Specifies whether column is visible. The default value is True.

**AllowExport**

**Boolean**. Optional. Specifies whether to allow exporting the column. The default value is True.

---

## ComponentIdentifier

Obsolete since iHub Release 2.

---

## ComponentType

Obsolete since iHub Release 2.

---

## CopyFile

A complex data type describes a copied file or list of files. To copy a single file or folder, specify Name or Id. To copy a list of files or folders, specify NameList or IdList. To copy files or folders that match specific conditions, specify Search.

**Elements**

**Target**

[String](#). The new location for the file or folder. The following rules apply:

- If Target is a file, the operation fails if the source contains a folder.
- If Target is a folder that does not exist, a folder is created.
- If Target is a folder and the source contains a single folder, the contents of the source folder are copied to the target folder and merged with target folder contents.
- If Target is a folder and the source contains a single file or multiple files and folders, the source files and folders are copied to the target folders. All source folders are copied as children of the target folder.

**WorkingFolderName**

[String](#). The name of the working folder of the file or folder to copy. Specify either WorkingFolderName or WorkingFolderId.

**WorkingFolderId**

[String](#). The ID of the working folder of the file or folder to copy. Specify either WorkingFolderId or WorkingFolderName.

**Recursive**

[Boolean](#). Specifies whether to search subfolders. If True, the search includes subfolders. The default value is False.

**Search**

[FileSearch](#). The search condition that specifies which folders or files to copy.

**IdList**

[ArrayOfString](#). The list of file or folder IDs to copy. Specify either IdList or NameList.

**NameList**

[ArrayOfString](#). The list of file or folder names to copy. Specify either NameList or IdList.

**Id**

[String](#). The ID of the single file or folder to copy. Specify either Id or Name.

**Name**

[String](#). The name of the single file or folder to copy. Specify either Name or Id.

**ReplaceExisting**

[Boolean](#). If True, the copied file replaces the existing file, if one exists. If the existing file has any dependencies, it is not replaced regardless of the ReplaceExisting setting. If False or if the existing file has any dependencies, a new version of the file is created. The default value is True.

**MaxVersions**

[Long](#). The maximum number of versions to create. MaxVersions applies only for files and is ignored for folders.

**LatestVersionOnly**

[Boolean](#). Specifies whether all versions of the file are copied or only the latest version. Used only when a Search tag is specified. If True, only the latest version of the file that matches the search criteria is copied. If False, all versions of the file are copied. The default value is False.

---

## CounterInfo

A complex data type that describes a counter.

**Elements****CounterId**

[Long](#). The ID of the counter.

**CounterName**

[String](#). The name of the counter.

**CounterValue**

[Long](#). The value of the counter.

---

## CreateChannel

Obsolete since iHub Release 3.

---

## CreateFileType

Adds a file type. Available only to users with the Administrator user group.

## CreateFolder

### Elements

#### **FileType**

**FileType**. The properties of the file type to add. The following properties are required:

- Name
- Extension
- IsNative
- IsExecutable
- OutputType
- IsPrintable

#### **IgnoreDup**

**Boolean**. Specifies whether to report an error when creating the file type if one with the same name already exists. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. If False, BIRT iHub reports an error. The default value is False.

---

## CreateFolder

Creates a folder in a volume into which you are currently logged in. To create a folder, you must have permission to add folders to the volume.

### Elements

#### **WorkingFolderName**

**String**. The name of the working folder for the new folder. Specify either WorkingFolderName or WorkingFolderId.

#### **WorkingFolderId**

**String**. The ID of the working folder for the new folder. Specify either WorkingFolderId or WorkingFolderName.

#### **FolderName**

**String**. The name of the new folder, relative to the working folder, if specified. If you do not specify a working folder, you must specify a full path.

#### **Description**

**String**. The description of the folder.

#### **IgnoreDup**

**Boolean**. Specifies whether to report an error when creating the folder if one with the same name already exists. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. If False, BIRT iHub reports an error. The default value is False.

---

## CreateGroup

Obsolete since iHub Release 3.

---

## CreateRole

Deprecated since iHub Release 3. Replaced by [CreateUserGroup](#).

---

## CreateUser

Creates a user. Available only to users in the Administrator user group.

**Elements** **User**

[User](#). The properties of the user to create. Only a user name is required.

**IgnoreDup**

[Boolean](#). Specifies whether to report an error when creating the user if one with the same name already exists. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. If False, BIRT iHub reports an error. The default value is False.

---

## CreateUserGroup

Creates a user group. Available only to users in the Administrator user group.

**Elements** **UserGroup**

[UserGroup](#). The properties of the user group to create. Only a user group name is required.

**ProductFamilies**

[ArrayOfString](#). The list of product families associated with this user group.

**IgnoreDup**

[Boolean](#). Specifies whether to report an error when creating the user group if one with the same name already exists. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. If False, BIRT iHub reports an error. The default value is False.

---

## CustomEvent

A complex data type that specifies information used within a custom event.

## Daily

**Element** **EventParameter**

**String**. A value used within the custom event.

---

## Daily

A complex data type that describes daily types of job scheduling.

**Elements** **FrequencyInDays**

**Long**. The number of times a job is run daily in days.

**OnceADay**

**String**. Optional. A string representing when a job is to run once a day.

**Repeat**

**Repeat**. Optional. The number of times the schedule is repeated.

---

## DatabaseConnectionDefinition

Obsolete since iHub Release 2.

---

## DataCell

A complex data type describing the type of data within a data cell.

**Elements** One of the following elements:**int**

**Int**. An integer value.

**sht**

**Short**. A short value.

**dbl**

**Double**. A double value.

**dbn**

**acDouble**. An Actuate double value.

**cur**

**String**. A currency value.

**dtm**

**DateTime**. An date-and-time value.

**str****String.** A string value.**bln****Boolean.** A Boolean value.**nll****acNull.** The null value.

## DataExtractionFormat

A complex data type that describes the format of a file and its mime type.

**Elements** **OutputFormat**

**String.** The format of the file.

**MimeType**

**String.** The file mime type.

## DataFilterCondition

A complex data type that describes a filter condition.

**Elements** **ColumnName**

**String.** The name of the column to filter.

**Operation**

**String.** The filtering operation.

**Operand1**

**String.** The first operand of the filter.

**Operand2**

**String.** Optional. The second operand of the filter.

**Operand3**

**ArrayOfString.** Optional. A list of operands for the filter.

## DataRow

A complex data type that contains the information from a data row.

**Element** **Cell**

**DataCell.** A data cell from the row. A DataRow contains as many Cell values as there are fields in the row.

---

## DataSchema

A complex data type that describes a data schema by column.

**Element** **Column**

**ColumnDetail**. The schema of the information stored within a column. A DataSchema contains as many Column values as there are fields in a row.

---

## DataSortColumn

A complex data type that describes a sorted data column.

**Elements** **ColumnName**

**String**. The name of the sorted column.

**SortDirection**

**String**. The direction of the sort. Valid values are:

- ASC - Ascending
- DES - Descending

---

## DataSourceType

A simple data type that specifies the type of file in which a parameter exists.

**Element** **String**. One of the following values:

**InfoObject**

An information object.

**ABInfoObject**

An Actuate Basic information object. Obsolete since iHub Release 2.

---

## DataType

A simple data type that specifies a data type.

**Element** **String**. One of the following values:

**Currency**

A Currency data type.

**Date**

A **DateTime** data type.

**DateOnly**

A [Date](#) data type.

**Time**

A Time data type.

**Double**

A [Double](#) data type.

**Integer**

An [Int](#) data type.

**String**

A [String](#) data type.

**Boolean**

A [Boolean](#) data type.

**Structure**

A structure. Obsolete since Actuate release 11.

**Table**

A table. Obsolete since Actuate release 11.

---

## Date

Date is a standard XML Date data type that displays dates in the format CCYY-MM-DD.

---

## DateTime

DateTime is a standard XML TimeInstant data type that shows the date and time in the format CCYY-MM-DDThh:mm:ss-sss. DateTime ignores milliseconds, if any.

---

## Decimal

Decimal is a standard XML Decimal data type that represents a number having arbitrary precision up to a maximum of 18 digits.

## DeleteChannel

Obsolete since iHub Release 3.

---

## DeleteFile

Deletes files or folders. To delete a single file or folder, specify Name or Id. To delete several files or folders, specify NameList or IdList. To delete files or folders that match the specified conditions, specify Search.

**Elements** **WorkingFolderId**

**String**. The ID of the working folder of the file or folder to delete. Specify either WorkingFolderId or WorkingFolderName.

**WorkingFolderName**

**String**. The name of the working folder of the file or folder to delete. Specify either WorkingFolderName or WorkingFolderId.

**Recursive**

**Boolean**. Specifies whether to delete subfolders. If True, subfolders are deleted. If False, only the specified folder is deleted. The default value is False.

**LatestVersionOnly**

**Boolean**. Specifies whether to delete only the latest version of the file. If True, only the latest version of the file is deleted. The default value is False.

**Search**

**FileSearch**. The search condition that specifies which folders or files to delete.

**IdList**

**ArrayOfString**. The list of file or folder IDs to delete. Specify either IdList or NameList.

**NameList**

**ArrayOfString**. The list of file or folder names to delete. Specify either NameList or IdList.

**Id**

**String**. The ID of the single file or folder to delete. Specify either Id or Name.

**Name**

**String**. The name of the single file or folder to delete. Specify either Name or Id.

**IgnoreMissing**

**Boolean**. Specifies what to do if the specified file or folder does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

---

## DeleteFileType

Deletes file types. To delete a single file type, specify Name or Id. To delete several file types, specify NameList or IdList.

**Elements** **NameList**

[ArrayOfString](#). The list of file types to delete.

**Name**

[String](#). The name of a single file type to delete.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified file type does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

---

## DeleteGroup

Obsolete since iHub Release 3.

---

## DeleteJob

Deletes scheduled, completed, cancelled, or failed jobs. If an instance of a scheduled report is running when the request is submitted, an exception is thrown.

To delete a single job, specify Id. To delete several jobs, specify IdList. To delete jobs that match the specified conditions, specify Search.

**Elements** **Search**

[JobSearch](#). The search conditions. If conditions apply to multiple fields, use ConditionArray.

**IdList**

[ArrayOfString](#). The list of job IDs to delete.

**Id**

[String](#). The ID of the single job to delete.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified job does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

## DeleteJobNotices

### **IgnoreActiveJob**

**Boolean.** Flag indicating whether to delete a job if it is active.

---

## DeleteJobNotices

Deletes job notices. A user in the Administrator user group can delete all job notices. To delete all job notices, do not specify the user or group.

#### **Element** **Search**

**JobNoticeSearch.** The search conditions. If conditions apply to multiple fields, use ConditionArray.

---

## DeleteJobSchedule

Deletes a job schedule. If an instance of a scheduled report is running when the request is submitted, an exception is thrown.

To delete a job schedule, specify Id. To delete several jobs, specify IdList. To delete jobs that match the specified conditions, specify Search.

#### **Elements** **Search**

**JobScheduleSearch.** The search conditions. If conditions apply to multiple fields, use ConditionArray.

#### **IdList**

**ArrayOfString.** The list of job schedule IDs to delete.

#### **Id**

**String.** The ID of the single job schedule to delete.

#### **IgnoreMissing**

**Boolean.** Specifies what to do if the specified job does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

#### **IgnoreActiveJob**

**Boolean.** Flag indicating whether to delete a job if it is active.

---

## DeleteRole

Deprecated since iHub Release 3. Replaced by [DeleteUserGroup](#).

---

## DeleteUser

Deletes users. To delete a single user, specify Name or Id. To delete several users, specify NameList or IdList. To delete users that match the specified conditions, specify Search.

**Elements****Search**

[UserSearch](#). The search condition that specifies which users to delete.

**IdList**

[ArrayOfString](#). The list of user IDs to delete. Specify either IdList or NameList.

**NameList**

[ArrayOfString](#). The list of user names to delete. Specify either NameList or IdList.

**Id**

[String](#). The ID of the single user to delete. Specify either Id or Name.

**Name**

[String](#). The name of the single user to delete. Specify either Name or Id.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified user does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

**PurgeUserInfo**

[Boolean](#). Purges user information from the system.

---

## DeleteUserGroup

Deletes a user group. Available only to users with the Administrator user group.

**Elements****Search**

[UserGroupSearch](#). The search condition that specifies which user groups to delete.

**IdList**

[ArrayOfString](#). The list of user group IDs to delete. Specify either IdList or NameList.

**NameList**

[ArrayOfString](#). The list of user group names to delete. Specify either NameList or IdList.

**Id**

[String](#). The ID of the single user group to delete. Specify either Id or Name.

## DocumentConversionOptions

### Name

[String](#). The name of the single user group to delete. Specify either Name or Id.

### IgnoreMissing

[Boolean](#). Specifies what to do if the specified user group does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

---

## DocumentConversionOptions

A complex data type that describes the conversion options of a file.

### Elements

#### FileType

[String](#). The file type of the file.

#### OutputFormat

[String](#). The output format of the file.

#### MimeType

[String](#). The mime type of the file.

#### Options

[ArrayOfParameterDefinition](#). The list of conversion options.

---

## Double

Double is a standard XML double data type, which is a signed 64-bit floating-point number.

---

## Event

A complex type that describes an event and its status.

### Elements

#### FileEvent

[FileEvent](#). Optional. Specifies information for a file event. Specify one of this value, JobEvent, or CustomEvent.

#### JobEvent

[JobEvent](#). Optional. Specifies information for a job event. Specify one of this value, FileEvent, or CustomEvent.

#### CustomEvent

[CustomEvent](#). Optional. Specifies information for a custom event. Specify one of this value, FileEvent, or JobEvent.

**EventName**

[String](#). The name of the event.

**EventType**

[EventType](#). The type of event.

**PollingInterval**

[Long](#). Optional. Specifies the amount of time to wait between event status checks. The minimum value is 10 seconds.

**PollingDuration**

[Long](#). Optional. Specifies the amount of time to check the event status.

**LagTime**

[Long](#). Optional. Specifies lag time value for the event.

**EventStatus**

[String](#). Optional. The current status of the event. Valid values are:

- Uninitialized
- Polling
- Satisfied
- Expired

## EventOptions

A complex data type that describes polling and other options for an event.

**Elements** **DefaultEventPollingInterval**

[Long](#). Optional. The amount of time to wait between polling the event.

**DefaultEventPollingDuration**

[Long](#). Optional. The duration of time to poll for an event.

**DefaultEventLagTime**

[Long](#). Optional. The amount of lag time for the event.

**EnableCustomEventService**

[Boolean](#). Optional. A flag indicating whether to enable the custom event service.

## EventType

A simple data type that represents a type of event.

**Element** [String](#). One of the following values:

## ExecuteReportStatus

### **FileEvent**

A file type event.

### **JobEvent**

A job type event.

### **CustomEvent**

A custom type event.

### **NoEvent**

No event.

---

## ExecuteReportStatus

A simple data type that represents the status of report execution.

**Element** [String](#). One of the following values:

### **Done**

The report execution succeeded.

### **Failed**

The report execution failed.

### **FirstPage**

The first page is complete. Applies only if progressive viewing is enabled.

### **Pending**

The job is either in the queue or in the process of generating. Applies only if WaitTime is specified.

---

## ExternalTranslatedUserNames

A complex data type that represents a member of the Administrators user group.

**Element** [Administrator](#)

[String](#). The external name of the administrator user.

---

## ExternalTranslatedRoleNames

Deprecated since iHub Release 3. Replaced by

[ExternalTranslatedUserGroupNames](#).

---

## ExternalTranslatedUserGroupNames

A complex data type that represents one of the following user groups:

- Administrator
- Operator
- All

**Elements** **Administrator**

**String**. The external name of the administrator user group.

**Operator**

**String**. The external name of the operator user group.

**All**

**String**. The external name of the all user group.

---

## FieldDefinition

A complex data type that describes a scalar parameter.

**Elements** **Name**

**String**. The name of the parameter.

**DisplayName**

**String**. Optional. The display name of the parameter.

**DataType**

**ScalarDataType**. Optional. The data type of the parameter. Valid values are:

- Currency
- Date
- Double
- Integer
- String
- Boolean

**IsHidden**

**Boolean**. Optional. Specifies whether the parameter is hidden.

**IsRequired**

**Boolean**. Optional. Specifies whether the parameter is required.

## FieldValue

### DefaultValue

[String](#). Optional. The default value of the parameter.

### SelectValueList

[ArrayOfString](#). Optional. The list of available parameter values.

### FieldControlType

[String](#). Optional. The type of control used to represent the parameter. Valid values are:

- ControlListAllowNew  
A text box
- ControlList  
A drop-down list

### SelectNameValueList

[ArrayOfNameValuePair](#). Optional. A list of name-value pairs used within the field.

---

## FieldValue

Obsolete since iHub Release 2.

---

## File

A complex data type that describes a file.

### Elements

#### Id

[String](#). Optional. The file ID.

#### Name

[String](#). Optional. The name of the file. Actuate's internal data store imposes a fixed upper limit on the length of certain text strings.

#### FileType

[String](#). Optional. The file type.

#### Description

[String](#). Optional. The description of the file.

#### PageCount

[Long](#). Optional. The number of pages in the file.

#### Size

[Long](#). Optional. The size of the file.

**TimeStamp**

[DateTime](#). Optional. The time the file was created or modified, in Coordinated Universal Time (UTC).

**Version**

[Long](#). Optional. The version number.

**VersionName**

[String](#). Optional. The version name.

**Owner**

[String](#). Optional. The owner of the file.

**UserPermissions**

[String](#). Optional. The current user permissions for the file.

**AccessType**

[FileAccess](#). Optional. The file's access type. Valid values are:

- Private

Only the owner of the file and an administrator can access the file.

- Shared

All users and user groups specified in the access control list (ACL) for the file can access the file.

## FileAccess

A simple data type that specifies the file's access type.

**Element** [String](#). One of the following values:

**Private**

Only the owner of the file and an administrator can access the file.

**Shared**

All users and user groups specified in the access control list (ACL) for the file can access the file.

## FileCondition

A complex data type that represents the fields on which a search can be performed and the condition to match.

**Elements** [Field](#)

[FileField](#). File field on which to perform a search.

## FileContent

### Match

[String](#). The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

```
05\‐25\‐04Report.rptdocument
05 [-] 25 [-] 04Report.rptdocument
```

---

## FileContent

A complex data type that represents a list of attached files or the content of embedded files.

**Elements**

**File**

[File](#). The attached file.

**Content**

[Attachment](#). The content of an embedded file.

---

## FileEvent

A complex data type that contains information pertaining to file type events.

**Element** **MonitoredFilePath**

[String](#). The file path of the file the event is monitoring.

---

## FileField

A simple type that describes different fields that may exist for a file.

**Element** **String**. One of the following values:

**Name**

The name of the file.

**FileType**

The file type.

**Description**

The description of the file.

**PageCount**

The number of pages in the file.

**Size**

The size of the file.

**TimeStamp**

The time the file was created or modified, in Coordinated Universal Time (UTC).

**Version**

Optional. The version number.

**VersionName**

The version name.

**Owner**

The owner of the file.

---

## FileSearch

A complex data type that represents a file search.

**Elements****Condition**

[FileCondition](#). Optional. The search condition. Specify either this parameter or ConditionArray.

**ConditionArray**

[ArrayOfFileCondition](#). Optional. An array of search conditions. Use if search conditions apply to multiple fields.

**Owner**

[String](#). Optional. The file owner.

**DependentFileName**

[String](#). Optional. The name of the dependent file. Specify one of this parameter, DependentFileDialog, RequiredFileName, or RequiredFileDialog.

**DependentFileDialog**

[String](#). Optional. The ID of the dependent file. Specify one of this parameter, DependentFileName, RequiredFileName, or RequiredFileDialog.

**RequiredFileName**

[String](#). Optional. The name of the required file. Specify one of this parameter, DependentFileName, DependentFileDialog, or RequiredFileDialog.

**RequiredFileDialog**

[String](#). Optional. The ID of the required file. Specify one of this parameter, DependentFileName, DependentFileDialog, or RequiredFileName.

## FileType

### PrivilegeFilter

**PrivilegeFilter**. Optional. The privileges for which to search. Use PrivilegeFilter to determine whether the specified user or user group has the specified privileges on the file.

### AccessType

**FileAccess**. Optional. The file's access type. Valid values are:

- Private  
Only the owner of the file and an administrator can access the file.
- Shared  
All users and user groups specified in the access control list (ACL) for the file can access the file.

### FetchSize

**Int**. Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

### FetchDirection

**Boolean**. Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

### CountLimit

**Int**. Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

### FetchHandle

**String**. Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specifies the same search criteria as in the original call.

### IncludeHiddenObject

**Boolean**. Optional. Flag indicating if search should include hidden objects.

---

## FileType

A complex data type that describes a file type. Specify FileType in the SOAP header for all execute, submit, and view IDAPI requests.

### Elements

#### Name

**String**. Optional. The file type name.

#### IsNative

**Boolean**. Optional. Specifies whether the file is an internal Actuate type. IsNative is read-only. Providing an input value for this attribute in CreateFileType or UpdateFileType causes SetAttributes to be ignored.

**IsExecutable**

**Boolean**. Optional. Specifies whether the file is executable. If False, the file type is set to Document file type. The OutputType and ExportBeforeViewing attributes do not apply to Document file type.

**IsPrintable**

**Boolean**. Optional. Specifies whether the file is printable. If the file type is Executable, IsPrintable refers to the output file.

**IsRequired**

**Boolean**. Optional. Specifies whether the file is required.

**OutputType**

**String**. Optional. The file type for the output file. Required if the file type is Executable.

**LocalExtension**

**String**. Optional. The local extension.

**DisplayType**

**String**. Optional. Specifies either Simple or Advanced display types.

**ShortDescription**

**String**. Optional. The short description of the file type.

**LongDescription**

**String**. Optional. The long description of the file type.

**SmallImageURL**

**String**. Optional. The URL of the small image for the file.

**LargeImageURL**

**String**. Optional. The URL of the large image for the file.

**ExportBeforeViewing**

**Boolean**. Optional. Specifies whether the file is exported before viewing.

**DriverName**

**String**. Optional. The name of the driver. Required if file type is Executable or Printable.

**MutexClass**

**String**. Optional. The mutex class name.

**ContentType**

**String**. Optional. The content type.

**EnableAutoParamCollection**

**Boolean**. Optional. True enables automatic parameter collection for the file type.

## FilterCriteria

### **IsCompoundDoc**

**Boolean.** Optional. Specifies whether the file is a compound document. The default value is False.

### **AllowViewTimeParameter**

**Boolean.** Optional. Specifies whether to allow view-time parameters. The default value is True.

---

## FilterCriteria

Obsolete since iHub Release 2.

---

## Group

Obsolete since iHub Release 2.

---

## GroupCondition

Obsolete since iHub Release 2.

---

## GroupField

Obsolete since iHub Release 2.

---

## Grouping

Obsolete since iHub Release 2.

---

## GroupSearch

Obsolete since iHub Release 2.

---

## Header

The SOAP header that contains authentication data, locale information, and other required or optional data.

**Elements****AuthId**

A system-generated, encrypted [String](#). All requests except Login requests must have a valid AuthId in the SOAP header. The header passes this identifier to BIRT iHub for validation.

**TargetVolume**

[String](#). Optional. The volume to which to direct the request.

**Locale**

[String](#). Optional. The format of Locale is <ll>\_<CC>, where <ll> is the two-character language code and <CC> is the two-character country code of the locale. For example, fr\_CA is the locale code for the French language used in Canada. Locale is used to format data using the language, date and time conventions, currency and other locale-specific conventions.

**ConnectionHandle**

[String](#). Optional. Supports keeping a connection open to view a persistent report.

**TargetServer**

[String](#). Optional. Refers to the BIRT iHub within a cluster to which to direct the request.

**DelayFlush**

[Boolean](#). Optional. BIRT iHub writes updates to the disk when the value is False.

**FileType**

[String](#). Optional. Supports specifying the file type to run, such as a BIRT design or HTML file.

**TargetResourceGroup**

[String](#). Optional. Supports assigning a synchronous report generation request to a specific resource group at run time.

**RequestID**

[String](#). Optional. A unique value that identifies the SOAP message.

---

## InfoObjectData

A complex data type that describes the data from a BIRT information object.

**Elements****DataSchema**

The schema for the data rows.

## InfoObjectDataFormat

### **DataRows**

The data rows from the information object.

---

## InfoObjectDataFormat

A simple data type that describes an information object's data format.

**Element** NMOKEN. One of the following values:

### **XML**

The file is in XML format.

### **CSV**

The file is in comma separated values format.

---

## Int

A standard XML Integer data type that represents a number. Int derives from the [Decimal](#) data type by fixing the value of scale at 0.

---

## IOCacheDBIndexConstraint

Obsolete since iHub Release 2.

---

## IOCacheDefinition

Obsolete since iHub Release 2.

---

## IOCacheState

Obsolete since iHub Release 2.

---

## JobCondition

A complex data type that represents the field on which to perform a search and the condition to match.

**Elements** **Field**

[JobField](#). An element that includes a field on which to perform a search.

**Match**

[String](#). The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

05\-\-25\-\-04Report.rptdocument  
05 [-] 25 [-] 04Report.rptdocument

## JobEvent

A complex data type that represents the information pertaining to a job type event.

**Elements** **JobId**

[String](#). The ID of the job.

**JobName**

[String](#). Optional. The job name.

**JobStatus**

[ArrayOfString](#). Optional. The job status.

## JobField

A simple data type that represents the fields on which a search for jobs can be performed.

**Element** [String](#). One of the following values:**JobName**

The job name.

**Owner**

The owner of the job.

**JobType**

The type of job. Valid values are:

- RunReport
- PrintReport
- RunAndPrintReport
- ConvertReport

## JobInputDetail

### Priority

The job priority.

### RoutedToNode

The node to which the job is routed.

### StartTime

The start time.

### DurationSeconds

The job duration.

### CompletionTime

The time the job is completed.

### State

The state of the job. Valid values are:

- Succeeded
- Failed
- Cancelled

### NotifyCount

The number of notifications sent about the job.

### OutputFileSize

The size of the output file.

---

## JobInputDetail

A complex data type that describes the job input and output files.

### Elements

#### InputFileVersionName

[String](#). Optional. The input file version.

#### OutputFileVersionName

[String](#). Optional. The output file version.

#### ReplaceLatestVersion

[Boolean](#). Optional. Specifies whether to replace the latest version of the file with the current version.

#### OutputMaxVersion

[Int](#). Optional. The maximum number of versions to keep after a new version is generated.

**ValueFileType**

[String](#). Optional. The type of a value file. If present, the value is either Temporary or Permanent.

**ValueFileVersionName**

[String](#). Optional. The value file name.

**IsBundled**

[Boolean](#). Optional. Specifies whether the output object is bundled with the input object.

**RetryOption**

[RetryOptionType](#). Optional. The retry settings. Valid values are:

- Disabled
- VolumeDefault
- Specified

**MaxRetryCount**

[Int](#). Optional. The maximum number of retry attempts.

**RetryInterval**

[Int](#). Optional. The interval between retry attempts. Measured in seconds.

**MaxVersions**

[Long](#). Optional. The maximum number of versions to keep in the volume.

**NeverExpire**

[Boolean](#). Optional. Specifies whether the item expires.

**ArchiveRuleInherited**

[Boolean](#). Optional. Specifies whether the archive rules are inherited from another object.

**ExpirationAge**

[Int](#). Optional. Specifies the expiration age for the object.

**ExpirationDate**

[DateTime](#). Optional. The date when the job expires.

**ArchiveOnExpire**

[Boolean](#). Optional. Specifies whether the object is archived before it is expired.

**KeepWorkspace**

[Boolean](#). Optional. Specifies whether to keep or remove the workspace directory after executing the job.

**DriverTimeout**

[Int](#). Optional. The time for the driver to return from executing the job.

**PollingInterval**

**Int**. Optional. The time interval to get status messages. The minimum value is 10 seconds.

**DebugInstruction**

**String**. Optional. The debug instructions.

**SendSuccessNotice**

**Boolean**. Optional. Specifies whether success notices are sent if the job succeeds. Used only if OverrideRecipientPref is True.

**SendFailureNotice**

**Boolean**. Optional. Specifies whether failure notices are sent if the job fails. Used only if OverrideRecipientPref is True.

**SendEmailForSuccess**

**Boolean**. Optional. Specifies whether e-mail notifications are sent if the job succeeds. Used only if OverrideRecipientPref is True. If SendEmailForSuccess is True, e-mail notifications are sent to specified users if the job succeeds. The default value is False.

**SendEmailForFailure**

**Boolean**. Optional. Specifies whether e-mail notifications are sent if the job fails. Used only if OverrideRecipientPref is True. If SendEmailForFailure is True, e-mail notifications are sent to specified users if the job fails. The default value is False.

**AttachReportInEmail**

**Boolean**. Optional. Specifies whether the output file is attached to the e-mail notification for successful jobs. Used only if OverrideRecipientPref is True. If AttachReportInEmail is True, the output file is attached to the e-mail notification. If False, only a link to the output file is sent. Specify the format for the attachment in the EmailFormat element. The default value is False.

**OverrideRecipientPref**

**Boolean**. Optional. Specifies whether e-mail notifications and output attachments are sent according to job settings or user settings. If True, e-mail notifications and output attachments are sent according to job settings. If False, e-mail notifications and output attachments are sent according to user settings. The default value is False. If False, the following elements are ignored:

- AttachReportInEmail
- SendEmailForSuccess
- SendEmailForFailure
- SendSuccessNotice
- SendFailureNotice

**EmailFormat**

**String**. Optional. Specifies the output format of the report attached to the e-mail notification. Valid formats are:

- Excel (XLS)
- Excel (XLSX)
- PDF
- PostScript (PS)
- Word (DOC)
- Word (DOCX)
- PowerPoint (PPT)
- PowerPoint (PPTX)

**RecordSuccessStatus**

**Boolean**. Optional. Specifies whether to record job success notices.

**RecordFailureStatus**

**Boolean**. Optional. Specifies whether to record job failure notices.

**KeepOutputFile**

**Boolean**. Optional. Specifies whether the generated output file remains in the volume if the generation request succeeds but the printing request fails. Used if the job is to be generated and printed. If True, the output file remains in the volume. If False, the output file is deleted if the printing request fails. The default value is False.

**ConversionOptions**

Obsolete since iHub Release 2.

**DataACL**

**ArrayOfString**. Optional. Specifies the access control list (ACL) restricting data privileges.

## JobNotice

A complex data type that describes a job notice.

**Elements****JobId**

**String**. Optional. The job ID.

**JobName**

**String**. Optional. The name of the job.

## JobNoticeCondition

### **Headline**

[String](#). Optional. The job headline.

### **JobState**

[String](#). Optional. The state of the job. Valid values are:

- Succeeded
- Failed
- Cancelled

### **CompletionTime**

[DateTime](#). Optional. The time the job is completed.

### **ActualOutputFileName**

[String](#). Optional. The output file name that the BIRT iHub generated.

### **OutputFileName**

[String](#). Optional. The output file name.

### **OutputFileVersionName**

[String](#). Optional. The output file version name.

### **ActualOutputFileSize**

[Long](#). Optional. The size of the output file.

### **ActualOutputFileId**

[String](#). Optional. The output file ID that the BIRT iHub generated.

### **NotifiedUserId**

[String](#). Optional. The ID of the user who received the notice.

### **NotifiedUserName**

[String](#). Optional. The name of the user who received the notice.

### **NotifiedChannelId**

Obsolete since iHub Release 3.

### **NotifiedChannelName**

Obsolete since iHub Release 3.

### **OutputFileVersion**

[Long](#). Optional. The output file version number.

---

## JobNoticeCondition

A complex data type that represents the field on which to perform a search and the condition to match.

**Elements** **Field**

**String.** Optional. The field on which to perform a search. Valid values are:

- JobId
- JobName
- OutputFileName
- JobState
- HeadLine
- CompletionTime

**Match**

**String.** The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

05\ -25\ -04Report .rptdocument  
05 [-] 25 [-] 04Report .rptdocument

## JobNoticeField

A simple data type that represents the fields on which a search can be performed.

**Element** **String.** One of the following values:

**JobId**

The job ID.

**JobName**

The name of the job.

**OutputFileName**

The output file name.

**JobState**

The state of the job.

**HeadLine**

The job headline.

**CompletionTime**

The time the job is completed.

## JobNoticeSearch

A complex data type that represents the job notice search.

**Elements** **Condition**

[JobNoticeCondition](#). Optional. The search condition. Specify one of this parameter or ConditionArray.

**ConditionArray**

[ArrayOfJobNoticeCondition](#). Optional. An array of search conditions. Specify one of this parameter or Condition.

**NotifiedUserId**

[String](#). Optional. The ID of the user who received the notice. Specify either this parameter or NotifiedUserName.

**NotifiedUserName**

[String](#). Optional. The name of the user who received the notice. Specify either this parameter or NotifiedUserId.

**NotifiedChannelId**

Obsolete since iHub Release 3.

**NotifiedChannelName**

Obsolete since iHub Release 3.

**FetchSize**

[Int](#). Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

[Boolean](#). Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

[Int](#). Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

[String](#). Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specify the same search criteria as in the original call.

---

## JobPrinterOptions

A complex data type that describes the job printer options.

**Elements** **PrinterName**

[String](#). The printer name.

**Orientation**

**String.** Optional. The paper orientation.

**PageSize**

**String.** Optional. The page size.

**Scale**

**Long.** Optional. The scaling factor.

**Resolution**

**String.** Optional. The resolution.

**NumberOfCopies**

**Long.** Optional. The number of copies.

**CollationOption**

**Boolean.** Optional. Specifies whether the printer's collation property is set.

**PaperTray**

**Boolean.** Optional. Specifies whether the printer's paper tray option is set.

**Duplex**

**String.** Optional. The value of the printer's duplex property.

**IsColor**

**Boolean.** Optional. Specifies whether the printer can print in color.

**PaperLength**

**Long.** Optional. The paper length.

**PaperWidth**

**Long.** Optional. The paper width.

**PageRange**

**String.** Optional. The page range. Cannot exceed 20 characters.

**FormName**

**String.** Optional. The form name.

**PrintToFile**

**String.** Optional. The setting of the print to file property. Cannot exceed 256 characters.

## JobProperties

A complex data type that specifies the general job attributes, such as input document file name, output file name, and job execution status.

**Elements** **JobId**

**String.** Optional. The job ID.

**JobName**

**String**. Optional. The name of the job.

**Priority**

**Long**. Optional. The job priority.

**ResourceGroup**

**String**. Optional. The name of the resource group to which a job is assigned, if any.

**Owner**

**String**. Optional. The owner of the job.

**JobType**

**String**. Optional. The type of job. Valid values are:

- RunReport
- PrintReport
- RunAndPrintReport
- ConvertReport

**State**

**String**. Optional. The state of the job. Valid values are:

- Scheduled
- Pending
- Waiting
- Running
- Succeeded
- Failed
- Cancelled
- Expired

**InputFileDialog**

**String**. Optional. The input file ID.

**InputFileName**

**String**. Optional. The input file full name and version number.

**RunLatestVersion**

**Boolean**. Optional. Specifies whether to run the latest version of the file.

**ParameterFileDialog**

**String**. Optional. The parameter file ID. Exists only if the parameter file is specified.

**ParameterFileName**

[String](#). Optional. The parameter file name. Exists only if the parameter file is specified.

**ActualOutputFileId**

[String](#). Optional. The output file ID that the BIRT iHub generated.

**ActualOutputFileName**

[String](#). Optional. The output file name that the BIRT iHub generated. This might be different from the RequestedOutputFileName.

**RequestedOutputFileName**

[String](#). Optional. The requested name for the output file.

**OutputFileVersionName**

[String](#). Optional. The output file version name.

**SubmissionTime**

[DateTime](#). Optional. The time the job was submitted.

**CompletionTime**

[DateTime](#). Optional. The time the job is completed.

**PageCount**

[Long](#). Optional. The number of pages.

**OutputFileSize**

[Long](#). Optional. The size of the output file.

**RoutedToNode**

[String](#). Optional. The node to which the job is routed.

**DurationSeconds**

[Long](#). Optional. The job duration.

**StartTime**

[DateTime](#). Optional. The start time.

**NextStartTime**

[DateTime](#). Optional. The next time the job is scheduled to run. Applies only to scheduled jobs.

**RequestedHeadline**

[String](#). Optional. The headline for the job.

**ActualHeadline**

[String](#). Optional. The headline that the BIRT iHub generated.

**NotifyCount**

[String](#). Optional. The number of notifications sent about the job.

## JobSchedule

### EventName

[String](#). Optional. The name of the job event.

### EventType

[EventType](#). Optional. The job event type.

### EventStatus

[String](#). Optional. The job event status.

### EventParameter

[String](#). Optional. The parameter for the job event.

---

## JobSchedule

A complex data type that represents details about a job schedule.

### Elements

#### TimeZoneName

[String](#). Optional. The time zone. Cannot exceed 32 characters.

#### ScheduleDetails

[ArrayOfJobScheduleDetail](#). The schedule details.

---

## JobScheduleCondition

A complex data type that represents the field on which to perform a search and the condition to match.

### Elements

#### Field

[JobScheduleField](#). Field on which to perform a search.

#### Match

[String](#). The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([ ]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

05\ -25\ -04Report . rptdocument

05 [ - ] 25 [ - ] 04Report . rptdocument

---

## JobScheduleDetail

A complex data type that specifies a schedule for running a job.

**Elements** **ScheduleType**

[String](#). The type of schedule. Valid values are:

- AbsoluteDate
- Daily
- Weekly
- Monthly

**ScheduleStartDate**

[String](#). Optional. The date on which to start the schedule. The date is a standard XML String data type in the format YYYY-MM-DDThh:mm:ss-sss. Milliseconds, if specified, are ignored.

**ScheduleEndDate**

[String](#). Optional. The date on which to end the schedule. The date is a standard XML String data type using the format YYYY-MM-DDThh:mm:ss-sss. Milliseconds, if specified, are ignored.

**DatesExcluded**

[ArrayOfDate](#). Optional. An array of dates to exclude from the schedule.

**AbsoluteDate**

[AbsoluteDate](#). Optional. This value applies if ScheduleType is AbsoluteDate.

**Daily**

[Daily](#). Optional. This value applies if ScheduleType is Daily.

**Weekly**

[Weekly](#). Optional. This value applies if ScheduleType is Weekly.

**Monthly**

[Monthly](#). Optional. This value applies if ScheduleType is Monthly.

## JobScheduleField

A simple data type describing job schedule fields upon which a search can be performed.

**Element** [String](#). One of the following values:

**JobName**

The name of the job.

**Owner**

The owner of the job.

## JobScheduleSearch

### **JobType**

The type of job. Valid values are:

- RunReport
- PrintReport
- RunAndPrintReport
- ConvertReport

### **Priority**

The job priority.

### **NextStartTime**

The next time the job is scheduled to run. Applies only to scheduled jobs.

### **State**

The state of the job. Valid values are:

- Scheduled
- Pending
- Running
- Succeeded
- Failed
- Cancelled
- Expired

### **ParameterFileId**

The parameter file ID. Exists only if the parameter file is specified.

---

## JobScheduleSearch

A complex data type that represents a job schedule search.

### **Elements** **Condition**

[JobScheduleCondition](#). Optional. The search condition. Specify this value or ConditionArray.

### **ConditionArray**

[ArrayOfJobScheduleCondition](#). Optional. The array of search conditions. Specify this value or Condition.

### **RequestedOutputFileName**

[String](#). Optional. The output file name.

**InputFileName**

[String](#). Optional. The input file name.

**InputFileId**

[String](#). Optional. The input file ID.

**EventType**

[EventType](#). Optional. The event type of the job.

**NotifiedUserId**

[String](#). Optional. The ID of the user to notify. Specify either this value or NotifiedUserName.

**NotifiedUserName**

[String](#). Optional. The name of the user to notify. Specify either this value or NotifiedUserId.

**NotifiedChannelId**

Obsolete since iHub Release 3.

**NotifiedChannelName**

Obsolete since iHub Release 3.

**FetchSize**

[Int](#). Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

[Boolean](#). Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

[Int](#). Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

[String](#). Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specifies the same search criteria as in the original call.

---

## JobSearch

A complex data type that represents a job search.

**Elements****Condition**

[JobCondition](#). Optional. The search condition. Specify this value or ConditionArray.

**ConditionArray**

**ArrayOfJobCondition**. Optional. The array of search conditions. Specify this value or Condition.

**Owner**

**String**. Optional. The owner of the job.

**ActualOutputFileName**

**String**. Optional. The output file name that the BIRT iHub generated.

**ActualOutputFileId**

**String**. Optional. The output file ID that the BIRT iHub generated.

**RequestedOutputFileName**

**String**. Optional. The output file requested name.

**InputFileName**

**String**. Optional. The input file name.

**InputFieldId**

**String**. Optional. The input file ID.

**NotifiedUserId**

**String**. Optional. The ID of the user who received notification. Specify either this value or NotifiedUserName.

**NotifiedUserName**

**String**. Optional. The name of the user who received notification. Specify either this value or NotifiedUserId.

**NotifiedChannelId**

Obsolete since iHub Release 3.

**NotifiedChannelName**

Obsolete since iHub Release 3.

**FetchSize**

**Int**. Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

**Boolean**. Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

**Int**. Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

**String**. Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specifies the same search criteria as in the original call.

---

## LicenseOption

A complex data type that represents a license option.

**Elements****Name**

[String](#). Optional. The name of the license option.

**Description**

[String](#). Optional. The description of the option.

**ShortDescription**

[String](#). Optional. A shorter description of the option.

**Value**

[String](#). Optional. The value of the option.

---

## Long

Long is a standard XML long data type, which supports a 64-bit value, ranging from -9223372036854775808 to 9223372036854775807.

---

## MDSInfo

A complex data type that describes a Message Distribution service (MDS).

**Elements****ServerName**

[String](#). The server name.

**MDSIPAddress**

[String](#). The IP address of the MDS.

**MDSPortNumber**

[Int](#). The port number the MDS uses.

**MDSSSLPortNumber**

[Int](#). Optional. The SSL port number the MDS uses.

**ServerState**

[ServerState](#). Optional. The server state.

---

## Monthly

A complex data type that describes monthly job scheduling.

## Monthly

### Elements **FrequencyInMonths**

**Long**. The amount of times a job is to be run, in months.

### OnDay

**Int**. Optional. The day of the month on which to run the job.

### OnWeekDay

**Int**. Optional. The day of the month, excluding weekends, on which to run the job.

### RunOn

Optional. A complex type containing two **String** values. The first string, WeekDay, specifies the day of the week on which to run the job. Valid values for WeekDay are:

- Mon
- Tue
- Wed
- Thu
- Fri
- Sat
- Sun

The second string, Occurrence, specifies which occurrence in the month of that day to run the job. Valid values for Occurrence are:

- First
- Second
- Third
- Fourth
- Last

For example, the strings, Tue and Third, specify the third Tuesday of the month.

### OnceADay

**String**. Optional. Specifies the time the job is to be run.

### Repeat

**Repeat**. Optional. Specifies how often the schedule is to be repeated.

---

## MoveFile

Moves files or folders to a new location. To move a single file or folder, specify Name or Id. To move a list of files or folders, specify NameList or IdList. To move files or folders that match the specified conditions, specify Search.

**Elements****Target**

**String**. The new location for the file or folder. The following rules apply:

- If Target is a file, the operation fails if the source contains a folder.
- If the source is a folder and a folder with the same name exists in the target location, the operation fails.
- If the source is a file and a file with an identical name exists in the target location, the existing file in the target location is versioned or replaced, depending on the setting of the ReplaceExisting tag. If the existing file has any dependencies, the file is not replaced regardless of the ReplaceExisting setting.
- If Target is a folder that does not exist, a folder is created.

**WorkingFolderName**

**String**. The name of the working folder of the file or folder to move. Specify either WorkingFolderName or WorkingFolderId.

**WorkingFolderId**

**String**. The ID of the working folder of the file or folder to move. Specify either WorkingFolderId or WorkingFolderName.

**Recursive**

**Boolean**. Specifies whether to search subfolders. If True, the search includes subfolders. The default value is False.

**Search**

**FileSearch**. The search condition that specifies which folders or files to move.

**IdList**

**ArrayOfString**. The list of file or folder IDs to move. Specify either IdList or NameList.

**NameList**

**ArrayOfString**. The list of file or folder names to move. Specify either NameList or IdList.

**Id**

**String**. The ID of the single file or folder to move. Specify either Id or Name.

**Name**

**String**. The name of the single file or folder to move. Specify either Name or Id.

## NameValuePair

### ReplaceExisting

**Boolean.** If True, the existing file, if one exists, is replaced. If the existing file has any dependencies, it is not replaced. If False or the existing file has any dependencies, the file is versioned. The default value is True.

### MaxVersions

**Long.** The maximum number of versions to create. MaxVersions applies only if a file is moved. If a folder is moved, MaxVersions is ignored.

### LatestVersionOnly

**Boolean.** Specifies whether all versions or only the latest version of the file is moved. Used only when a Search tag is specified. If True, only the latest version of the file is moved. The default value is False.

---

## NameValuePair

A complex data type that represents a named piece of data and its value.

**Elements** **Name**

**String.** The name of the data.

**Value**

**String.** The value of the data. Value supports the null value.

---

## NewFile

A complex data type that describes a file.

**Elements** **Name**

**String.** The name of the file.

**VersionName**

**String.** Optional. The version name of the file.

**ReplaceExisting**

**Boolean.** Optional. Deprecated. Use [Versioning](#) instead of ReplaceExisting. Specifies whether to overwrite the latest existing version when uploading a file. If the existing file has any dependencies, BIRT iHub does not overwrite the file and creates a new version, regardless of the ReplaceExisting setting.

**Versioning**

**VersioningOption.** Optional. Specifies what to do with the latest existing version when uploading a file. Valid values are:

- CreateNewVersion

Always creates a new version. This is the default value.

- ReplaceLatestIfNoDependents  
Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub creates a new version instead of replacing the existing version.
- ReplaceLatestDropDependency  
Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub drops the dependency.
- ReplaceLatestMigrateDependency  
Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub moves the dependency to the new version.

**MaxVersions**

**Long**. Optional. The maximum number of versions to keep in the volume.

**Description**

**String**. Optional. The description of the file.

**ArchiveRule**

**ArchiveRule**. Optional. The autoarchive rules for the file.

**ACL**

**ArrayOfPermission**. Optional. The access rights to the file.

**AccessType**

**FileAccess**. Optional. The file's access type. Valid values are:

- Private  
Only the owner of the file and an administrator can access the file.
- Shared  
All users and user groups specified in the access control list (ACL) for the file can access the file.

## ObjectIdentifier

A complex data type that describes object identifiers.

**Elements** **Id**

**String**. Optional. The ID of the object.

**Name**

**String**. Optional. The name of the object.

**Type**

**String**. Optional. The object type.

## OpenServerOptions

### Version

[Long](#). Optional. The object version number.

---

## OpenServerOptions

Obsolete since iHub Release 2.

---

## PagelIdentifier

A complex data type that describes page numbers.

**Elements** **Range**

[String](#). Optional. A page range.

**PageNum**

[Long](#). Optional. A page number.

**ViewMode**

[Int](#). Optional. The page viewing mode.

---

## ParameterDefinition

A complex data type that defines a report parameter.

**Elements** **Group**

[String](#). Optional. The parameter group.

**Name**

[String](#). The parameter name.

**Position**

[Int](#). Optional. The index location of the parameter in the information object (.iob) or data source map (.sma) file. The index is 1-based. For a regular parameter, do not specify a value or specify 0.

**DataType**

[String](#). Optional. The data type of the parameter. Valid values are:

- Currency
- Date
- Double
- Integer
- String

- Boolean
- Structure Obsolete since Actuate Release 11
- Table Obsolete since Actuate Release 11

**DefaultValue**

`String`. Optional. The default value of the parameter.

**IsRequired**

`Boolean`. Optional. Specifies whether the parameter is required.

**IsPassword**

`Boolean`. Optional. Specifies whether a password is required.

**IsHidden**

`Boolean`. Optional. Specifies whether the parameter is hidden.

**DisplayName**

`String`. Optional. The display name of the parameter.

**IsAdHoc**

`Boolean`. Optional. Specifies whether the parameter is ad hoc.

**ControlType**

`String`. Optional. The type of control used to represent the parameter. Valid values are:

- AutoSuggest  
An autosuggest control
- ControlRadioButton  
A radio button
- ControlList  
A drop-down list
- ControlListAllowNew  
A text box
- ControlCheckBox  
A check box
- FilterSimple  
A simple filter
- FilterAdvanced  
An advanced filter

**SelectValueList**

`ArrayOfString`. Optional. The list of available parameter values.

## ParameterDefinition

### **OperatorList**

[ArrayOfString](#). Optional. Contains the operators used with ad hoc parameters.

### **ColumnName**

Obsolete since iHub Release 2.

### **ColumnType**

Obsolete since iHub Release 2.

### **RecordDefinition**

Obsolete since iHub Release 2.

### **DefaultTableValues**

Obsolete since iHub Release 2.

### **DataSourceType**

Obsolete since iHub Release 2.

### **CascadingParentName**

[String](#). Optional. The cascading parent name for this parameter definition.

### **SelectNameValuePairList**

[ArrayNameValuePair](#). Optional. The list of names of available parameters.

### **HelpText**

[String](#). Optional. The text to display when the user holds the cursor over a parameter. For example, a value of a data column.

### **IsViewParameter**

[Boolean](#). Optional. Whether the parameter is a view parameter. The default value is False.

### **IsDynamicSelectionList**

[Boolean](#). Optional. Flag indicating whether the selection list is dynamic or static.

### **AutoSuggestThreshold**

[Int](#). Optional. The minimum number of characters to be entered before the AutoSuggest selection list is displayed.

### **StartExpanded**

[Boolean](#). Optional.

### **GroupPromptText**

[String](#). Optional.

### **DefaultValueIsNull**

[Boolean](#). Optional. Flag indicating that the default value is null.

---

## ParameterValue

A complex data type that defines the value of a report parameter.

<b>Elements</b>	
<b>Group</b>	<b>String</b> . Optional. The parameter group.
<b>Name</b>	<b>String</b> . The parameter name.
<b>DisplayName</b>	<b>String</b> . Optional. The label or display name for the parameter that appears in the user interface.
<b>Position</b>	<b>Int</b> . Optional. The index location of the parameter in the information object (.iob) or data source map (.sma) file. The index is 1-based. For a regular parameter, do not specify a value or specify 0.
<b>Value</b>	<b>String</b> . Optional. The parameter value.
<b>ValueIsNull</b>	<b>Boolean</b> . Optional. A flag indicating a null value.
<b>PromptParameter</b>	<b>Boolean</b> . Optional. Allows the user to select the parameter.
<b>TableValue</b>	Obsolete since iHub Release 2.
<b>DataSourceType</b>	Obsolete since iHub Release 2.
<b>IsViewParameter</b>	<b>Boolean</b> . Optional. Specifies whether the parameter is a view parameter.

---

## PendingSyncJob

A complex data type that describes a job in the queue waiting for Factory processing.

<b>Elements</b>	<b>ConnectionHandle</b>
	<b>Base64Binary</b> . An optional element that supports keeping a connection open to view a persistent report. If ConnectionHandle is present in the SOAP header, the system routes subsequent viewing requests to the same View service that returned the ConnectionHandle. If present, BIRT iHub System ignores the value of TargetVolume.

## Permission

### **ObjectId**

**String.** The ID of the synchronous report for which to retrieve information.

### **IsTransient**

**Boolean.** True if the synchronous report is transient, False if the synchronous report is persistent.

### **Volume**

**String.** The volume on which the job originated.

### **ServerName**

**String.** Optional. The node on which the job is pending.

### **Owner**

**String.** Optional. The name of the user who submitted the job.

### **ExecutableFileName**

**String.** Optional. The fully qualified name of the report executable file.

### **ExecutableVersionNumber**

**Long.** Optional. The fully qualified version number of the report executable file.

### **ExecutableVersionName**

**String.** Optional. The fully qualified version name of the report executable file.

### **SubmissionTime**

**Date Time.** Optional. The time at which the job was submitted to the server.

### **PendingTime**

**Long.** Optional. The number of seconds elapsed since the job entered the queue.

### **QueueTimeout**

**Long.** Optional. The number of seconds remaining before the job is deleted from the queue.

### **QueuePosition**

**Long.** Optional. The job's position in the queue.

---

## Permission

A complex data type that describes a user or user group's privileges.

### **Elements** **RoleName**

Deprecated since iHub Release 3. Replaced by UserGroupName.

### **UserGroupName**

**String.** Optional. The user group name. Specify this value or UserName.

### **UserName**

**String.** Optional. The user name. Specify this value or UserGroupName.

**RoleId**

Deprecated since iHub Release 3. Replaced by UserGroupId.

**UserGroupId**

[String](#). Optional. The user group ID. Specify this value or UserId.

**UserId**

[String](#). Optional. The user ID. Specify this value or UserGroupId.

**AccessRight**

[String](#). The privileges the user or user group has on an object. One or more of the following characters representing a privilege:

- D—Delete
- E—Execute
- G—Grant
- V—Visible
- S—Secured Read
- R—Read
- W—Write

## Printer

A complex data type that describes a printer.

**Elements****Name**

[String](#). Optional. The name of the printer. Cannot exceed 50 characters.

**Manufacturer**

[String](#). Optional. The manufacturer of the printer.

**Model**

[String](#). Optional. The model of the printer.

**Location**

[String](#). Optional. The location of the printer.

**Description**

[String](#). Optional. The description of the printer.

**SupportOrientation**

[Boolean](#). Optional. Specifies whether the printer supports setting paper orientation.

**Orientation**

[String](#). Optional. The setting of the printer's orientation property.

**OrientationOptions**

`ArrayOfString`. Optional. The setting of the printer's orientation options.

**SupportPageSize**

`Boolean`. Optional. Specifies whether page size can be set on the printer.

**PageSize**

`String`. Optional. The setting of the printer's page size property.

**PageSizeOptions**

`ArrayOfString`. Optional. The page sizes the printer supports.

**SupportScale**

`Boolean`. Optional. Specifies whether the printer supports setting the scaling factor.

**Scale**

`Long`. Optional. The setting of the printer's scaling factor.

**ScaleOptions**

`ArrayOfInt`. Optional. The setting of the printer's scaling options.

**SupportResolution**

`Boolean`. Optional. Specifies whether the printer supports setting the resolution.

**Resolution**

`String`. Optional. The setting of the printer's resolution property.

**ResolutionOptions**

The setting of the printer's resolution options.

**SupportNumberOfCopies**

`Boolean`. Optional. Specifies whether the printer supports setting the number of copies.

**NumberOfCopies**

`Long`. Optional. The setting of the number of copies property.

**SupportCollation**

`Boolean`. Optional. Specifies whether the printer supports setting the collation.

**Collation**

`Boolean`. Optional. The setting of the printer's collation property.

**SupportPaperTray**

`Boolean`. Optional. Specifies whether the printer supports setting the paper tray.

**PaperTray**

`String`. Optional. The setting of the printer's paper tray property.

**PaperTrayOptions**

`ArrayOfString`. Optional. The setting of the printer's paper tray options.

**SupportDuplex**

**Boolean.** Optional. Specifies whether the printer supports duplex printing.

**Duplex**

**String.** Optional. The setting of the printer's duplex property.

**DuplexOptions**

**ArrayOfString.** Optional. The setting of the printer's duplex options.

**SupportColorMode**

**Boolean.** Optional. Specifies whether the printer supports printing in color.

**ColorMode**

**String.** Optional. The setting of the printer's color mode property.

**ColorModeOptions**

**ArrayOfString.** Optional. The setting of printer's color mode options.

**Status**

**String.** Optional. Indicates printer's availability.

## PrinterOptions

A complex data type that describes printer options.

**Elements** **PrinterName**

**String.** The name of the printer.

**Orientation**

**String.** Optional. The paper orientation.

**PageSize**

**String.** Optional. The page size.

**Scale**

**Long.** Optional. The scaling factor.

**Resolution**

**String.** Optional. The resolution.

**NumberOfCopies**

**Long.** Optional. The number of copies.

**CollationOption**

**Boolean.** Optional. Turns collation on and off.

**PaperTray**

**String.** Optional. The paper tray.

## PrivilegeFilter

### Duplex

**String.** Optional. Sets duplex printing.

### IsColor

**Boolean.** Optional. Specifies whether the printer can print in color.

### IsDefaultPrinter

**Boolean.** Optional. Specifies whether the printer is the default printer.

---

## PrivilegeFilter

A complex data type that represents a privilege filter. Use PrivilegeFilter to retrieve only the data accessible to user groups or users with the specified privileges and to determine whether a user or user group has the specified privileges on an item.

### Elements

#### GrantedUserName

**String.** Optional. The name of the user privileges to retrieve. Specify one of this value, GrantedUserId, GrantedUserGroupId, or GrantedUserGroupName.

#### GrantedUserId

**String.** Optional. The ID of the user privileges to retrieve. Specify one of this value, GrantedUserName, GrantedUserGroupId, or GrantedUserGroupName.

#### GrantedRoleName

Deprecated since iHub Release 3. Replaced by GrantedUserGroupName.

#### GrantedRoleId

Deprecated since iHub Release 3. Replaced by GrantedUserGroupId.

#### GrantedUserGroupName

**String.** Optional. The name of the user group whose privileges to retrieve. Specify one of this value, GrantedUserName, GrantedUserId, or GrantedUserGroupId.

#### GrantedUserGroupId

**String.** Optional. The ID of the user group privileges to retrieve. Specify one of this value, GrantedUserName, GrantedUserId, or GrantedUserGroupName.

#### AccessRights

**String.** The privileges.

---

## PropertyValue

A complex data type that specifies a name-value pair.

### Elements

#### Name

**String.** The name of the property.

**Value**

[String](#). Optional. The value of the property.

---

## Range

Obsolete since Actuate Release 11.

---

## Repeat

A complex data type that describes how often a job run is to be repeated.

**Elements** **StartTime**

[String](#). The time that the job is to start repeatedly running.

**EndTime**

[String](#). The time that the job is to no longer run.

**IntervalInSeconds**

[Long](#). The time to wait between job runs.

---

## Record

Obsolete since Actuate Release 11.

---

## ReportParameterType

A simple data type that describes parameter types.

**Element** [String](#). One of the following values:**Execution**

An execution parameter

**View**

A view parameter.

**All**

An all parameter type.

---

## ResourceGroup

A complex data type that describes a resource group.

**Elements****Name**

**String**. The name of the resource group.

**Disabled**

**Boolean**. Optional. Specifies whether the resource group can run jobs. If True, the resource group does not run jobs. The default value is False.

**Description**

**String**. Optional. The description of the resource group.

**Type**

**String**. Optional. The type of jobs the resource group runs. Valid values are:

- Sync  
The resource group runs synchronous jobs.
- Async  
The resource group runs asynchronous jobs.

**ReportType**

**String**. Optional. The type of report the resource group creates.

**Volume**

**String**. Optional. The name of a volume to which to assign the resource group. Valid values are:

- An empty string  
Assigns all volumes on the BIRT iHub
- A volume name  
Assigns the specified volume

**MinPriority**

**Long**. Optional. Applies only to an asynchronous resource group. Specifies the minimum priority for the resource group. Valid values are 0–1,000, where 1,000 is the highest priority. MinPriority must be less than MaxPriority. The default value is 0.

**MaxPriority**

**Long**. Optional. Applies only to an asynchronous resource group. Specifies the maximum priority for the resource group. Valid values are 0–1,000, where 1,000 is the highest priority. MaxPriority must be more than MinPriority. The default value is 1,000.

**Reserved**

**Boolean**. Optional. Applies only to a synchronous resource group. True reserves the resource group to run only the jobs assigned to it. Use the TargetResourceGroup element in the SOAP header of an ExecuteReport request to assign a job.

**StartArguments**

**String**. Optional. The starting arguments for the resource group.

**WorkUnitType**

**String**. Optional. The license option type. An aggregate licensing model that defines iHub System features in terms of work units.

## ResourceGroupSettings

A complex data type that describes the settings of a resource group.

**Elements** **TemplateName**

**String**. The name of the BIRT iHub template on which the resource group runs.

**Activate**

**Boolean**. Optional. Specifies whether the BIRT iHub is a member of the resource group. If True, the BIRT iHub is a member of the resource group. The default value is False.

**MaxFactory**

**Int**. Optional. The maximum number of Factory processes available to the resource group.

**MinFactory**

**Int**. Optional. The minimum number of Factory processes available to the resource group.

**FileTypes**

**ArrayOfString**. Optional. The file types the resource group can run.

**StartArguments**

**String**. Optional. The starting arguments for the resource group.

## ResultSetSchema

A complex data type that describes the result set schema.

**Elements** **ResultSetName**

**String**. Optional. Name of the result set.

## RetryOptions

### **ResultSetDisplayName**

**String**. Optional. The display name of the result set. If not specified, the value of the Name element is used. If the query is performed on an information object (.iob) or data source map (.sma) file, DisplayName is used as the group label.

### **ArrayOfColumnSchema**

**ArrayOfColumnSchema**. Optional. An array of ColumnSchema objects.

---

## RetryOptions

A complex data type that describes how to retry report generation or printing jobs that have failed.

**Elements** **RetryOption**

**RetryOptionType**. The retry options.

**MaxRetryCount**

**Long**. Optional. The maximum number of retry attempts.

**RetryInterval**

**Long**. Optional. The interval between retry attempts. Measured in seconds.

---

## RetryOptionType

A simple data type that describes a retry option for failed jobs.

**Element** **String**. One of the following values:

**Disabled**

Specifies not to retry the job.

**VolumeDefault**

Specifies using the default retry options for the volume.

**Specified**

Specifies using the retry option defined on the job itself.

---

## Role

Deprecated since iHub Release 3. Replaced by [UserGroup](#).

---

## RoleCondition

Deprecated since iHub Release 3. Replaced by [UserGroupCondition](#).

---

## RoleField

Deprecated since iHub Release 3. Replaced by [UserGroupField](#).

---

## RoleSearch

Deprecated since iHub Release 3. Replaced by [UserGroupSearch](#).

---

## RunningJob

A complex data type that describes a job the Factory is currently processing.

**Elements** **IsSyncJob**

[Boolean](#). Specifies whether the job is synchronous. True if the job is synchronous, False if the job is asynchronous.

**ConnectionHandle**

[Base64Binary](#). Optional. An element that supports keeping a connection open to view a persistent report. If ConnectionHandle is present in the SOAP header, the system routes subsequent viewing requests to the same View service that returned the ConnectionHandle. If present, BIRT iHub System ignores the value of TargetVolume.

**ObjectId**

[String](#). Optional. The ID of the synchronous report for which to retrieve information.

**IsTransient**

[Boolean](#). Optional. Specifies whether the synchronous report is transient. True if the synchronous report is transient, False if the synchronous report is persistent.

**IsProgressive**

[Boolean](#). Optional. Specifies whether progressive viewing is enabled. True if progressive viewing is enabled.

**JobId**

[String](#). Optional. The ID of the asynchronous job.

## ScalarDataType

### Volume

**String.** Optional. The volume on which the job originated.

### ServerName

**String.** Optional. The node on which the job is running.

### Owner

**String.** Optional. The name of the user who submitted the job.

### ExecutableFileName

**String.** Optional. The fully qualified name of the executable file.

### ExecutableVersionNumber

**Long.** Optional. The version number of the executable file.

### ExecutableVersionName

**String.** Optional. The version name of the executable file.

### ResourceGroup

**String.** Optional. The resource group for the job.

### SubmissionTime

**Date Time.** Optional. The time at which the job was submitted to the server.

### StartTime

**Date Time.** Optional. The time at which job execution started.

### RunningTime

**Long.** Optional. The time elapsed since job execution started.

### ExecutionTimeout

**Long.** Optional. The number of seconds remaining before job execution times out. The number is always zero (infinite) for asynchronous reports.

### IsSyncFactory

**Boolean.** Optional. Specifies whether the Factory is running synchronous jobs. True if the Factory is running synchronous jobs, False if the Factory is running asynchronous jobs.

### FactoryPid

**Long.** Optional. The process ID of the Factory.

---

## ScalarDataType

A simple data type that specifies a scalar parameter.

**Element** **String.** One of the following values:

**Currency**

A Currency parameter.

**Date**

A Date parameter.

**DateOnly**

A DateOnly parameter.

**Time**

A Time parameter.

**Double**

A Double parameter.

**Integer**

An Integer parameter.

**String**

A String parameter.

**Boolean**

A Boolean parameter.

---

## SearchReportByIdList

Obsolete since iHub Release 2.

---

## SearchReportByIdNameList

Obsolete since iHub Release 2.

---

## SearchReportByNameList

Obsolete since iHub Release 2.

---

## SearchResultProperty

Obsolete since iHub Release 2.

## ServerInformation

A complex data type that describes a BIRT iHub.

**Elements** **ServerName**

[String](#). The name of the BIRT iHub.

**ServerStatusInformation**

[ServerStatusInformation](#). Optional. The status of the BIRT iHub. Valid values are:

- ServerState
- SystemType
- StatusErrorCode
- StatusErrorDescription

**ServiceList**

[ArrayOfService](#). The list of available services.

**OwnsVolume**

[Boolean](#). Optional. True if there are any volumes on the BIRT iHub, False otherwise.

**Description**

[String](#). Optional. The description of the BIRT iHub.

**ServerVersionInformation**

[ServerVersionInformation](#). The following information about the BIRT iHub version:

- ServerVersion
- ServerBuild
- OSVersion

**ChangesPending**

[String](#). Optional. Server configuration has changed and the BIRT iHub or the system must be restarted for the changes to take effect.

**NodeLockViolation**

[Boolean](#). Optional. Specifies whether a licensing node-lock violation exists. The default value is False.

**NodeLockViolationExpirationDate**

[String](#). Optional. The date on which the grace period for a node-lock violation expires and the node lock takes effect. Contact Actuate Licensing about a node-lock licensing problem.

**TemplateName**

**String.** The name of the BIRT iHub configuration template.

**ServerIPAddress**

**String.** Optional. The IP address or host name of the BIRT iHub.

**PmdPortNumber**

**Int.** Optional. The port where the Process Management Daemon (PMD) listens.

**LocalServer**

**Boolean.** Optional. Specifies whether the BIRT iHub is running on the local machine.

---

## ServerResourceGroupSetting

A complex data type that describes the settings of a resource group available to a BIRT iHub.

**Elements** **ResourceGroupName**

**String.** The name of the resource group.

**Activate**

**Boolean.** Optional. Specifies whether the BIRT iHub is a member of the resource group. If True, the BIRT iHub is a member of the resource group. The default value is False.

**Type**

**String.** Optional. The type of jobs the resource group runs. Valid values are:

- Sync  
The resource group runs synchronous jobs.
- Async  
The resource group runs asynchronous jobs.

**MaxFactory**

**Int.** Optional. The maximum number of Factory processes available to the resource group.

**MinFactory**

**Int.** Optional. The minimum number of Factory processes available to the resource group.

**FileTypes**

**ArrayOfString.** Optional. The file types the resource group can run.

## ServerState

### StartArguments

[String](#). Optional. The list of arguments used when starting a resource group process. For example, the Default Java Async resource group uses the following arguments:

```
-Xmx256M -Djava.awt.headless=true
-Djava.protocol.handler.pkgs=com.actuate.javaserver.protocol
com.actuate.javaserver.Server
```

---

## ServerState

A simple data type that describes the state of Actuate iHub.

**Element** [String](#). One of the following values:

### OFFLINE

The server is offline.

### STARTING

The server is starting.

### ONLINE

The server is online.

### STOPPING

The server is stopping.

### FAILED

The server failed.

---

## ServerStatusInformation

A complex data type that describes the status of an Actuate iHub.

**Elements** [ServerState](#)

[ServerState](#). The state of the Actuate iHub. Valid values are:

- OFFLINE
- STARTING
- ONLINE
- STOPPING
- FAILED

### SystemType

[SystemType](#). The type of Actuate iHub, cluster or stand-alone.

**ServerStateErrorCode**

**Long**. Optional. The code of the error if status is Failed.

**ServerStateErrorDescription**

**String**. Optional. The description of the error if status is Failed.

---

## ServerVersionInformation

A complex data type that describes the version of an Actuate iHub.

**Elements** **ServerVersion**

**String**. The version of the Actuate iHub.

**ServerBuild**

**String**. The build number of the Actuate iHub.

**OSVersion**

**String**. The version of the operating system.

---

## Service

A simple data type that represents a service.

**Element** **String**. One of the following values:**Request**

A Message Distribution service (MDS).

**Viewing**

A View service.

**Generation**

A Factory service.

**Caching**

A Caching service.

**Integration**

An Integration service.

---

## Short

Short is a standard XML short data type, which is a signed 16-bit integer, ranging in value from -32,768 to 32,767.

## SortColumn

Obsolete since iHub Release 2.

---

## Stream

A complex data type that represents a streamed image.

**Elements** **Name**

The name of the image.

**EmbeddedProperty**

Specifies whether the image is embedded.

---

## String

String is a standard XML String data type that represents character data.

---

## SystemType

A simple data type that describes the type of BIRT iHub System.

**Element** [String](#). One of the following values:

**Cluster**

A cluster system.

**Standalone**

A stand-alone system.

---

## SupportedQueryFeatures

Obsolete since iHub Release 2.

---

## Time

Time is a standard XML Time data type that displays time in the format hh:mm:ss.sss with an optional time zone indicator. For example, 13:20:00-05:00

indicates 1:20 p.m. Eastern Standard Time, which is five hours behind Coordinated Universal Time (UTC).

---

## Transaction

A packaging mechanism for Administrate operations. If a failure occurs anywhere in a transaction, all operations in the transaction fail.

**Elements** [TransactionOperation](#)

The transaction operation.

---

## TransactionOperation

Controls the ability to create, delete, update, copy, and move items within a volume. A TransactionOperation represents a single unit of work within a Transaction. Only a volume administrator or a user in the Administrator user group uses these operations.

**Elements** One or more of the following elements:

[CreateUser](#)

Creates a user in the volume.

[DeleteUser](#)

Deletes one or more users.

[UndeleteUser](#)

Undeletes one or more users.

[UpdateUser](#)

Updates user properties in the volume.

[CreateGroup](#)

Obsolete since iHub Release 3.

[DeleteGroup](#)

Obsolete since iHub Release 3.

[UpdateGroup](#)

Obsolete since iHub Release 3.

[CreateChannel](#)

Obsolete since iHub Release 3.

[DeleteChannel](#)

Obsolete since iHub Release 3.

**[UpdateChannel](#)**

Obsolete since iHub Release 3.

**[CreateRole](#)**

Deprecated since iHub Release 3. Replaced by [CreateUserGroup](#).

**[DeleteRole](#)**

Deprecated since iHub Release 3. Replaced by [DeleteUserGroup](#).

**[UpdateRole](#)**

Deprecated since iHub Release 3. Replaced by [UpdateUserGroup](#).

**[CreateUserGroup](#)**

Creates a user group in a volume.

**[DeleteUserGroup](#)**

Deletes one or more user groups.

**[UpdateUserGroup](#)**

Updates user group properties in the volume.

**[CreateFileType](#)**

Creates a new file type in the volume.

**[DeleteFileType](#)**

Deletes a file type from the volume.

**[UpdateFileType](#)**

Updates file type properties in the volume.

**[CreateFolder](#)**

Creates a folder in a volume.

**[DeleteFile](#)**

Deletes files or folders from the volume.

**[MoveFile](#)**

Moves a file or folder from the working directory to a specified target directory in the volume.

**[CopyFile](#)**

Copies a file or folder in the working directory to a specified target directory.

**[UpdateFile](#)**

Updates file or folder properties in the volume.

**[DeleteJob](#)**

Deletes one or more jobs.

**[DeleteJobNotices](#)**

Deletes one or more job notices.

**DeleteJobSchedule**

Deletes a job schedule.

**UpdateJobSchedule**

Updates a job schedule.

**UpdateVolumeProperties**

Updates the properties of a specific volume.

**UpdateOpenSecurityCache**

Flushes the volume's open security cache and retrieves new data from an external security source.

---

## TypeName

A simple data type that describes names of data types.

**Element** [String](#). One of the following values:

**int**

An integer value.

**sht**

A short value.

**dbl**

A double value.

**dbn**

An Actuate double value.

**cur**

A currency value.

**dtm**

An date-and-time value.

**str**

A string value.

**bln**

A Boolean value.

**nll**

The null value.

## UndeleteUser

Undeletes users, reversing a DeleteUser operation within the unit of work of an AdminOperation. To undelete a single user, specify Id. To delete several users, specify IdList.

**Elements**

**IdList**

[ArrayOfString](#). The list of user IDs to undelete.

**Id**

[String](#). The ID of the single user to undelete.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified user does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

---

## UpdateChannel

Obsolete since iHub Release 3.

---

## UpdateChannelOperation

Obsolete since iHub Release 3.

---

## UpdateChannelOperationGroup

Obsolete since iHub Release 3.

---

## UpdateFile

Updates files or folders. To update files or folders, specify the types of updates to make using UpdateFileOperationGroup, then specify which files or folders to update.

To update the properties of a file or folder, you must have the write privilege on the file or folder. To update privileges to the file or folder, you must have the grant privilege on the file or folder.

To update a single file or folder, specify Name or Id. To update a list of files or folders, specify NameList or IdList. To update files or folders matching the specified conditions, specify Search.

**Elements****WorkingFolderId**

**String**. The ID of the working folder of the file or folder to update. Specify either WorkingFolderId or WorkingFolderName.

**WorkingFolderName**

**String**. The name of the working folder of the file or folder to update. Specify either WorkingFolderName or WorkingFolderId.

**LatestVersionOnly**

**Boolean**. Specifies whether to search all versions of the file. If True, the search includes only the latest version. The default value is False.

**Recursive**

**Boolean**. Specifies whether to search subfolders. If True, the search includes subfolders. The default value is False.

**UpdateFileOperationGroup**

Specifies the group criteria used to perform the update file operation task:

## ■ Search

**FileSearch**. The search conditions. If conditions apply to multiple fields, use ConditionArray.

## ■ NameList

**ArrayOfString**. The list of file or folder names to update. Specify either NameList or IdList.

## ■ IdList

**ArrayOfString**. The list of file or folder IDs to update. Specify either IdList or NameList.

## ■ Id

**String**. The ID of the single file or folder to update. Specify either Id or Name.

## ■ Name

**String**. The name of the single file or folder to update. Specify either Name or Id.

**IgnoreMissing**

**Boolean**. Specifies what to do if the specified file or folder does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

---

## UpdateFileOperation

Specifies the tasks to perform during the UpdateFile operation. To specify which files to update, use UpdateFile.

**Elements****SetAttributes**

**File**. The general attributes to update.

**AddDependentFilesByName**

**ArrayOfString**. The names of files to add as dependents. Specify either AddDependentFilesByName or AddDependentFilesById.

**RemoveDependentFilesByName**

**ArrayOfString**. The names of dependent files to remove. Specify either RemoveDependentFilesByName or RemoveDependentFilesById.

**SetDependentFilesByName**

**ArrayOfString**. The names of dependent files to update. Specify either SetDependentFilesByName or SetDependentFilesById.

**AddRequiredFilesByName**

**ArrayOfString**. The names of required files to add. Specify either AddRequiredFilesByName or AddRequiredFilesById.

**RemoveRequiredFilesByName**

**ArrayOfString**. The names of required files to remove. Specify either RemoveRequiredFilesByName or RemoveRequiredFilesById.

**SetRequiredFilesByName**

**ArrayOfString**. The names of required files to update. Specify either SetRequiredFilesByName or SetRequiredFilesById.

**AddDependentFilesById**

**ArrayOfString**. The IDs of files to add as dependents. Specify either AddDependentFilesById or AddDependentFilesByName.

**RemoveDependentFilesById**

**ArrayOfString**. The IDs of dependent files to remove. Specify either RemoveDependentFilesById or RemoveDependentFilesByName.

**SetDependentFilesById**

**ArrayOfString**. The IDs of dependent files to update. Specify either SetDependentFilesById or SetDependentFilesByName.

**AddRequiredFilesById**

**ArrayOfString**. The IDs of required files to add. Specify either AddRequiredFilesById or AddRequiredFilesByName.

**RemoveRequiredFilesById**

[ArrayOfString](#). The IDs of required files to remove. Specify either RemoveRequiredFilesById or RemoveRequiredFilesByName.

**SetRequiredFilesById**

[ArrayOfString](#). The IDs of required files to update. Specify either SetRequiredFilesById or SetRequiredFilesByName.

**GrantPermissions**

[ArrayOfPermission](#). The new privileges to grant. You cannot grant privileges to a file with private access.

**RevokePermissions**

[ArrayOfPermission](#). The privileges to revoke. You cannot revoke privileges to a file with private access.

**SetPermissions**

[ArrayOfPermission](#). The privileges to update. You cannot update privileges to a file with private access.

**AddArchiveRules**

[ArrayOfArchiveRule](#). The new autoarchive rules.

**RemoveArchiveRules**

[ArrayOfArchiveRule](#). The autoarchive rules to remove.

**SetArchiveRules**

[ArrayOfArchiveRule](#). The autoarchive rules to update.

**SetParameterDefinitions**

[ArrayOfParameterDefinition](#). The dynamic report parameters for third-party executable files.

## UpdateFileOperationGroup

Specifies the UpdateFileOperation element within UpdateFile.

**Element** [UpdateFileOperation](#)

The UpdateFileOperation element to use during the UpdateFile operation.

## UpdateFileType

Updates file types. To update file types, specify the types of updates to make using UpdateFileTypeOperationGroup, then specify which file types to update.

To update a single file type, specify Name or Id. To update a list of file types, specify NameList or IdList.

## UpdateFileTypeOperation

### Elements [UpdateFileTypeOperationGroup](#)

The tasks to perform.

#### **NameList**

[ArrayOfString](#). The list of file types to update.

#### **Name**

[String](#). The name of a single file type to update.

#### **IgnoreMissing**

[Boolean](#). Specifies what to do if the specified file type does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

#### **IgnoreDup**

[Boolean](#). Specifies whether to report an error for a duplicate request. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. In a name list or ID list, True prevents BIRT iHub from performing duplicate operations. If False, BIRT iHub reports an error. The default value is False.

---

## UpdateFileTypeOperation

Specifies the tasks to perform during the UpdateFileType operation.

### Elements [SetAttributes](#)

[FileType](#). The general attributes to update.

#### **SetParameterDefinitions**

[ArrayOfParameterDefinition](#). The parameters to update.

#### **SetWindowsIcon**

[Base64Binary](#). The Windows icon to display for the file type.

#### **SetLargeWebIcon**

[Base64Binary](#). The large icon to display for the file type in a browser.

#### **SetSmallWebIcon**

[Base64Binary](#). The small icon to display for the file type in a browser.

---

## UpdateFileTypeOperationGroup

Specifies the UpdateFileTypeOperation element to use during the UpdateFileType operation.

### Element [UpdateFileTypeOperation](#)

The UpdateFileTypeOperation element to use in the UpdateFileType operation.

---

## UpdateGroup

Obsolete since iHub Release 3.

---

## UpdateGroupOperation

Obsolete since iHub Release 3.

---

## UpdateGroupOperationGroup

Obsolete since iHub Release 3.

---

## UpdateJobSchedule

Updates job schedules. To update scheduled jobs, specify the types of updates to make using `UpdateJobScheduleOperationGroup`, then specify which jobs to update.

To update a single scheduled job, specify `Id`. To update a list of scheduled jobs, specify `IdList`. To update scheduled jobs matching the specified conditions, specify `Search`.

**Elements** [UpdateJobScheduleOperationGroup](#)

The tasks to perform.

**Search**

[JobScheduleSearch](#). The search conditions. If conditions apply to multiple fields, use `ConditionArray`.

**IdList**

[ArrayOfString](#). The list of job IDs to update.

**Id**

[String](#). The ID of a single job to update.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified job does not exist. If `True`, BIRT iHub ignores the request. If `False`, the operation reports an error and stops. The default value is `True`.

## UpdateJobScheduleOperation

Specifies the tasks to perform during the UpdateJobSchedule operation.

### Elements

**SetAttributes**

**JobProperties**. The general attributes to update.

**SetParameters**

**JobInputDetail**. The input parameters, output parameters, autoarchive settings, and distribution location settings to update.

**SetPrinterOptions**

**JobPrinterOptions**. The printer options to update.

**SetSchedules**

**JobSchedule**. The schedules to set.

**AddUserNotificationByName**

**ArrayOfString**. The name of the user to add to the notification list. Specify either AddUserNotificationByName or AddUserNotificationById.

**RemoveUserNotificationByName**

**ArrayOfString**. The name of the user to remove from the notification list. Specify either RemoveUserNotificationByName or RemoveUserNotificationById.

**SetUserNotificationByName**

**ArrayOfString**. The name of the user for whom to update notification. Specify either SetUserNotificationByName or SetUserNotificationById.

**AddGroupNotificationByName**

Obsolete since iHub Release 3.

**RemoveGroupNotificationByName**

Obsolete since iHub Release 3.

**SetGroupNotificationByName**

Obsolete since iHub Release 3.

**AddChannelNotificationByName**

Obsolete since iHub Release 3.

**RemoveChannelNotificationByName**

Obsolete since iHub Release 3.

**SetChannelNotificationByName**

Obsolete since iHub Release 3.

**AddUserNotificationById**

**ArrayOfString**. The ID of the user to add to the notification list. Specify either AddUserNotificationById or AddUserNotificationByName.

**RemoveUserNotificationById**

[ArrayOfString](#). The ID of the user to remove from the notification list. Specify either RemoveUserNotificationById or RemoveUserNotificationByName.

**SetUserNotificationById**

[ArrayOfString](#). The ID of the user for whom to update notification. Specify either SetUserNotificationById or SetUserNotificationByName.

**AddGroupNotificationById**

Obsolete since iHub Release 3.

**RemoveGroupNotificationById**

Obsolete since iHub Release 3.

**SetGroupNotificationById**

Obsolete since iHub Release 3.

**AddChannelNotificationById**

Obsolete since iHub Release 3.

**RemoveChannelNotificationById**

Obsolete since iHub Release 3.

**SetChannelNotificationById**

Obsolete since iHub Release 3.

**AddOutputFilePermissions**

[ArrayOfPermission](#). The output file permissions to add. You cannot add file permissions to a file with private access.

**RemoveOutputFilePermissions**

[ArrayOfPermission](#). The output file permissions to remove. You cannot remove file permissions from a file with private access.

**SetOutputFilePermissions**

[ArrayOfPermission](#). The output file permissions to update. You cannot update file permissions of a file with private access.

**SetParameterValues**

[ArrayOfParameterValue](#). The parameter values to update.

**SetQuery**

Obsolete since iHub Release 3.

**SetOutputFileAccess**

[FileAccess](#). The access rights to the output file to update. Valid values are:

- Private

Only the owner of the file and an administrator can access the file.

## UpdateJobScheduleOperationGroup

- Shared

All users and user groups specified in the access control list (ACL) for the file can access the file.

**SetWaitForEvent**

**Event.** The event to set that is being waited on. When set, processes that are waiting for this event will proceed.

---

## UpdateJobScheduleOperationGroup

Specifies the UpdateJobScheduleOperation element within the UpdateJobSchedule operation.

**Element** **UpdateJobScheduleOperation**

The UpdateJobScheduleOperation element for use with the UpdateJobSchedule operation.

---

## UpdateOpenSecurityCache

Flushes the volume's open security cache and retrieves new data from the external security source. Use UpdateOpenSecurityCache when information in the external data source has changed and must be updated immediately.

In the request, specify the list of users, user groups, and translated user or user group names to update. If no users, user groups, or translated names are specified, all users, user groups, and translated names are updated. Otherwise, only the specified items are updated.

**Elements** **UserNameList**

**ArrayOfString.** The list of user names to retrieve.

**GroupList**

Obsolete since iHub Release 3.

**RoleList**

Deprecated since iHub Release 3. Replaced by UserGroupList.

**UserGroupList**

**ArrayOfString.** The list of user groups to retrieve.

**TranslatedUserNames**

**ExternalTranslatedUserNames.** The translated user names to retrieve, such as Administrator.

**TranslatedRoleNames**

Deprecated since iHub Release 3. Replaced by TranslatedUserGroupNames.

**TranslatedUserGroupNames**

**ExternalTranslatedUserGroupNames.** The translated user group names to retrieve, such as Administrator.

---

## UpdateRole

Deprecated since iHub Release 3. Replaced by [UpdateUserGroup](#).

---

## UpdateUser

Updates user properties. To update users, specify the types of updates to make using **UpdateUserOperationGroup**, then specify which users to update.

To update a single user, specify Name or Id. To update a list of users, specify NameList or IdList. To update users matching the specified conditions, specify Search.

**Elements** [UpdateUserOperationGroup](#)

The tasks to perform. The valid value are Search, IdList, NameList, Id, and Name.

**Search**

[UserSearch](#). The search conditions. If search conditions apply to multiple fields, use ConditionArray.

**IdList**

[ArrayOfString](#). The list of user IDs to update. Specify either IdList or NameList.

**NameList**

[ArrayOfString](#). The list of user names to update. Specify either NameList or IdList.

**Id**

[String](#). The ID of a single user to update. Specify either Id or Name.

**Name**

[String](#). The name of a single user to update. Specify either Name or Id.

**IgnoreMissing**

[Boolean](#). Specifies what to do if the specified user does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

**IgnoreDup**

[Boolean](#). Specifies whether to report an error for a duplicate request. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. In a name list or ID list, True prevents BIRT

## UpdateUserGroup

iHub from performing duplicate operations. If False, BIRT iHub reports an error. The default value is False.

### SkipPermissionError

Obsolete since iHub Release 3.

---

## UpdateUserGroup

Updates a user group. Available only to users in the Administrator user group.

### Elements

[UpdateUserGroupOperationGroup](#)

The UpdateUserGroupOperation element to use during the UpdateUserGroup operation.

#### Search

[UserGroupSearch](#). The search condition that specifies which user groups to delete.

#### IdList

[ArrayOfString](#). The list of user group IDs to delete. Specify either IdList or NameList.

#### NameList

[ArrayOfString](#). The list of user group names to delete. Specify either NameList or IdList.

#### Id

[String](#). The ID of the single user group to delete. Specify either Id or Name.

#### Name

[String](#). The name of the single user group to delete. Specify either Name or Id.

#### IgnoreMissing

[Boolean](#). Specifies what to do if the specified user group does not exist. If True, BIRT iHub ignores the request. If False, the operation reports an error and stops. The default value is True.

#### IgnoreDup

[Boolean](#). Specifies whether to report an error when creating the user group if one with the same name already exists. BIRT iHub always rejects a duplicate request regardless of the IgnoreDup setting. If True, BIRT iHub does not report an error. If False, BIRT iHub reports an error. The default value is False.

---

## UpdateUserGroupOperation

Specifies the tasks to perform during the UpdateUserGroup operation.

**Elements**    **SetAttributes**

[UserGroup](#). The general attributes to update.

**AssignedToUsersByName**

[ArrayOfString](#). The names of user groups to which to assign users. Specify either AssignedToUsersByName or AssignedToUsersById.

**DroppedFromUsersByName**

[ArrayOfString](#). The names of user groups from which to delete users. Specify either DroppedFromUsersByName or DroppedFromUsersById.

**SetBearingUsersByName**

[ArrayOfString](#). The names of user groups to which to reassign users. Specify either SetBearingUsersByName or SetBearingUsersById.

**AddChildUserGroupsByName**

[ArrayOfString](#). The names of user groups to add to the user group as descendant user groups. Specify either AddChildUserGroupsByName or AddChildUserGroupsById.

**RemoveChildUserGroupsByName**

[ArrayOfString](#). The names of descendant user groups to remove from the UserGroup. Specify either RemoveChildUserGroupsByName or RemoveChildUserGroupsById.

**SetChildUserGroupsByName**

[ArrayOfString](#). The names of the user group's descendant user groups. Specify either SetChildUserGroupsByName or SetChildUserGroupsById.

**AddParentUserGroupsByName**

[ArrayOfString](#). The names of user groups to add to the user group as its ascendant user groups. Specify either AddParentUserGroupsByName or AddParentUserGroupsById.

**RemoveParentUserGroupsByName**

[ArrayOfString](#). The name of the parent user group to remove. Specify either RemoveParentUserGroupsByName or RemoveParentUserGroupsById.

**SetParentUserGroupsByName**

[ArrayOfString](#). The names of the user group's ascendant user groups. Specify either SetParentUserGroupsByName or SetParentUserGroupsById.

**AssignedToUsersById**

[ArrayOfString](#). The IDs of user groups to which to assign users. Specify either AssignedToUsersById or AssignedToUsersByName.

**DroppedFromUsersById**

[ArrayOfString](#). The IDs of user groups from which to delete users. Specify either DroppedFromUsersById or DroppedFromUsersByName.

## UpdateUserGroupOperationGroup

### **SetBearingUsersById**

[ArrayOfString](#). The IDs of user groups to which to reassign users. Specify either SetBearingUsersById or SetBearingUsersByName.

### **AddChildUserGroupsById**

[ArrayOfString](#). The IDs of user groups to add to the user group as descendant user groups. Specify either AddChildUserGroupsByName or AddChildUserGroupsById.

### **RemoveChildUserGroupsById**

[ArrayOfString](#). The IDs of descendant user groups to remove from the user group. Specify either RemoveChildUserGroupsById or RemoveChildUserGroupsByName.

### **SetChildUserGroupsById**

[ArrayOfString](#). The IDs of the user group's descendant user groups. Specify either SetChildUserGroupsById or SetChildUserGroupsByName.

### **AddParentUserGroupsById**

[ArrayOfString](#). The IDs of user groups to add to the user group as its ascendant user groups. Specify either AddParentUserGroupsById or AddParentUserGroupsByName.

### **RemoveParentUserGroupsById**

[ArrayOfString](#). The IDs of the ascendant user groups to remove. Specify either RemoveParentUserGroupsByName or RemoveParentUserGroupsById.

### **SetParentUserGroupsById**

[ArrayOfString](#). The IDs of the user group's ascendant user groups. Specify either SetParentUserGroupsById or SetParentUserGroupsByName.

---

## UpdateUserGroupOperationGroup

Specifies the UpdateUserGroupOperation element to use during the UpdateUserGroup operation.

**Element** [UpdateUserGroupOperation](#)

The UpdateUserGroupOperation element for use with the UpdateUserGroup operation.

---

## UpdateUserOperation

Specifies the tasks to perform during the UpdateUser operation.

**Elements** [SetAttributes](#)

[User](#). The general attributes to update.

**AddToGroupsByName**

Obsolete since iHub Release 3.

**RemoveFromGroupsByName**

Obsolete since iHub Release 3.

**SetGroupMembershipsByName**

Obsolete since iHub Release 3.

**AssignRolesByName**

Deprecated since iHub Release 3. Replaced by AssignUserGroupsByName.

**DropRolesByName**

Deprecated since iHub Release 3. Replaced by DropUserGroupsByName.

**SetRolesByName**

Deprecated since iHub Release 3. Replaced by SetUserGroupsByName.

**AssignUserGroupsByName**

[ArrayOfString](#). The names of user groups to assign to users. Specify either AssignUserGroupsByName or AssignUserGroupsById.

**DropUserGroupsByName**

[ArrayOfString](#). The names of user groups from which to remove users. Specify either DropUserGroupsByName or DropUserGroupsById.

**SetUserGroupsByName**

[ArrayOfString](#). The names of user groups to update. Specify either SetUserGroupsByName or SetUserGroupsById.

**SubscribeToChannelsByName**

Obsolete since iHub Release 3.

**UnsubscribeFromChannelsByName**

Obsolete since iHub Release 3.

**SetChannelSubscriptionsByName**

Obsolete since iHub Release 3.

**AddToGroupsById**

Obsolete since iHub Release 3.

**RemoveFromGroupsById**

Obsolete since iHub Release 3.

**SetGroupMembershipsById**

Obsolete since iHub Release 3.

**AssignRolesById**

Deprecated since iHub Release 3. Replaced by AssignUserGroupsById.

## UpdateUserOperation

### **DropRolesById**

Deprecated since iHub Release 3. Replaced by DropUserGroupsById.

### **SetRolesById**

Deprecated since iHub Release 3. Replaced by SetUserGroupsById.

### **AssignUserGroupsByID**

**ArrayOfString**. The IDs of user groups to assign to users. Specify either AssignUserGroupsByName or AssignUserGroupsById.

### **DropUserGroupsByID**

**ArrayOfString**. The IDs of user groups from which to remove users. Specify either DropUserGroupsByName or DropUserGroupsById.

### **SetUserGroupsByID**

**ArrayOfString**. The IDs of user groups to update. Specify either SetUserGroupsByName or SetUserGroupsById.

### **SubscribeToChannelsById**

Obsolete since iHub Release 3.

### **UnsubscribeFromChannelsById**

Obsolete since iHub Release 3.

### **SetChannelSubscriptionsById**

Obsolete since iHub Release 3.

### **AddFileCreationPermissions**

**ArrayOfString**. Grants users the permissions to add files.

### **RemoveFileCreationPermissions**

Revokes users' ability to add files.

### **SetFileCreationPermissions**

**ArrayOfString**. Modifies users' ability to add files.

### **SetPrinterOptions**

**ArrayOfString**. The printer options to set for the users.

### **SetLicenseOptions**

**ArrayOfString**. The license options to set for the users.

### **AddLicenseOptions**

**ArrayOfString**. Grants users the right to add license options.

### **RemoveLicenseOptions**

**ArrayOfString**. Removes the right of users to add license options.

---

## UpdateUserOperationGroup

Specifies the UpdateUserOperation element to use during the UpdateUser operation.

**Element** [UpdateUserOperation](#)

The UpdateUserOperation element for use with the UpdateUser operation.

---

## UpdateVolumeProperties

Updates a volume. To update a volume, specify the types of updates to make using UpdateVolumeOperationGroup, then specify which volume to update.

**Element** [UpdateVolumePropertiesOperationGroup](#)

The tasks to perform.

---

## UpdateVolumePropertiesOperation

Specifies the tasks to perform during the UpdateVolumeProperties operation.

**Elements** [SetAttributes](#)

[Volume](#). Contains one or more of the following volume properties to update:

- DefaultPrinterName
- MaxJobRetryCount
- JobRetryInterval
  - The interval between retry attempts. Measured in seconds.
- DefaultViewingPreference
- DHTMLPageCaching
- DHTMLPageCachingExpiration
  - If DHTMLPageCaching is True, set the DHTMLPageCachingExpiration to a valid value. To disable DHTMLPageCaching, set DHTMLPageCachingExpiration to -1.

[SetPrinterOptions](#)

[ArrayOfPrinterOptions](#). The volume printer options to update.

[SetSystemPrinters](#)

[ArrayOfPrinter](#). The printer to set as the system printer for the volume.

[ClearSystemPrinters](#)

[ArrayOfString](#). The printer to remove from the volume.

## UpdateVolumePropertiesOperationGroup

### **SetAutoArchiveSchedules**

**JobSchedule.** The start of the autoarchive schedule for folders and files on the volume.

---

## UpdateVolumePropertiesOperationGroup

Specifies the UpdateVolumePropertiesOperation element to use during the UpdateVolumeProperties operation.

**Element** **UpdateVolumePropertiesOperation**

The UpdateVolumePropertiesOperation element for use with the UpdateVolumeProperties operation.

---

## User

A complex data type that describes a user.

**Elements**

**Id**

**String.** Optional. The user ID.

**Name**

**String.** Optional. The user name. A user name is a string of 1 to 256 characters, including any character except a control character. A user name is not case-sensitive. BIRT iHub stores a user name in mixed case, always displaying it exactly the way it was typed during creation.

**Password**

**String.** Optional. The user password. A password is a string of 1 to 256 characters, including any character except a control character or space. Security experts recommend using passwords of at least eight characters, including mixed-case alphabetic and numeric characters. A password is case-sensitive. The Administrator can change any user password. Users can only change their own passwords. BIRT iHub encrypts a user password.

**EncryptedPwd**

**String.** Optional. The encrypted password of the user.

**Description**

**String.** Optional. The description of the user.

**IsLoginDisabled**

**Boolean.** Optional. Specifies whether the user can log in.

**EmailAddress**

**String.** Optional. The user e-mail address.

**HomeFolder**

**String.** Optional. The user home folder.

**ViewPreference**

**String.** Optional. The user viewer, Default or DHTML.

**MaxJobPriority**

**Long.** Optional. The maximum priority that the user can assign to a job.

**MaxNotices**

**Long.** Optional. The maximum number of notices that the user can retain.

**SendNoticeForSuccess**

**Boolean.** Optional. Specifies whether the BIRT iHub sends success notices to the user.

**SendNoticeForFailure**

**Boolean.** Optional. Specifies whether the BIRT iHub sends failure notices to the user.

**SuccessNoticeExpiration**

**Long.** Optional. Specifies the minimum number of minutes success notices remain in the Completed folder. After this time elapses, BIRT iHub removes the notices the next time it removes requests from the volume's Requests\Completed folder. If not set or set to 0, DefaultSuccessNoticeExpiration specified in Volume is used. To set the user success notices to never expire, set the value to 0xffffffff.

**FailureNoticeExpiration**

**Long.** Optional. Specifies the minimum number of minutes failure notices remain in the Completed folder. After this time elapses, BIRT iHub removes the notices the next time it removes requests from the volume's Requests\Completed folder. If not set or set to 0, DefaultFailureNoticeExpiration specified in Volume is used. To set the user failure notices to never expire, set the value to 0xffffffff.

**SendEmailForSuccess**

**Boolean.** Optional. Specifies whether the BIRT iHub sends success notices in an e-mail message to the user.

**SendEmailForFailure**

**Boolean.** Optional. Specifies whether the BIRT iHub sends failure notices in an e-mail message to the user.

**AttachReportInEmail**

**Boolean.** Optional. Specifies whether to attach a report to an e-mail completion notice.

**DefaultPrinterName**

**String.** Optional. The name of the user default printer.

## UserCondition

A complex data type that represents the field on which to perform a search and the condition to match.

**Elements** **Field**

**String**. The field on which to perform a search.

**Match**

**String**. The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

```
05\‐25\‐04Report.rptdocument
05 [-] 25 [-] 04Report.rptdocument
```

---

## UserField

A simple data type that describes the fields within a user element.

**Element** **String**. One of the following values:

**Name**

The user name.

**Description**

The description of the user.

**IsLoginDisabled**

Specifies whether the user can log in.

**EmailAddress**

The user e-mail address.

**HomeFolder**

The user home folder.

**ViewPreference**

The user viewer, Default or DHTML.

**MaxJobPriority**

The maximum priority that the user can assign to a job.

**SuccessNoticeExpiration**

Specifies the minimum number of minutes success notices remain in the Completed folder.

**FailureNoticeExpiration**

Specifies the minimum number of minutes failure notices remain in the Completed folder.

**SendNoticeForSuccess**

Specifies whether the BIRT iHub sends success notices to the user.

**SendNoticeForFailure**

Specifies whether the BIRT iHub sends failure notices to the user.

**SendEmailForSuccess**

Specifies whether the BIRT iHub sends success notices in an e-mail message to the user.

**SendEmailForFailure**

Specifies whether the BIRT iHub sends failure notices in an e-mail message to the user.

**AttachReportInEmail**

Specifies whether to attach a report to an e-mail completion notice.

**DefaultPrinterName**

The name of the user default printer.

---

## UserGroup

A complex data type that describes a user group.

**Elements****Id**

[String](#). Optional. The user group ID.

**Name**

[String](#). Optional. The name of the user group. Cannot exceed 50 characters.

**Description**

[String](#). Optional. The description of the user group. Cannot exceed 500 characters.

---

## UserGroupCondition

A complex data type that represents the field on which to perform a user group search and the condition to match.

**Elements****Field**

[UserGroupField](#). The field on which to perform a search.

## UserGroupField

### Match

[String](#). The condition to match. If the condition includes special characters, for example a dash, either escape the special character with a backslash (\) or enclose it in brackets ([]). For example, to search for a file 05-25-04Report.rptdocument, specify one of the following:

```
05\ -25\ -04Report .rptdocument
05 [-] 25 [-] 04Report .rptdocument
```

---

## UserGroupField

A simple data type that lists user group fields on which a search can be performed.

**Element** [String](#). One of the following values:

### Name

The name of the user group.

### Description

The description of the user group.

---

## UserGroupSearch

A complex data type that represents a user group search.

**Elements** **Condition**

[UserGroupCondition](#). Optional. The search condition. Specify this value of ConditionArray.

### ConditionArray

[ArrayOfUserGroupCondition](#). Optional. An array of search conditions. Specify this value of Condition.

### ParentUserName

[String](#). Optional. The name of the parent user group of which the user is a member. Specify one of this value, ChildUserName, AssignedToUserName, ParentUserId, ChildUserId, or AssignedToUserId.

### ChildUserName

[String](#). Optional. The name of the child user group to which the user belongs. Specify one of this value, ParentUserName, AssignedToUserName, ParentUserId, ChildUserId, or AssignedToUserId.

### WithRightsToChannelName

Obsolete since iHub Release 3.

**AssignedToUserName**

**String.** Optional. The user name to which the user group is assigned. Specify one of this value, ParentUserGroupName, ChildUserGroupName, ParentUserGroupId, ChildUserGroupId, or AssignedToUserId.

**ParentUserGroupId**

**String.** Optional. The ID of the parent user group to which the group belongs. Specify one of this value, ParentUserGroupName, ChildUserGroupName, AssignedToUserName, ChildUserGroupId, or AssignedToUserId.

**ChildUserGroupId**

**String.** Optional. The ID f the child user group to which the group belongs. Specify one of this value, ParentUserGroupName, ChildUserGroupName, AssignedToUserName, ParentUserGroupId, or AssignedToUserId.

**WithRightsToChannelId**

Obsolete since iHub Release 3.

**AssignedToUserId**

**String.** Optional. The user ID to which the user group is assigned. Specify one of this value, ParentUserGroupName, ChildUserGroupName, AssignedToUserName, ParentUserGroupId, or ChildUserGroupId.

**FetchSize**

**Int.** Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

**Boolean.** Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

**Int.** Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

**String.** Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specifies the same search criteria as in the original call.

## UserSearch

A complex data type that represents a user search.

**Elements****Condition**

**UserCondition.** Optional. The search condition. Specify this value of ConditionArray.

**ConditionArray**

[ArrayOfUserCondition](#). Optional. An array of search conditions. Specify this value of ConditionArray.

**MemberOfGroupName**

Obsolete since iHub Release 3.

**WithRoleName**

Deprecated since iHub Release 3. Replaced by WithUserGroupName.

**WithUserGroupName**

[String](#). Optional. The name of the user group to which the user belongs. Specify one of this value, WithUserId, or WithLicenseOption.

**SubscribedToChannelName**

Obsolete since iHub Release 3.

**MemberOfGroupId**

Obsolete since iHub Release 3.

**WithRoleId**

Deprecated since iHub Release 3. Replaced by WithUserGroupId.

**WithUserId**

[String](#). Optional. The ID of the user group to which the user belongs. Specify one of this value, WithUserName, or WithLicenseOption.

**WithLicenseOption**

[String](#). Optional. The name of the license option assigned to the user. Specify one of this value, WithUserName, or WithUserId.

**SubscribedToChannelId**

Obsolete since iHub Release 3.

**FetchSize**

[Int](#). Optional. The maximum number of records to retrieve and return in a result set. The default value is 500.

**FetchDirection**

[Boolean](#). Optional. If True, records are retrieved forward. If False, records are retrieved backward. The default value is True.

**CountLimit**

[Int](#). Optional. The maximum number of records to count. By default, CountLimit is equal to FetchSize. To count all records, set CountLimit to -1.

**FetchHandle**

[String](#). Optional. Retrieves more items from the result set. In the second and subsequent calls for data, specifies the same search criteria as in the original call.

---

## VersioningOption

A simple data type that specifies the options for handling the latest existing version when uploading a file.

**Element** [String](#). One of the following values:

**CreateNewVersion**

Always creates a new version. This is the default value.

**ReplaceLatestIfNoDependents**

Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub creates a new version instead of replacing the existing version.

**ReplaceLatestDropDependency**

Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub drops the dependency.

**ReplaceLatestMigrateDependency**

Replaces the latest existing version if it does not have any dependent files. If the existing version has any dependents, BIRT iHub moves the dependency to the new version.

---

## ViewParameter

A complex data type that describes a viewing parameter.

**Elements** [Format](#)

[String](#). Optional. The format in which the report displays. Valid formats are:

- CSS
- DHTML
- DHTMLLong
- DHTMLRaw
- ExcelData
- ExcelDisplay
- ImageMapURL

Supports users clicking a point in a chart to navigate to different report sections

- PDF
- PPT

## **ViewParameter**

- PPTFullyEditable
- Reportlet
  - Valid only if ShowInReportlet is enabled during report design
- RTF
- RTFFullyEditable
- XMLCompressedDisplay
- XMLCompressedExcel
- XMLCompressedPDF
- XMLCompressedPPT
- XMLCompressedReportlet
- XMLCompressedRTF
- XMLData
- XMLDisplay
- XMLReportlet
- XMLStyle

### **UserAgent**

**String.** Optional. The browser to use for report viewing, such as Mozilla/4.0.

### **ScalingFactor**

**Long.** Optional. Adapts the size of a Reportlet to the Reportlet frame.

### **AcceptEncoding**

**String.** Optional. The list of encoding methods the browser supports.

### **ViewOperation**

**String.** Optional. The view operation, View or Print.

### **PathInformation**

**String.** Optional. The path to the report.

### **EmbeddedObjPath**

**String.** Optional. The base URL to prepend to a static or dynamic object in a report. When viewing a report in a browser, the URL of an image, chart, JavaScript, or another resource refers to the volume. Use EmbeddedObjPath to change this URL.

### **RedirectPath**

**String.** Optional. Maps from the current URL to a new target.

### **PdfQuality**

**Long.** Optional. The viewing quality of a PDF.

---

## Volume

A complex data type that describes a volume.

**Elements****Name**

[String](#). The name of the volume.

**ActuateVersion**

[String](#). Optional. The version number.

**ActuateBuildNumber**

[String](#). Optional. The build number.

**SecurityIntegrationOption**

[Long](#). Optional. The security integration option.

**OpenSecuritySelectUsersOfRole**

Deprecated since iHub Release 3. Replaced by OpenSecuritySelectUsersOfUserGroup.

**OpenSecuritySelectUsersOfUserGroup**

[Boolean](#). Optional. Applies only if using external registration security level.

Indicates whether the SelectUsers operation for a user group is supported. If the operation is supported, iHub enables appropriate features in iHub Visualization Platform.

**OpenSecuritySelectGroupsOfUser**

Obsolete since iHub Release 3.

**DefaultPrinterName**

[String](#). Optional. The name of the default printer.

**MaxJobRetryCount**

[Long](#). Optional. The maximum number of retry attempts.

**JobRetryInterval**

[Long](#). Optional. The interval between retry attempts. Measured in seconds.

**DefaultViewingPreference**

Obsolete since iHub Release 2.

**DHTMLPageCaching**

[Boolean](#). Optional. True enables DHTML page caching.

**DHTMLPageCachingExpirationAge**

[Long](#). Optional. If DHTMLPageCaching is True, set

DHTMLPageCachingExpirationAge to a valid value. To disable

DHTMLPageCaching, set DHTMLPageCachingExpirationAge to -1.

## Weekly

### **IsAutoArchiveRunning**

**Boolean.** Optional. Determines whether an archive pass is currently running. If True, an archive pass is running.

### **AuthorizationIsExternal**

**Boolean.** Optional. Specifies whether user registration is external.

### **ConnectionPropertiesAreExternal**

**Boolean.** Optional. Specifies whether connection properties are externalized using the Report Server Security Extension (RSSE).

### **DefaultSuccessNoticeExpiration**

**Long.** Optional. Specifies the minimum number of minutes success notices remain in the Completed folder. After this time elapses, BIRT iHub removes the notices the next time it removes requests from the volume's Requests\Completed folder. This time applies to all users whose SuccessNoticeExpiration time is not set or is set to 0. The default value is 0, which means notices never expire.

### **DefaultFailureNoticeExpiration**

**Long.** Optional. Specifies the minimum number of minutes failure notices remain in the Completed folder. After this time elapses, BIRT iHub removes the notices the next time it removes requests from the volume's Requests\Completed folder. This time applies to all users whose SuccessNoticeExpiration time is not set or is set to 0. The default value is 0, which means notices never expire.

### **ResourcePath**

**String.** Optional. The resource path to the volume.

---

## Weekly

A complex data type describing weekly job scheduling.

### **Elements** **FrequencyInWeeks**

**Long.** The number of times a job is to run, in weeks.

### **RunOn**

**String.** The day to run the job.

### **OnceADay**

**String.** Optional. The time to run the job.

### **Repeat**

**Repeat.** Optional. The number of times to repeat the schedule.

# Text string limits in Actuate operations

Actuate's internal data store imposes a fixed upper limit on the length of certain text strings. An application that uses elements such as user names, URLs, file types, and descriptions must adhere to these limits. Table 15-1 lists the maximum field lengths for text elements in BIRT iHub Visualization Platform and for elements you create using Actuate Information Delivery API.

**Table 15-1** Text string limits in Actuate operations

Complex data type	Element name	Maximum length, in characters
ArchiveRule	FileType	20
Attachment	ContentEncoding	10
File	Name	255
	FileType	20
	Description	500
	VersionName	100
FileType	Name	20
	DriverName	100
	MutexClass	50
	ShortDescription	40

*(continues)*

**Table 15-1** Text string limits in Actuate operations (continued)

Complex data type	Element name	Maximum length, in characters
FileType ( <i>continued</i> )	LongDescription	60
	LocalExtension	20
	OutputType	20
	SmallImageURL	100
	LargeImageURL	100
	ContentType	200
JobProperties	JobName	100
	InputFileName	276
	RequestedOutputFileName	1000
	ActualOutputFileName	1000
	RequestedHeadline	100
	ActualHeadline	100
JobNotice	JobName	100
	OutputFileName	276
	OutputFileVersionName	100
	Headline	100
JobPrinterOptions	PageRange	20
	PrintToFile	256
Printer	Name	50
	Manufacturer (UNIX only)	50
	Model (UNIX only)	50
	Location (UNIX only)	50
	Description	100
	Orientation	20
	PageSize	50
	Resolution	20
	PaperTray	50
JobSchedule	Duplex	20
	TimeZoneName	32

**Table 15-1** Text string limits in Actuate operations (continued)

Complex data type	Element name	Maximum length, in characters
User	Name	256
	Password	256
	EmailAddress	80
	DefaultPrinterName	50
	Description	100
UserGroup	Name	50
	Description	500



# Part **Five**

---

**Accessing BIRT applications using URIs**



# 16

## Actuate application URIs

This chapter contains the following topics:

- Working with Actuate application URIs
- Actuate application URIs overview
- Actuate application URIs quick reference
- Common URI parameters
- Visualization Platform Struts actions
- Actuate application URIs reference

---

# Working with Actuate application URIs

Uniform resource identifiers (URIs) convey user requests to the iHub System. URIs access functionality including generating and storing reports, managing volume contents, and viewing reports.

## Visualization Platform URIs

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

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

- <web server> is the fully qualified domain name or IP address of the machine running the application server or servlet engine. You can use 127.0.0.1 as a trusted application's machine name if your local machine is running the server.
- <port> is the port on which you access the application server or page or servlet engine. The default port for Visualization Platform is 8700.
- iportal is the default context root for accessing the Visualization Platform pages.
- <path> is the directory containing the page to invoke.
- <page> is the name of the page or method.
- <type> is jsp or do.
- <parameter=value> specifies the parameters and values that the page requires.

For example, to view the login page, Visualization Platform uses a URI with the following format:

```
http://<web server>:<port>/iportal
/login.jsp?TargetPage=<folder/file>
```

- iportal/login.jsp is the JSP that provides default login functionality for Visualization Platform.
- TargetPage is the viewframeset.jsp parameter that specifies the page to direct the user to after the login completes.
- <folder/file> is the complete pathname for the file that the client opens after the login completes.

## Using a special character in a URI

Visualization Platform URIs use encoding for characters that a browser can misinterpret. The following example uses hexadecimal encoding in the Visualization Platform URI to display the report, Newsfeed.rptdesign, from a volume:

```
http://127.0.0.1:8700/iportal/executereport.do?__requesttype=immediate&__executableName=%2fNewsfeed%2erptdesign&__vp=server1
```

You do not have to use hexadecimal encoding in all circumstances. Use the encoding only when the possibility of misinterpreting a character exists. The following unencoded URI displays the same report as the preceding URI:

```
http://127.0.0.1:8700/iportal/executereport.do?__requesttype=immediate&__executableName=\Newsfeed.rptdesign&__vp=server1
```

Always encode characters that have a specific meaning in a URI when you use them in other ways. Table 16-1 describes the available character substitutions. An ampersand introduces a parameter in a URI, so you must encode an ampersand that appears in a value string. For example, use:

&company=AT%26T

instead of:

&company=AT&T

**Table 16-1** Encoding sequences for use in URIs

Character	Encoded substitution
ampersand (&)	%26
asterisk (*)	%2a
at (@)	%40
backslash (\)	%5c
colon (:)	%3a
comma (,)	%2c
dollar sign (\$)	%24
double quote (")	%22
equal (=)	%3d
exclamation (!)	%21
forward slash (/)	%2f
greater than (>)	%3e
less than (<)	%3c
number sign (#)	%23

(continues)

**Table 16-1** Encoding sequences for use in URIs (continued)

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

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

```
<%-- Import the StaticFuncs class. --%>
<%@ page import="com.actuate.reportcast.utils.*" %>

<%
String url =
 "http://127.0.0.1:8700/iportal/getfolderitems.do?folder=" +
 StaticFuncs.encode("/Training/Sub Folder");
response.sendRedirect(url);
%>
```

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

```
/Training/Sub Folder
to:
%2fTraining%2fSub%20Folder
```

## UTF-8 encoding

All communication between Visualization Platform and iHub uses UTF-8 encoding. UTF-8 encoding is the default encoding that web browsers support. For 8-bit (single-byte) characters, UTF-8 content appears the same as ANSI content. However, if extended characters are used (typically for languages that require large character sets), UTF-8 encodes these characters with two or more bytes.

UTF-8 encoding support is encoded for all Visualization Platform web pages. When customizing these pages or adding customized web pages to a

Visualization Platform web application, provide UTF-8 encoding support using the following code:

```
<META
HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=utf-8">
```

---

## Actuate application URIs overview

This chapter describes Actuate application URIs. The following sections provide quick reference tables and detailed reference information about URIs. A Visualization Platform URI is a directive to an Actuate application to perform an action, such as showing a list of files.

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

---

## Actuate application URIs quick reference

Table 16-2 lists the Actuate application URIs. For more information about the application directory structure, see *Managing Volumes and Users*.

**Table 16-2**    Visualization Platform URI pages

Visualization Platform page	Description
browse file page	Provides file and folder browsing functionality for the submit request pages.
browse page	See browse file page.
canceljob page	See request drop page.
create folder page	Creates a folder.
dashboard page	Provides the dashboard interface.
delete job page	Deletes a scheduled job.
delete status page	Deletes the completed job notice.
detail page	Supports error handling and presenting object details.
do_update page	See output page.
drop page	Supports deleting files or canceling running jobs.
error page	Retrieves an error message from the exception or the request and displays it.

(continues)

**Table 16-2** Visualization Platform URI pages (continued)

Visualization Platform page	Description
execute report page	Submits a run report job request to the server.
file or folder detail page	Supports error handling and presenting file or folder details.
file and folder index page	Lists all files and folders for a given path.
getfiledetails page	See file or folder detail page.
getfolderitems page	See file and folder index page.
getjobdetails page	See request detail page.
login page	Logs in to the reporting web application.
logout page	Logs the user out of the current session and clears all user settings, such as filters.
myfiles page	The display page for the repository dashboard with the Visualization Platform banner.
output page	Presents a form to specify output information for report jobs, such as report object name and location.
pending page	Presents a list of the request parameters.
pending page	Lists all requests awaiting execution.
ping page	Diagnostics for iHub System components.
privileges page	Sets file and folder privileges.
request detail page	Supports error handling and presenting request details.
running page	Lists all requests currently executing.
scheduled job page	Lists all requests awaiting execution at specified dates and times.
search folders page	Searches folders recursively for files and folders.
submit job page	Submits a scheduled job request to the server.
BIRT Viewer page	Displays BIRT documents and the toolbar.

---

## Common URI parameters

All Visualization Platform URIs have the parameters shown in Table 16-3. String values that are too long are truncated for all parameters. The web browser that you use determines the length of parameters. The common URI parameters support Visualization Platform authentication using cookies.

**Table 16-3** Common Visualization Platform URI parameters

URI parameter	Description
forceLogin	True to force a login, False to display the login page. The default is False. For example, when switching between volumes and using a Visualization Platform security manager class, set forceLogin=true to force the Visualization Platform Login module to call the security manager to perform the login operation.
iPortalID	The unique authentication ID assigned to the user upon successful login. Use this parameter in conjunction with the userID parameter to ensure that a user's personalized settings appear in the Visualization Platform pages.
locale	The current user's locale, such as U.S. English (en-US).
password	The password associated with the userID.
serverURL	The URI that accesses the iHub, such as http://Services:8000.
timezone	The current user's time zone.
userID	The user's unique identifier, required to log in to the repository. Use this parameter in conjunction with the iPortalID parameter to ensure that a user's personalized settings appear in the Visualization Platform pages.
volume	The volume to which the user connects. This parameter overrides the volume from the __vp parameter if they are used together.
__vp	The name of a server configured in VolumeProfile.xml. Visualization Platform uses the volume information in a VolumeProfile entry except when a volume parameter specifies a different one.

The following Visualization Platform URI shows most of the common URI parameters in use:

```
http://127.0.0.1:8700/iportal/dashboard?folder=/Training
 &volume=Encyc2&locale=en_AU&userID=Mike&password=pw123
 &serverURL=http://server1:8000&timeZone=Australia/Perth
```

This URI lists the contents of the Training folder in the Encyc2 volume on the iHub named server1 at port 8000. The locale is set to Australian English and the time zone is Australia/Perth (GMT plus eight hours). The user is Mike and the password is pw123. The password is shown in plain text, as entered.

If the server and volume information for server1 above is configured as a Volume Profile, you can use a simplified URL as shown in the following lines:

```
http://127.0.0.1:8700/iportal/dashboard?folder=/Training
 &__vp=server1&locale=en_AU&userID=Mike&password=pw123
 &timeZone=Australia/Perth
```

---

## Visualization Platform Struts actions

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

Table 16-4 lists the global forwards defined in struts-config.xml.

**Table 16-4** Visualization Platform global forwards

Action	Forward
authexpired	/login.do
browsefile	/browsefile.do
canceljob	/canceljob.do
da	/da
dashboard	/dashboard
deletefile	/deletefile.do
deletejob	/deletejob.do
deletejobnotice	/deletejobnotice.do
deletejobschedule	/deletejobschedule.do
downloadfile	/servlet/DownloadFile
error	/private/common/errors/errorpage.jsp
executedocument	/executedocument.do
executereport	/executereport.do
getjobdetails	/getjobdetails.do
getrequesterjobdetails	/getrequesterjobdetails.do
getsavedsearch	/viewer/getsavedsearch.do
goto	/private/common/goto.jsp
iv	/iv
login	/login.do
logout	/logout.do
requestercanceljob	/requestercanceljob.do
requesterdeletejob	/requesterdeletejob.do
requesterdeletejobschedule	/requesterdeletejobshdule.do
ssologout	/ssologout.do
viewframeset	/viewer/viewframeset.jsp

**Table 16-4** Visualization Platform global forwards

Action	Forward
viewpage	/servlet/ViewPage
wr	/wr

Table 16-5 lists the action, input JSP, and forward name and path defined in struts-config.xml.

**Table 16-5** Visualization Platform actions

Action	Input JSP	Forward name path
/browsefile	/iportal/activePortal /private/newrequest /browse.jsp	name=success path=/iportal/activePortal/private /newrequest/browse.jsp  name=browseFile path=/iportal/activePortal/private /common/browseFile.jsp
/browseportletfile	/iportal/portlets /browsefile.jsp	name=success path=/iportal/portlets/browsefile.jsp
/canceljob		name=success path=/iportal/activePortal/private /jobs/joboperationstatus.jsp
/cancelreport		name=Succeeded path=/iportal/activePortal/viewer /closewindow.jsp?status=succeeded  name=Failed path=/iportal/activePortal/viewer /closewindow.jsp?status=failed  name=InActive path=/iportal/activePortal/viewer /closewindow.jsp?status=inactive
/createfolder		name=success path=/getfolderitems.do  name=cancel path=/getfolderitems.do  name=showform path=/iportal/activePortal/private /filesfolders/createfolder.jsp

(continues)

**Table 16-5** Visualization Platform actions (continued)

Action	Input JSP	Forward name path
/deletefile		name=success path=/iportal/activePortal/private/filesfolders/deletefilestatus.jsp  name=error path=/iportal/activePortal/private/filesfolders/deletefilestatus.jsp  name=confirm path=/iportal/activePortal/private/filesfolders/confirm.jsp
/deletejob		name=success path=/iportal/activePortal/private/jobs/joboperationstatus.jsp
/deletejobnotice		name=success path=/iportal/activePortal/private/jobs/joboperationstatus.jsp
/deletejobschedule		name=success path=/iportal/activePortal/private/jobs/joboperationstatus.jsp
/editTableRow	/iportal/activePortal/private/parameters/table/roweditor.jsp	name=close path=/iportal/activePortal/private/parameters/table/close.jsp  name=tableRowEditor path=/iportal/activePortal/private/parameters/table/roweditor.jsp
/executedocument		name=success path=/executereport.do
/executereport	/private/newrequest/newrequest.jsp	name=viewbirt path=/iv  name=viewreport path=/servlet/DownloadFile  name=wait path=/iportal/activePortal/private/newrequest/waitforexecution.jsp
/filefoldersprivilege	/iportal/activePortal/private/filesfolders/privilege.jsp	name=success path=/getfolderitems.do
/getfiledetails		name=success path=/iportal/activePortal/private/filesfolders/filedetail.jsp

**Table 16-5** Visualization Platform actions (continued)

Action	Input JSP	Forward name path
/getfolderitems		name=success path=/iportal/activePortal/private/filesfolders/filefolderlist.jsp
		name=dashboard path=/dashboard
/getjobdetails		name=success path=/iportal/activePortal/private/jobs/getjobdetails.jsp
/getnoticejobdetails		name=success path=/iportal/activePortal/private/jobs/getjobdetails.jsp
/getportletfolderitems		name=success path=/iportal/portlets/filefolderlist/filefolderlistportlet.jsp
/getrequesterjobdetails		name=success path=/iportal/activePortal/private/jobs/getrequesterjobdetails.jsp
/iPortalLogin	/iportal/login.jsp	name=landing path=/landing.jsp
		name=iPortalLoginForm path=/login.jsp
/login	/iportal/activePortal/private/login.jsp	name=loginform path=/iportal/activePortal/private/login.jsp
		name=success path=/dashboard/jsp/myfiles.jsp
		name=dashboard path=/dashboard
		name=ajclogin path=/ajclanding.jsp
		name=landing path=/landing.jsp
/logout		name=landing path=/login.jsp

*(continues)*

**Table 16-5** Visualization Platform actions (continued)

Action	Input JSP	Forward name path
/options	/iportal/activePortal /private/options /options.jsp	name=success path=/iportal/activePortal/private /options/options.jsp  name=saved path=/getfolderitems.do  name=dashboard path=/iportal/activePortal/private /options/options.jsp
/options/save	/iportal/activePortal /private/options /options.jsp	name=success path=/getfolderitems.do  name=saved path=/getfolderitems.do  name=success path=/iportal/activePortal/private /diagnosis/pingresponse.jsp
/ping		name=success path=/iportal/activePortal/private /diagnosis/pingresponse.jsp
/requestercanceljob		name=success path=/iportal/activePortal/private /jobs/requesterjoboperationstatus.jsp
/requesterdeletejob		name=success path=/iportal/activePortal/private /jobs/requesterjoboperationstatus.jsp
/requesterdeletejobschedule		name=success path=/iportal/activePortal/private /jobs/requesterjoboperationstatus.jsp
/searchfiles		name=success path="iportal/activePortal/private /filesfolders/search/filefolderlist.jsp
/selectjobnotices		name=success path=/iportal/activePortal/private /channels/channelnoticelist.jsp ?showBreadCrumb=false
/selectjobs		name=success path=/iportal/activePortal/private /jobs/selectjobs.jsp ?showBreadCrumb=false
/searchfiles		name=success path=/iportal/activePortal/private /filesfolders/search/filefolderlist.jsp

**Table 16-5** Visualization Platform actions (continued)

Action	Input JSP	Forward name path
/submitjob	/iportal/activePortal /private/newrequest /newrequest.jsp	name=createquery path=/query/ create.do name=query path=/query/submit.do name=success path=/iportal/activePortal/private/ newrequest/submitjobstatus.jsp name=viewbirt path=/iv name=viewessreport path=/servlet
/tableList	/iportal/activePortal /private/parameters /table /tableparameters.jsp	name=close path=/iportal/activePortal/private/ parameters/table/close.jsp name=tableParamList path=/iportal/activePortal/ private/parameters/table/ tableparameters.jsp
/uploadimage		name=success path=/iportal/activePortal/private/ customization/fileupload.jsp
/uploadlicense		name=success path=/iportal/admin/fileupload.js
/viewer /getsavedsearch		name=success path=/getfolderitems.do name=searchreport path=/iportal/activePortal/viewer/ searchreportpage.jsp name=requestsearch path=/iportal/activePortal/viewer/ requestsearch.jsp
/viewer/savesearch	/iportal/activePortal /viewer/savesearch.jsp	name=success path=/iportal/activePortal/viewer/ requestsearch.jsp name=browse path=/browsesfile.do

*(continues)*

**Table 16-5** Visualization Platform actions (continued)

Action	Input JSP	Forward name path
/waitforreport execution	/iportal/activePortal/private/newrequest/waitforexecution.jsp	name=success path=/iportal/activePortal/viewer/viewreport.jsp  name=fail path=/iportal/activePortal/viewer/closewindow.jsp

## Actuate application URIs reference

This section provides the detailed reference for Actuate application URIs. In the definitions, <context root> represents the name of your Actuate application context root, initially iportal. Table 16-6 lists the topics this chapter covers and the file names discussed in each topic. All pages are under the Actuate application context root.

**Table 16-6** Visualization Platform pages

Topic	Visualization Platform file
browse file page	browsefile.do
create folder page	createfolder.do
dashboard page	DashboardServlet
delete job page	iportal\activePortal\private\filesfolders\deletefilestatus.jsp
delete job page	deletejob.do
delete status page	deletejobnotice.do
detail page	
■ error detail page	iportal\activePortal\errors\detail.jsp
	getfilde details.do
■ file or folder detail page	iportal\activePortal\private\filesfolders\filedetail.jsp
■ request detail page	getjobdetails.do
	iportal\activePortal\private\jobs\getjobdetails.jsp
drop page	
■ file or folder drop page	deletefile.do
■ request drop page	canceljob.do

**Table 16-6** Visualization Platform pages

Topic	Visualization Platform file
error page	errors\error.jsp iportal\activePortal\private\common\errors\error.jsp
execute report page	executereport.do
index page	
■ file and folder index page	getfolderitems.do iportal\activePortal\private\filesfolders\filefolderlist.jsp
■ new request index page	executereport.do
■ requests index page	selectjobs.do iportal\activePortal\private\jobs\selectjobs.jsp
file and folder list page	getfolderitems.do iportal\activePortal\private\filesfolders\filefolderlist.jsp
login page	login.do iportal\activePortal\private\login.jsp
logout page	logout.do
myfiles page	dashboard\jsp\myfiles.jsp
output page	iportal\activePortal\private\newrequest\output.jsp
pending page	iportal\activePortal\private\newrequest\parameters.jsp
pending page	iportal\activePortal\private\jobs\pendingjob.jsp
ping page	ping.do iportal\activePortal\private\diagnosis\pingresponse.jsp
privileges page	iportal\activePortal\viewer\print.jsp
running page	iportal\activePortal\private\jobs\runningjob.jsp
scheduled job page	iportal\activePortal\private\jobs\scheduledjob.jsp
search folders page	searchfiles.do iportal\activePortal\private\filesfolders\search \filefolderlist.jsp
submit job page	submitjob.do iportal\activePortal\private\newrequest \submitjobstatus.jsp
viewer page for BIRT reports	IVServlet

## browse file page

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

**Name** <context root>\browsefile.do

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

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

---

## create folder page

Creates a folder in the current volume. Createfolder.do uses <context root>\iportal\activePortal\private\filesfolders\createfolder.jsp to create the new folder.

**Name** <context root>\createfolder.do

**Parameters** Table 16-7 lists and describes the parameters for the create folder page. The create folder page also uses the common URI parameters.

**Table 16-7** Parameters for create folder URI

URI parameter	Description
workingFolderID	The ID of the folder to contain the new folder. Specify either workingFolderID or workingFolderName.
workingFolderName	The name of the folder to contain the new folder. Specify either workingFolderID or workingFolderName.

**Used by** Not applicable.

---

## dashboard page

Provides the dashboard interface servlet for Visualization Platform.

**Name** <context root>\dashboard

**Used by** Not applicable.

---

## delete job page

Deletes the specified job, then redirects the page to a completion status page. Specify the name or the ID of the job to delete.

The default redirection JSP is  
`<context root>\iportal\activePortal\private\jobs\joboperationstatus.jsp.`

<b>Name</b>	<code>&lt;context root&gt;\deletejob.do</code>
<b>Parameters</b>	Table 16-8 lists and describes the parameters for the delete job page. The delete job page also uses the common URI parameters.

**Table 16-8** Parameters for delete job URI

URI parameter	Description
jobID	Unique request identifier.
jobName	The name of the job to delete. This parameter is ignored if jobID is also specified.
jobState	The state of the job to delete.
redirect	URI to which to redirect the job deletion page.

<b>Used by</b>	Not applicable.
----------------	-----------------

---

## delete status page

Deletes a job notice corresponding to a request. Specify the job notice by name or by ID.

<b>Name</b>	<code>&lt;context root&gt;\deletejobnotice.do</code>
	The default redirection action forwards to <code>&lt;context root&gt;\iportal\activePortal\private\jobs\joboperationstatus.jsp.</code>
<b>Parameters</b>	Table 16-9 lists and describes the parameters for the delete status page. The delete status page also uses the common URI parameters.

**Table 16-9** Parameters for delete status URI

URI parameter	Description
channelID	The unique identifier of the channel to delete the job notice from.

(continues)

## detail page

**Table 16-9** Parameters for delete status URI (continued)

URI parameter	Description
channelName	The name of the channel to delete the job notice from.
jobID	Unique request identifier.
jobName	The name of the job notice to delete. This parameter is ignored if jobID is also specified.
jobState	The state of the job to delete.
redirect	URL to which to redirect the delete status page.
userName	The name of the user to notify about the deleted job.

**Used by** Not applicable.

---

## detail page

Displays detailed information about volume objects. There are three detail pages:

<context root>\iportal\activePortal\errors  
<context root>\iportal\activePortal\filesfolders  
<context root>\iportal\activePortal\requests

## error detail page

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

**Name** <context root>\iportal\activePortal\errors\detail.jsp  
**Used by** iportal\activePortal\private\common\errors\error.jsp

## file or folder detail page

Displays detailed information about the selected viewable folder or file. Users request file or folder details by choosing the magnifying glass icon to the right of files or folders listed on the folder page or breadcrumb. Users can request another document or delete the current file or folder from the file or folder detail page. filedetail.jsp uses the HTML code in <context root>\iportal\activePortal\private\filesfolders\filedetailcontent.jsp to display the information. The file detail page lists information for a specified file, as shown in Figure 16-1.

**General**

Name: Revenue by Product Line  
 Type: ( GADGET )  
 Location: /Dashboards  
 Version  
 Name:  
 Version: 1  
 Description:  
 Size: 4.92 KB  
 Created: 11/26/2013 10:34 AM  
 Created by: Administrator  
 Shared: Yes

**Access Rights**

✓	✓	✓	✓	✓	✓ Secure	✓
Read	Write	Execute	Delete	Grant	Read	Visible

**Auto Archive**

This file will never expire.

**Figure 16-1** File detail page

**Name** <context root>\getfiledetails.do

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

**Parameters** Table 16-10 lists and describes the parameters for the file or folder detail page. The file or folder detail page also uses the common URI parameters.

**Table 16-10** Parameters for file or folder detail URI

URI parameter	Description
name	The full path name of the object for which to show details, if objectID is not specified.
objectID	The object's unique identifier.
version	The object's version number. The default is the latest version.

**Used by** Not applicable.

## request detail page

Lists detailed request information for a specified job, as shown in Figure 16-2.

## detail page

Job Status	
<input type="button" value="Cancel Request"/>	
Schedule (Running)	
Job Name:	Customer Order History
Owner:	Administrator
Priority:	1000
Submitted:	Dec 5, 2013 10:20:08 AM
Started:	Dec 5, 2013 10:20:08 AM
Run Job:	The job was scheduled immediately.
Event Name:	
Event Type:	No event
Event Parameter:	
Event Status:	
Report (Executable)	
Name:	 Customer Order History
Type:	Actuate BIRT Design
Version number:	1

**Figure 16-2** Request detail page

getjobdetails.jsp uses the HTML code in <context root>\iportal\activePortal\private\jobs\getjobdetailscontent.jsp to display the information.

**Name** <context root>\getjobdetails.do

<context root>\iportal\activePortal\private\jobs\getjobdetails.jsp

<context root>\getrequesterjobdetails.do

**Parameters** The request detail page uses the common URI parameters, as shown in Table 16-11.

**Table 16-11** Parameters for request detail URI

URI parameter	Description
jobID	The job's unique identifier
userName	The user that submitted the job
channelName	The channel to receive the request

**Used by** iportal\activePortal\private\jobs\completedjob.jsp  
iportal\activePortal\private\jobs\pendingjob.jsp  
iportal\activePortal\private\jobs\runningjob.jsp  
iportal\activePortal\private\jobs\scheduledjob.jsp

---

## drop page

Deletes one or more files or folders, or cancels a running job.

### file or folder drop page

Deletes the specified file or folder.

**Name** <context root>\deletefile.do

**Parameters** Table 16-12 lists and describes the parameters for the file or folder drop page. The file or folder drop page also uses the common URI parameters.

**Table 16-12** Parameters for file or folder drop URI

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

**Used by** Not applicable.

### request drop page

Cancels a running job.

**Name** <context root>\canceljob.do

**Parameters** Table 16-13 lists and describes the parameters for the request drop page. The request drop page also uses the common URI parameters.

**Table 16-13** Parameters for request drop URI

URI parameter	Description
jobID	The unique identifier of the object to delete.
jobName	The full path name of the object to delete. This parameter is ignored if jobID is also specified.
jobState	The state of the job to delete. processedaction_status uses jobState to build a link to pass to the list of scheduled and completed jobs.
redirect	URI to navigate to upon completion. The default redirect page is processedaction_status.

## error page

**Used by** Not applicable.

---

## error page

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

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

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

---

## execute report page

Submits a run report job request to the iHub. When executing a report job, a Cancel button appears after a specified wait time passes. Change the time by setting the EXECUTE\_REPORT\_WAIT\_TIME configuration parameter in the appropriate Visualization Platform configuration file. For reports that accept run-time parameters, you can set the parameter in the URL by adding an ampersand (&), the parameter name, and an equal (=) sign, followed by the parameter value in quotes.

**Name** <context root>\executereport.do

**Parameters** Table 16-14 lists and describes the parameters for the execute report page. The execute report page also uses the common URI parameters.

**Table 16-14** Parameters for execute report URI

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

**Table 16-14** Parameters for execute report URI (continued)

URI parameter	Description
__archiveBeforeDelete	Indicate whether to archive the output objects of the current request before deleting them, according to __archivePolicy's setting. Set this parameter to True to archive objects before deleting them. The default value is False. This parameter has no effect if __archivePolicy is set to Folder.
__archivePolicy	The archive policy to implement for the objects created as output for the current request. Values are folder, age, and date. Set to folder to use the archive policy that is already set for the folders to which the output is distributed. Set to age to delete objects older than a specific time period. Set to date to delete objects on a specific date.
__dateToDelete	The date on which to delete the output objects of the current request. Use only if __archivePolicy is set to Date. Set __dateToDelete to a date in a locale-specific format. The default format is mm/dd/yyyy.
__executableName	The name of the executable file for this request.
__format	Optional parameter that specifies the output format of the report. Visualization Platform appends a matching file extension to the outputName. Do not use a period in the value of this parameter. A period is inserted automatically before the file extension.
__headline	A descriptive tag line for a report document.
invokeSubmit	Appears on Channel Contents. Use the character string %20 to represent a space in the headline string.
__isnull	Controls whether the browser is redirected to the parameter screen or whether the report job is run immediately. If True, the report job is executed without displaying the parameters. If False, the parameters are displayed. False is the default.
__jobName	Sets the value of the named parameter to null. Use a parameter name as input.
__limit	The name of the job to execute.
	Indicate whether to limit the number of versions of the output files for the current request. Set __limit to Limit to limit the number of versions. Any other value means that the number of versions is unlimited.

(continues)

**Table 16-14** Parameters for execute report URI (continued)

URI parameter	Description
__limitNumber	The number of versions to which to limit the output files for the current request. Use only if __limit is set to Limit. __limitNumber can be any positive number.
__outputFolderType	Specifies the root of the output file name. Set to Absolute to use the full __outputName value starting from the volume's root. Set to Personal to use the __outputName value relative to the user's home folder.
__outputDocName	Specifies a name for the output file.
__overwrite	New to create a new version of this report document, or Old to overwrite an existing report document. New is the default.
__priority	Specifies the job submission priority. Values are High, Medium, and Low.
__priorityValue	Specifies a number ranging from 1 to 1000 and corresponding to the job submission priority. Only specify values allowed by your functionality level.
__progressive	Indicates whether to display the report document after it generates. If False, the report document displays after it generates. If True, the report document displays progressively, as it generates.
__recurringDay	Specifies the scheduled recurring day on which to run the report job. Applies only to scheduled report jobs.
__saveOutput	Indicates whether to write the output document to the volume. True saves the output in the volume, applying the document archiving and file creation parameters. False does not save the output.
__serverURL	Contains the URI that accesses the JSP engine, such as http://<iHub machine name>:8700.
__timeToDelete	Specifies a time at which to delete an archived report document. Applies only scheduled report jobs.
__users	Contains the name of the user to notify of this scheduled request. You can notify more than one user. This parameter is valid only for scheduled jobs.
__versionName	Contains a string value for the new version name of this report document. The value can include a date/time expression enclosed in braces, { }, to ensure a unique version name.

**Table 16-14** Parameters for execute report URI (continued)

URI parameter	Description
__volume	Contains a string value specifying the volume for this report.
__wait	If "wait", Visualization Platform waits for the report generation to be completed before displaying it. If "nowait", Visualization Platform displays the first page right away even if the report job is not completed.

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

```
http://127.0.0.1:8700/iportal/executereport.do?
 __requesttype=immediate&__executableName=%2fPublic%2fBIRT and
 BIRT Studio Examples%2fSales by Territory.rptdesign&
 userid=Administrator&__saveOutput=false&Territory="EMEA"&
 invokeSubmit=True
```

Set string parameters to an empty string by adding the parameter to the executereport.do URI with no value following the equal (=) sign. For example, the following line sets parameterA and parameterB to empty strings:

```
¶meterA=&ParameterB=
```

Dynamic filter parameters support a number of operators in the execute report URL. The syntax of a dynamic filter parameter is the same as other parameters, but the value is a function rather than a string value. Table 16-15 lists the valid filter operators and example of their syntax.

**Table 16-15** Filter operators

Operator	Description	Example
BETWEEN	Between an inclusive range	BETWEEN([COUNTRY], "AAA","BBB")
=	Equal to	([COUNTRY] = "AAA")
>	Greater than	([COUNTRY] > "AAA")
>=	Greater than or equal to	([COUNTRY] >= "AAA")
IN	Matches any value in a set of values	IN([COUNTRY],"China", "USA")
<	Less than	([COUNTRY] < "AAA")
<=	Less than or equal	([COUNTRY] <= "AAA")
LIKE	Search for a pattern	LIKE([COUNTRY],"AAA")
MATCH	Matches a pattern	MATCH([COUNTRY],"A")

(continues)

**Table 16-15** Filter operators (continued)

Operator	Description	Example
NOT(BETWEEN)	Not between an inclusive range	NOT(BETWEEN([COUNTRY], "AAA","BBB"))
<>	Not equal	([COUNTRY] <> "China")
NOT(IN)	Does not match any value in a set of values	NOT(IN([COUNTRY],"China", "USA"))
NOT(LIKE)	Search for values that do not match a pattern	NOT(LIKE([COUNTRY]), "AAA"))
NOT(MATCH)	Does not match a pattern	NOT(MATCH([COUNTRY], "A"))
NOT(ISNULL)	Is not null	NOT(ISNULL([COUNTRY]))
ISNULL	Is null	ISNULL([COUNTRY])

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

- dofframe
- inputfile
- jobType
- name
- selectTab

**Used by** Not applicable.

## index page

Provides the entry point and structure for the parts of Visualization Platform generated from multiple files.

## file and folder index page

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

The file and folder index page uses the browse file page to provide the reporting web page banner. filefolderlist.jsp uses the HTML code in <context root>\iportal

\activePortal\private\filesfolders\filefolderlistcontent.jsp to display files and folders data.

**Name** <context root>\getfolderitems.do

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

**Parameters** Table 16-16 lists and describes the parameters for file and folder index page. The file and folder index page also uses the common URI parameters.

**Table 16-16** Parameters for file and folder index URI

URI parameter	Description
startUpMessage	Specifies a message to appear when Visualization Platform calls this page.
subpage	Specifies the content of the page. Possible values are: <ul style="list-style-type: none"> <li>■ _list: include list</li> <li>■ _detail: include detail</li> </ul> Specifying any other value for subpage invokes the page not found page.

## new request index page

Provides the entry point and structure to support the submit job functionality.

**Name** <context root>\executereport.do

**Parameters** Table 16-17 describes the parameter for the new request index page. The new request index page also uses the common URI parameters.

**Table 16-17** Parameter for new request index URI

URI parameter	Description
homeFolder	The location of the My Documents folder

## requests index page

Provides the outermost structure for the active request functionality. The requests index page displays the side menu and banner elements, and the tabbed property sheets defined by tabs. selectjobs.jsp uses the HTML code in <context root>\iportal\activePortal\private\jobs\selectjobscontent.jsp to display request data.

**Name** <context root>\selectjobs.do

<context root>\iportal\activePortal\private\jobs\selectjobs.jsp

**Parameters** Table 16-18 lists and describes the parameters for the requests index page. The requests index page also uses the common URI parameters.

**Table 16-18** Parameters for request index URI

URI parameter	Description
applyFilter	Specifies whether to apply cbFail, cbSuccess, and filter to the current user session. applyFilter applies only to list pages, such as the completed jobs page.
cbFail	Specifies whether to list the failed jobs in the completed jobs page.
cbSuccess	Specifies whether to list the successful jobs in the completed jobs page.
channelName	Specifies the channel to which a job completion notice was sent. channelName applies only to the details page.
clearFilter	Clears the job name filter. clearFilter causes Visualization Platform to retrieve job names from session cookies and to ignore cbFail and cbSuccess. clearFilter applies only to list pages, such as the completed jobs page.
filter	Specifies the job name filter. filter applies only to list pages, such as the completed jobs page.
jobID	Specifies the unique job identifier. jobID applies only to the details page.
resetFilter	Resets all filters to their default values. The default filter values are no filtering for job name, and listing all completed jobs, whether failed or successful. resetFilter applies only to list pages, such as the completed jobs page.
subpage	Determines the content page. Possible values for subpage are: <ul style="list-style-type: none"> <li>■ _completed</li> <li>■ _detail</li> <li>■ _pending</li> <li>■ _running</li> <li>■ _scheduled</li> </ul> _completed is the default content page.
userName	Specifies the name of the user who received the completed job notice. userName applies only to the detail page.

**Used by** Not applicable.

---

## file and folder list page

Presents a list of objects that reside in the current working repository folder. Users request folder listings by choosing links on the reporting web page. The file and folder list page includes a filter section where users specify criteria for viewing report documents. For example, users select check boxes to indicate whether they want to view only the last version of a report document or to see report executable files and report documents.

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

The following are the sources that the file and folder list page uses to obtain the values for filters and the state of check boxes:

- URI parameters. See the following parameters section.
- Session attributes. Visualization Platform uses session cookies to store the values that a user specifies. If the user browses the Visualization Platform application, then returns to the listing page, the list page obtains the user's values from the session cookie if cookies are enabled. If the user chooses another folder, that folder becomes the working folder, and the list page applies the same values that applied to the previous folder.

Table 16-19 lists and describes the session attribute variables.

**Table 16-19** Session attribute variables

Session attribute	Description
AcFilesFoldersFilter	Contains the string specifying the files and folders viewing filter
AcFilesFoldersTypeFilter	Contains True if the user specified a filter, False otherwise

<b>Name</b>	<context root>\getfolderitems.do  <context root>\iportal\activePortal\private\filesfolders\filefolderlist.jsp
<b>Parameters</b>	Table 16-20 lists and describes the parameters for the file and folder list page. The file and folder list page also uses the common URI parameters.

**Table 16-20** Parameters for file and folder list URI

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

**Used by** Not applicable.

## login page

Displays the Visualization Platform login page for logging in to the Visualization Platform web application.

**Name** <context root>\login.do

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

**Parameters** Table 16-21 lists and describes the parameters for the login page. The login page also uses the common URI parameters.

**Table 16-21** Parameters for login page URI

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

**Used by** Not applicable.

---

## logout page

Ends the user's Visualization Platform session. The logout page gathers the user's session information, clears it, and returns the user to the login page.

**Name** <context root>\logout.do

**Parameters** Table 16-22 lists and describes the parameters for the logout page. The logout page also uses the common parameters.

**Table 16-22** Parameters for logout page URI

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

**Used by** Not applicable.

---

## myfiles page

Provides a container for the default dashboard display and the iHub Visualization Platform banner.

**Name** <context root>\dashboard\jsp\myfiles.jsp

**Used by** /iportal/activePortal/private/login.jsp

## output page

Specifies report executable output data, such as the report headline and output file name. The output page appears only for scheduled report job and Run and Save report job submissions. Users access Output by choosing Save As. An output page looks like Figure 16-3.

The screenshot shows the 'Save As' dialog box for an output page. The 'Save As' tab is selected. The 'Headline' field is empty. The 'Output Location' section shows 'Home folder' selected. The 'Document Name' field contains 'Top 5 Sales Performers'. The 'Version Name' field is empty. The 'Document Format' dropdown is set to 'RPTDOCUMENT'. The 'Notification' section has a checked checkbox for 'Send me an email notification with' and a dropdown menu showing 'No Attachment'. The 'If the File Already Exists' section has 'Create a new version' selected. The 'Copy permissions from' section shows 'Output folder' selected. At the bottom are 'Cancel', 'Back', 'Next', and a highlighted 'Finish' button.

**Figure 16-3** Output page

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

**Parameters** Table 16-23 lists and describes the parameters for the output page. The output page also uses the common URI parameters.

**Table 16-23** Parameters for output URI

URI parameter	Description
headline	Specifies the headline for the report.

**Table 16-23** Parameters for output URI

URI parameter	Description
IfExists	Specifies the file replacement policy. Values are Create and Replace. If ifExists is Create, Visualization Platform creates a new version. If ifExists is Replace, Visualization Platform replaces the existing version.
outputFolderType	Specifies the report output's folder type. Values are personal and absolute. If outputFolderType is personal, the output is placed in the user's personal folder. If outputFolderType is absolute, the user specifies the full path name for the output by either typing the path or using the Browse button.
outputName	Specifies the name of the output file.
versionName	Specifies the version name.

**Used by** `iportal\activePortal\private\newrequest\newrequestpage.jsp`  
`iportal\submitjob.do`

## pending page

Lists all jobs that are currently awaiting execution.

**Name** `<context root>\iportal\activePortal\private\jobs\pendingjob.jsp`  
**Parameters** The pending page uses the common URI parameters.  
**Used by** `iportal\activePortal\private\jobs\selectjobscontent.jsp`

## ping page

The ping page tests whether a specific component of the reporting environment is operational, and optionally retrieves other diagnostic information about the component. You can test the following components of the reporting environment:

- Visualization Platform itself
- The Encyclopedia service
- The Factory service
- The Integration service
- The Message Distribution service (MDS)
- The View service
- An Actuate open server driver

## ping page

If a component is not operational, iHub returns an error message. If a component is operational, the response depends on the ping page parameters. For example, you can request a simple time stamp that shows the time elapsed between the time that a component receives the request and the time that it returns a reply, as shown with the following URI:

```
http://seamore:8700/iportal/ping.do?destination=EE&mode=trace
```

which generates the following response:

```
18:03:23.100: MDS(seamore) received Ping message
18:03:23.100: MDS(seamore) forwarding Ping request to node seamore
18:03:23.100: EncycEngine(seamore) received Ping message
EncycEngine(seamore): Echoing 0 bytes of payload data
18:03:23.100: EncycEngine(seamore) replying to Ping message.
 Elapsed=0 ms
18:03:23.100: MDS(seamore) received Ping reply from node seamore.
 Roundtrip= 0 ms
18:03:23.100: MDS(seamore) replying to Ping message. Elapsed=0 ms
```

You also can request more detailed information. A ping request to the MDS has no security restrictions. For all other components, the request is subject to volume authentication. The user must be a volume administrator or a user with the Operator security role.

**Name** <context root>/ping.do

**Parameters** Table 16-24 lists and describes the parameters for the ping page. The ping page also uses the common URI parameters.

**Table 16-24** Parameters for ping URI

URI parameter	Description
action	Specifies the action to take at the destination. Valid values are: <ul style="list-style-type: none"><li>■ Echo—Echoes data specified by the payloadSize parameter. Echo is the default action.</li><li>■ ReadFile—Opens a specified volume file, reads its content, and closes the file. Destination must be EE, FS, or VS.</li><li>■ WriteFile—Creates a temporary file at a storage location, writes a specified number of bytes, closes the file, and deletes it. Destination must be EE or FS.</li><li>■ Connect—Connects to a data source.</li><li>■ If you do not specify a value, the destination component responds to the request without taking any other actions.</li></ul>

**Table 16-24** Parameters for ping URI (continued)

URI parameter	Description
destination	<p>The reporting environment component to test. Valid values are:</p> <ul style="list-style-type: none"> <li>■ AP (Visualization Platform)</li> <li>■ MDS (Message Distribution service)</li> <li>■ EE (Encyclopedia Engine)</li> <li>■ FS (Factory service)</li> <li>■ VS (View service)</li> <li>■ AIS (Actuate Integration service)</li> </ul> <p>AIS only supports the Echo action.</p> <p>Except when AP is specified as destination, Visualization Platform sends a ping request to the iHub and passes on the destination as the ping request's destination parameter.</p> <p>The default value is AP.</p>
filename	<p>When action=ReadFile, this parameter is required to indicate the volume file to read. If you ping an open server driver, filename specifies the executable file to prepare for execution.</p>
mode	<p>Specifies the level of detail in the ping response. Valid values are:</p> <ul style="list-style-type: none"> <li>■ Concise—Returns the elapsed time between a component's receipt of the request and the time the component sends a reply.</li> <li>■ Normal—Returns the names of components in the test path and the time stamps of the request entering and leaving each component. This is the default mode.</li> <li>■ Trace—Returns a time stamp at times when the request enters and leaves major subcomponents of the component being tested. For example, a request to a node running the Encyclopedia service can provide a time stamp for times when the request enters and leaves the process queue.</li> </ul> <p>A ping request at the trace level also can return diagnostic information other than timing. For example, a request to test writing a temporary file to a storage location can return the amount of free disk space in the storage location.</p>
partitionName	<p>Specifies the name of the storage location at which to create the temporary file. Used only if the value of action is WriteFile.</p>
payloadSize	<p>Length of payload string in number of characters that Visualization Platform should generate. Used only if the value of action is Echo.</p>

*(continues)*

## privileges page

**Table 16-24** Parameters for ping URI (continued)

URI parameter	Description
processID	Specifies the process ID of the Factory or View service to test. Used with the server parameter.
server	Specifies which instance of a Factory service or View service to test. Works with the processID parameter. To test all available instance of the Factory or View service, use an asterisk (*). If you do not specify server, the iHub load balancing mechanism allocates an available instance of the requested service to respond to the ping request.

**Used by** Not applicable.

---

## privileges page

Assigns privileges to a file or folder. Filefoldersprivilege.do uses the HTML code in <context root>\iportal\activePortal\private\filefolders\filefolderlist.jsp to set the privileges. The following URI displays the privilege page for the hotgraph report executable in the Training folder:

\iportal\filefoldersprivilege.do?name=\Training\hotgraph.rptdesign

**Name** <context root>filefoldersprivilege.do

**Parameters** Table 16-25 lists and describes the parameter for the privileges page. The privileges page also uses the common URI parameters.

**Table 16-25** Parameters for privileges URI

URI parameter	Description
name	File or folder name to set privileges for

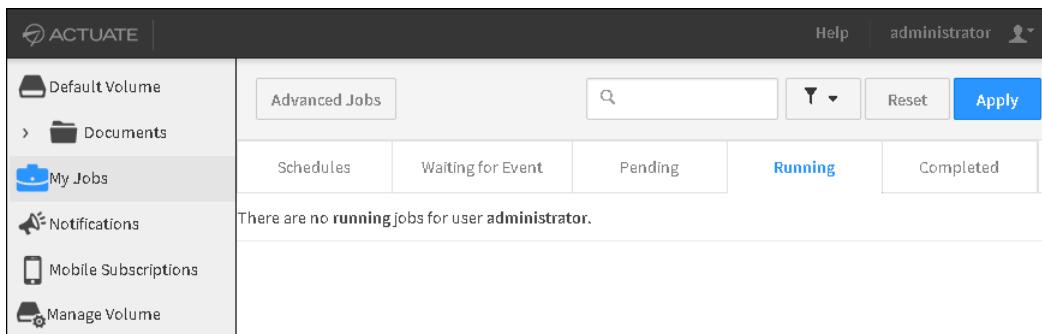
**Used by** iportal\activePortal\private\common\popupmenu.jsp  
iportal\activePortal\private\filefolders\filedetailcontent.jsp

---

## running page

Lists all jobs that were executing when the running page was last refreshed. The list is not live. To view the current list, the user must refresh the browser. Users access the running jobs list by choosing Running.

The running page looks like Figure 16-4.



**Figure 16-4** Running page

**Parameters** The running page uses the common URI parameters.

**Name** <context root>\iportal\activePortal\private\jobs\runningjob.jsp

**Used by** iportal\activePortal\private\jobs\selectjobscontent.jsp

## scheduled job page

Lists all jobs that activate at a specified date and time but are not yet active.

**Name** <context root>\iportal\activePortal\private\jobs\scheduledjob.jsp

**Parameters** The scheduled job page uses the common URI parameters.

**Used by** Not applicable.

## search folders page

Recursively searches from the current folder for files and folders whose names match the search string.

**Name** <context root>\searchfiles.do

**Parameters** Table 16-26 lists and describes the parameters for the search folders page. The search folders page also uses the common URI parameters.

**Table 16-26** Parameters for search folders URI

URI parameter	Description
folder	Folder name to start the search from. The default is the current location, as shown in the breadcrumb.
searchFilter	The name to search for. Expressions and wildcards are allowed. For more information about search expressions, see <i>Using Report Studio</i> .

## submit job page

For example, the following Visualization Platform URL searches the current folder and all subfolders for files or folders whose names begin with the string Cust:

```
http://127.0.0.1:8700/iportal/searchfiles.do?searchFilter=Cust*
```

**Used by** Not applicable.

---

## submit job page

Submits a scheduled report job for a report executable to the server. There is no user interface to the submit job page. submitjobstatus.jsp uses the HTML code in <context root>\iportal\activePortal\private\newrequest\submitjobstatuspage.jsp to display the new request information.

For reports that accept run-time parameters, set the parameter in the URL by adding an ampersand (&), the parameter name, and an equal (=) sign, followed by the parameter value in quotation marks.

**Name** <context root>\submitjob.do

<context root>\iportal\activePortal\private\newrequest\submitjobstatus.jsp

**Parameters** Table 16-27 lists and describes the parameters for the submit job page. The submit job page also uses the common URI parameters. All other parameters are passed to the report executable as report parameters. Report parameters are case-sensitive. Specify them exactly as defined in the report design.

**Table 16-27** Parameters for submit job URI

URI parameter	Description
__accessToGrant	Grants read or secure read privileges to those roles that have permission to view the report document. For users to view only the parts of the document matching an access control list (ACL), grant Secure Read access. Otherwise, grant Read access to enable users to view the whole document.  This parameter requires the __channels, __exclude, and invokeSubmit=true parameters, even if you use no value for them. Use the __exclude parameter with this parameter to exclude specific users from getting the privilege. Use the __channels parameter to grant read privileges to channels and notify them.
__ageDays	Used with __ageHours to determine how long output objects exist before they are deleted. Use only if __archivePolicy is set to age. __ageDays can be any positive number.

**Table 16-27** Parameters for submit job URI (continued)

URI parameter	Description
<code>__ageHours</code>	Use with <code>__ageDays</code> to determine how long output objects exist before they are deleted. Use only if <code>__archivePolicy</code> is set to age. <code>__ageHours</code> can be any positive number.
<code>__archiveBeforeDelete</code>	Indicates whether to archive the output objects of the request before deleting them, according to <code>__archivePolicy</code> 's setting. Set this parameter to True to archive objects before deleting them. The default value is False.
<code>__archivePolicy</code>	This parameter has no effect if <code>__archivePolicy</code> is set to folder.
<code>__archivePolicy</code>	The archive policy to implement for the objects created as output for the request. Values are folder, age, and date. Set this parameter to folder to use the archive policy already set for the folders to which the output is distributed, to age to delete objects older than a specific time period, or to date to delete objects on a specific date.
<code>__dateToDelete</code>	The date on which to delete the output objects of the current request. Use only if <code>__archivePolicy</code> is set to date. <code>__dateToDelete</code> must be a date in a locale-specific format. The default format is mm/dd/yyyy.
<code>__executableName</code>	The name of the executable file for this request.
<code>folderType</code>	Specifies the destination folder type for the report. Absolute indicates the repository root folder, /. Personal indicates the current user's home folder. Default is Personal.
<code>__headline</code>	A descriptive tag line for a report document. Appears on the Channel Contents page. Use the character string %20 to represent spaces in the headline string.
<code>__IfExists</code>	Indicates whether to overwrite an existing or create a new file, up to an optional limit. Values are: <ul style="list-style-type: none"> <li>■ <code>create</code>—creates a new output file.</li> <li>■ <code>create[n]</code>—creates a new output file up to n versions. For example, to create no more than seven versions, use <code>create7</code>.</li> <li>■ <code>replace</code>—overwrite any existing output.</li> </ul>
<code>invokeSubmit</code>	Controls whether the browser is redirected to the parameter screen or whether the report job is scheduled immediately. If True, the report job is scheduled without displaying the parameters. If False, the parameters are displayed. False is the default.
<code>__jobName</code>	The name for the job to submit.
<code>notificationSupported</code>	Specifies whether to notify users who have notification disabled. True sends notification and disregards user preferences. Default value is False.

(continues)

**Table 16-27** Parameters for submit job URI (continued)

URI parameter	Description
notify	Activates e-mail notification for the job.
__onceDate	Required for once schedules. Specify the date on which to run the report job, for report jobs with __scheduleType of once. Must be in the appropriate format for your locale, such as mm/dd/yyyy for the U.S. locale. The current date is the default.
__onceTime	Required for once schedules. Specify the time at which to run the report job, for report jobs with __scheduleType of once. Must be in the appropriate format for your locale, such as "hh:mm a" for the U.S. locale. The current time is the default.
__outputName	Specifies a name for the report output document.
outputName	Specifies a name for the report output document for the e-mail notification.
outputFormat	Optional parameter that appends a file extension to the outputName. Do not use a period in the value of this parameter, a period is inserted automatically before the file extension.
postback	Forces the browser not to display parameters. Set to False to display parameters. Do not set postback to True with invokeSubmit also set to True.
__priority	Specifies the job submission priority. Values are a number from 1 to 1000, High (800), Medium (500), and Low (200). Do not use with __priorityValue.
__priorityValue	Specifies a number corresponding to the job submission priority. Do not use with __priority.
__progressive	Indicate whether to display the report document after it generates. If False, the report document displays after it generates. If True, the report document displays progressively, as it generates. Applies only to run report jobs.
__recurringDay	Specifies the scheduled recurring day on which to run the report job. Applies only to scheduled report jobs.
__recurringTime	Required for recurring schedules. Specify the time at which to run the report job. Set only if report jobs __scheduleType is recurring. Must be in the appropriate format for your locale, such as hh:mm:ss for the U.S. (enu) locale.
__redirect	Specifies a relative or absolute URI to go to after do_executereport submits the report job. The default is Submittedjob_Status.

**Table 16-27** Parameters for submit job URI (continued)

URI parameter	Description
__schedulePeriod	Required for recurring schedules. Specify how often to run the report job, and on which days. Choose a day of the week. __schedulePeriod values are Every Day, Weekdays, Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays, First Day of the Month, Last Day of the Month. All values are case-sensitive.
__scheduleType	Every Day or Weekdays. Set only if __scheduleType is recurring.
__serverURL	Specify the type of schedule: immediate, once, or recurring. Immediate is the default.
__timeToDelete	Contains the URI that accesses the JSP engine, such as http://Services:8700.
__versionName	Specifies a time at which to delete an archived report document. Applies only to scheduled report jobs.
__volume	Contains a string value for the new version name of the job's report document output. The value can include a date/time expression enclosed in braces, { }, to ensure a unique version name.

The submit job page also accepts dynamic filter parameters for BIRT Reports in the URL, but the value of the parameter must form a complete expression, such as &Territory=([Territory] = "EMEA"). For a complete list of supported operators, refer to Table 16-15.

For example, the following URL schedules the Sales By Territory.rptdesign report to run once on September 16, 2010 with the Territory run-time parameter set to Japan:

```
http://127.0.0.1:8700/iportal/
 submitjob.do?__requesttype=scheduled&__executableName=%2fPublic
 %2fBIRT%20and%20BIRT%20Report%20Studio%20Examples%2fSales%20by%
 20Territory%2erptdesign%3b1&userid=administrator&__scheduleType
 =once&__onceDate=09/16/2010&__onceTime=1:55
 pm&Territory="Japan"&invokeSubmit=True
```

**Used by** iportal\activePortal\private\filefolders\filefolderlistcontent.jsp  
 iportal\activePortal\private\newrequest\newrequestpage.jsp

submit job page

# 17

## Actuate Report Studio URLs

This chapter contains the following topics:

- Accessing Report Studio using a URI
- Using the Report Studio servlet
- Using the Report Studio URLs

---

## Accessing Report Studio using a URI

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

You invoke the Report Studio servlet using the following syntax:

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

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

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

---

## Using the Report Studio servlet

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

**Name** com.actuate.erni.servlet.ERNIViewerServlet

Invoke the Report Studio servlet as:

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

**URI parameters** The Report Studio servlet requires repository parameters in order to operate. Table 17-1 lists and describes the URI parameters for the Report Studio servlet.

**Table 17-1** Report Studio URI parameters

URI parameter	Description
repositoryType	The repository type. Use Enterprise for a volume.
serverURL	The URL of an iHub machine.
volume	The name of a volume that is managed by the iHub URL to which you connect.

**Table 17-1** Report Studio URI parameters

URI parameter	Description
_vp	The name of a server configured in VolumeProfile.xml. Report Studio uses the volume information in a VolumeProfile entry except when a volume parameter specifies a different one.

## Using the Report Studio URLs

You can log in to Report Studio by typing a URL in a web browser's address field. After you type a URL and press Enter, the login page appears. What happens after you log in depends on which URL you use. In addition to the initial Report Studio page, you can open Report Studio with:

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

In the example URLs in the following topics, special characters are represented by codes, as shown in Table 17-2.

**Table 17-2** Codes for special characters in URLs

Character	Code
Colon (:)	%3a
Slash (/)	%2f
Period (.)	%2e
Space ( )	%20

### How to log in

To access the Report Studio login page, use a URL like the one in the following example:

`http://urup.domain.com:8700/iportal/wr?_vp=Default%20Volume`

- urup.domain.com:8700 is the fully qualified domain name of the computer on which iHub Visualization Platform client is installed and the port you use to access iHub Visualization Platform client.
- iportal is the context root for iHub Visualization Platform client.
- wr is the default context root for accessing Report Studio.

- ? indicates the beginning of a parameter that indicates where to access Report Studio files.
- \_\_vp=Default%20Volume specifies the default volume profile in which to work.

To log in directly to the Report Studio design environment, add userid and password parameters to the URL, as shown in the following example:

```
http://urup.domain.com:8700/iportal/wr?userid=MyUser
 &password=MyPwd&__vp=Default%20Volume
```

Sending security information such as the user name and password in a URL is not a recommended approach.

#### **How to open Report Studio and load an existing report design**

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

```
http://urup.domain.com:8700/iportal/wr?__report=
 %2fApplications%2fBIRT%20Sample%20App%2fCustomer%20Order
 %20History.rptdesign&pCountry=USA
```

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

# 18

## Actuate BIRT Viewer URIs

This chapter contains the topic About the BIRT Viewer servlet.

## About the BIRT Viewer servlet

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

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

- application server is the name of the machine hosting the application server.
- port is the port on which the application server listens for requests.
- context root is the iHub Visualization Platform client context root.
- iv is the name to which the servlet is mapped in the web application's web.xml file. A typical location for web.xml is <context root>\WEB-INF\web.xml.

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

## Using open source BIRT URIs in Actuate BIRT Viewer

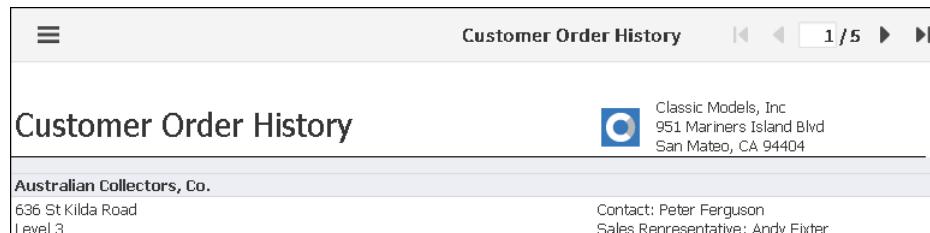
Actuate BIRT Viewer fully supports the URIs for the open source BIRT Viewer. After migrating from open source BIRT to Actuate BIRT, you can use the same URIs in the Actuate BIRT Viewer that you used in open source BIRT.

## Accessing the BIRT Viewer using a URI

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

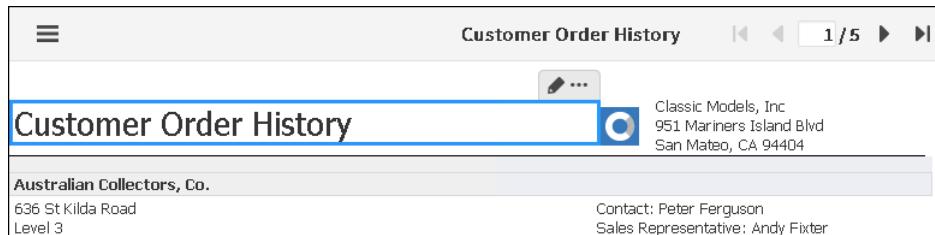
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

The BIRT Viewer provides navigation toolbar options, as shown in Figure 18-1.



**Figure 18-1** BIRT Viewer

The Interactive Viewer displays the report with toolbar options to navigate the report, a menu to perform additional tasks, and provides context menus to edit and format report elements, as shown in Figure 18-2.



**Figure 18-2** BIRT Interactive Viewer

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

Name	com.actuate.iv.servlet.IVServlet
URI parameters	Table 18-1 lists and describes the URI parameters for the BIRT Viewer servlet.

**Table 18-1** IVServlet URI parameters

URI parameter	Description
__bookmark	Name of the element of a report to display instead of the whole report file.
__floatingfooter	Boolean value to add a margin under the footer.
__format	A format for the displayed report: <ul style="list-style-type: none"><li>■ pdf: Adobe PDF</li><li>■ xls: Microsoft Excel</li><li>■ doc: Microsoft Word</li><li>■ ppt: Microsoft PowerPoint</li><li>■ ps: PostScript</li><li>■ html: HTML</li><li>■ flashchartsxml: Used to display a fusion chart</li><li>■ flashgadgetsxml: Used to display Flash gadgets in a fusion chart</li><li>■ reportlet: Used with __bookmark to show a particular part or element of the report</li></ul>
__from_page_range	The page range of a report to display.

(continues)

**Table 18-1** IVServlet URI parameters (continued)

URI parameter	Description
__from_page_style	The page style to use for a report in pdf or ps formats: <ul style="list-style-type: none"><li>■ auto: The page size and content size remain the same.</li><li>■ actualSize: Change the page size to fit the content.</li><li>■ fitToWholePage: Change the content size to fit the page size.</li></ul> Used with the __format parameter.
__imageid	Identifier of the report file image to display.
__instanceid	Identifier of the report file to display.
__launchinv	A Boolean value that enables interactivity.
__locale	Code for a locale. For example FR_fr specifies the French language and the country, France.
__page	The number of a page to render.
__report	Name of the report file to display.
__rtl	Boolean value that specifies right-to-left orientation for the report.
serverURL	The URL that accesses iHub, such as <a href="http://ESL02835:8000">http://ESL02835:8000</a> .
userid	The user's identifier, required to log in to the iHub.
volume	A string value specifying the volume for this report.

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

```
127.0.0.1:8700/iportal/iv?__report=/Home/administrator
/Top Sales Performers.rptdocument;1
```

# Part Six

---

**Using Actuate security**



# 19

## Using Visualization Platform security

This chapter contains the following topics:

- About Actuate Visualization Platform security
- Protecting corporate data
- Understanding the authentication process
- Creating a custom security adapter
- Creating an upload security adapter

---

## About Actuate Visualization Platform security

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

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

The types of security you can provide for Visualization Platform are:

- Default user authentication. Use the default Visualization Platform and BIRT iHub facilities to ensure that users access only those reports and other volume items for which they have permission.
- User authentication using the iPortal Security Extension (IPSE). Use IPSE to customize and control the user login and authentication process. For details about implementing custom user authentication, see “Creating a custom security adapter,” later in this chapter.

---

## Protecting corporate data

iHub provides a structured content generation solution for web applications. Deploying Actuate applications developed for the internet, such as Visualization Platform, requires planning for network security.

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

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

### Protecting corporate data using firewalls

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

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

## Protecting corporate data using proxy servers

Proxy servers, specialized web servers or hardware that operate on or behind a firewall, improve efficient use of network bandwidth and offer enhanced network security. For more information about proxy servers and Visualization Platform, see *Managing Volumes and Users*.

---

## Understanding the authentication process

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

- A user or client makes a request by choosing a link on a Visualization Platform page or by typing a Visualization Platform URI in a web browser. The Visualization Platform application processes the request.
- Visualization Platform checks the URI for the forceLogin parameter. If the forceLogin parameter is set to “true” in the URI, the application activates the Visualization Platform Login page, even if the user has already logged in. If forceLogin is set to “false” or does not appear, the request process continues. For details about the forceLogin URI parameter, see Chapter 16, “Actuate application URIs.”
- Visualization Platform authenticates the user for the cluster. If the login information is invalid, the login screen appears in the browser.

If a custom security adapter parameter is set in the web.xml file, Visualization Platform attempts to load the custom security adapter class. If the class loads successfully, the following steps occur:

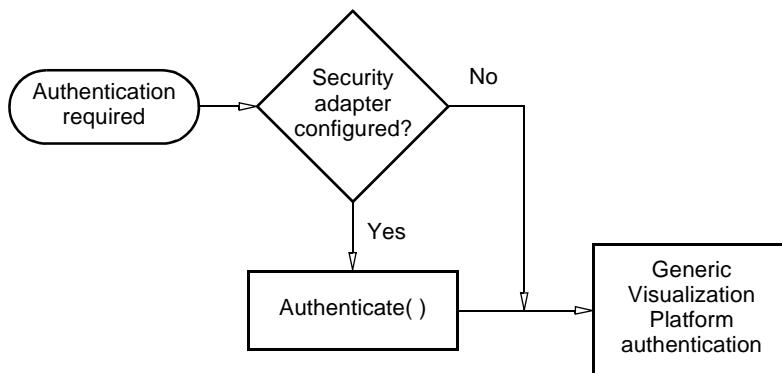
- Visualization Platform calls the custom security adapter’s authenticate( ) method with the parameters that the browser sent.
- The authenticate( ) method performs the custom validation.
- Visualization Platform calls the required getUsername( ), getPassword( ), and getVolumeProfile( ) methods to retrieve the user information needed by iHub.
- Optionally, Visualization Platform calls the getExtendedCredentials( ) method. If this method returns null, there are no extended credentials to send to iHub.

- Visualization Platform now has all the information that it requires for connecting to iHub. Visualization Platform creates the necessary SOAP message for connecting to iHub and sends a login request.

iHub Visualization Platform client uses the default Volume Profile setting if the server, volume, or volume profile if it is not registered as a service provider with the cluster.

## Creating a custom security adapter

The Visualization Platform security adapter enables other applications to authenticate users and log in to the Visualization Platform application, for example, by using a URL. A custom security adapter can define alternate authentication requirements. In this way, a Visualization Platform security adapter establishes an additional layer of logic to the existing Visualization Platform authentication, as shown in Figure 19-1.



**Figure 19-1** Visualization Platform authentication system

A user cannot update their password from Visualization Platform if a custom security adapter class is set. In this way, Visualization Platform prevents conflicts between the user's current password and the security system that is used to verify passwords.

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

- Ensure that your application can access the IPSE Java classes.
- Create a Java class that implements the custom security adapter class for IPSE.
- Deploy the Custom Security Adapter to iHub Visualization Platform client.

## Accessing the IPSE Java classes

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

Specifically, the JRE needs to access the following jars:

- <context root>\WEB-INF\lib\com.actuate.iportal.jar
- <context root>\WEB-INF\lib\org.apache.xerces\_<version>.jar
- <context root>\WEB-INF\lib\com.actuate.webcommon.jar
- <iPortal installation directory>\lib\servlet-api.jar
- <iPortal installation directory>\lib\jsp-api.jar

## Creating a custom security adapter class

Extend the iPortal Security Adapter class to customize authentication. The iPortal Security Extension requires access to the following libraries:

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

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

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

The login module of Visualization Platform calls methods in the custom security class to perform authentication and to retrieve login credentials to pass to iHub. The authenticate( ) method returns a boolean value to indicate whether the login credentials provided are acceptable. The getter methods return the credentials that iHub requires. Each user name and password must correspond to an authentic user account on the volume configured by the volume profile. If a volume profile isn't set by the security adapter, authentication uses the default volume profile configuration. For example, to support a URL that authenticates using a single parameter, code, override authenticate( ) to retrieve the parameter from the HttpServletRequest and set the user name, password, and volumeProfile as in the following class:

```

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

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

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

```

If there is a user "user1" with the password "user1" on the volume configured by the volume profile "CustomAccess," a valid URL that authenticates user1 using this security adapter is as follows:

<http://localhost:8700/iportal/getfolderitems.do?code=12345>

## Deploying a custom security adapter

To deploy a custom security adapter, the iHub Visualization Platform client application must have access to the class compressed into a JAR file. To meet this requirement, compile the class, compress it into a JAR, and move it into the <context root>\WEB-INF\lib directory for your iHub Visualization Platform client application. Then, add the name of the class as the value for the SECURITY\_ADAPTER\_CLASS parameter in <context root>\WEB-INF\web.xml. Finally, restart the application service running iHub Visualization Platform client to activate this change.

## How to deploy a custom security adapter to iHub Visualization Platform client

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

```
javac SecurityCode.java
```

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

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

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

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

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

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

- 5 Navigate to the parameter name SECURITY\_ADAPTER\_CLASS.

- 6 Change the param-value parameter of the SECURITY\_ADAPTER\_CLASS to the fully qualified class name for the security adapter class. Use an entry similar to this one:

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

- 7 Save and close web.xml.

- 8 Restart the application server running iHub Visualization Platform client. For the default installation, restart the Actuate Apache Tomcat for Visualization Platform Service.

## Understanding the security adapter class

To implement a custom security adapter, create a class that extends com.actuate.iportal.security.iPortalSecurityAdapter. This class contains the following methods.

### authenticate( )

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

**Parameter** **request**

The request parameter sent from the iHub Visualization Platform client web application.

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

**Returns** True for successful credential evaluation and False otherwise.

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

## **getExtendedCredentials( )**

**Syntax** byte[] getExtendedCredentials( )

**Description** Retrieves the current user's extended security credentials.

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

## **getPassword( )**

**Syntax** String getPassword( )

**Description** Required method that retrieves the current user's password. The Login module calls getPassword() and uses the password to establish a connection to the iHub and to access the volume.

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

## **getRepositoryType( )**

**Syntax** String getServerUrl( )

**Description** Retrieves the repository type. The Login module calls this method to check the repository type. Alternatively, provide isEnterprise().

**Returns** A string that indicates the repository type. The repository type for iHub is enterprise.

## **getRunAsUser( )**

**Syntax** String getRunAsUser( )

**Description** Retrieves the runAs setting if the runAs is enabled. The Login module calls this method to retrieve the user name used for a run as operation.

**Returns** A string containing the user name that corresponds to the runAs user setting.

## **getServerUrl( )**

**Syntax** String getServerUrl( )

**Description** Retrieves the URL of the server to which the current user connects. The Login module calls getServerUrl( ).

**Returns** A string containing the URL for the iHub currently connected.

## **getUserHomeFolder( )**

**Syntax** String getUserHomeFolder( )

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

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

## **getUserName( )**

**Syntax** String getUserName( )

**Description** Required method that retrieves the current user's login name. The Login module calls getUserName( ) to establish a connection to the iHub and to access the volume.

**Returns** A string containing the user name that the iHub recognizes.

## **getVolume( )**

**Syntax** String getVolume( )

**Description** Retrieves the volume to which the current user connects. The Login module calls getVolume( ) to retrieve the name of the volume to which the user wishes to connect.

**Returns** A string containing the domain and volume name for the volume to which the user connects to through the iHub. If null, the iHub connects to the default volume, read from the DEFAULT\_VOLUME parameter in the Visualization Platform web.xml file.

## **getVolumeProfile( )**

**Syntax** String getVolumeProfile( )

**Description** Required method that retrieves the volume profile to which the current user connects. The Login module calls getVolumeProfile( ) to retrieve the name of the volume profile to which the user wishes to connect.

**Returns** A string containing the server profile name for the volume to which the user connects through the iHub.

## **isEnterprise( )**

**Syntax** boolean isEnterprise( )

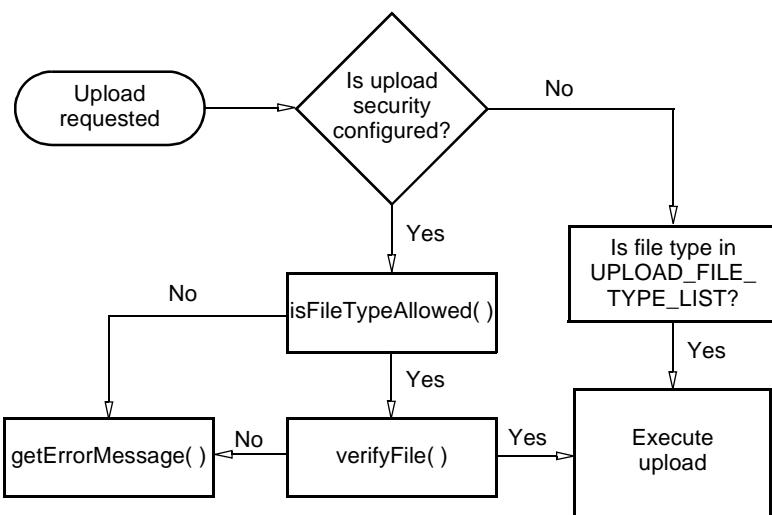
**Description** Evaluates whether the user connects to a volume. The Login module calls isEnterprise( ) to determine whether to use a volume repository.

**Returns** True.

## Creating an upload security adapter

The default security for iHub Visualization Platform client upload functionality checks a file's type against the value of the UPLOAD\_FILE\_TYPE\_LIST parameter in web.xml. The iHub Visualization Platform client upload security adapter provides additional external verification features for the file upload feature using a Java interface, com.actuate.iportal.security.IUploadSecurityAdapter.

Visualization Platform upload security adapter establishes an additional layer of logic to the existing Visualization Platform authentication, as shown in Figure 19-2.



**Figure 19-2** Visualization Platform upload security system

If an upload security adapter is configured, iHub Visualization Platform client calls isFileTypeAllowed to check whether the file's file type is allowed. If so, then it calls verifyFile to perform any additional verification steps. If either isFileTypeAllowed or verifyFile returns false, iHub Visualization Platform client displays an error message supplied by getErrorMessage, or a generic message if getErrorMessage returns null.

To create an upload security adapter, perform the following steps:

- Ensure that your application can access the necessary Java classes.
- Create a Java class that implements the upload security adapter interface.
- Deploy the upload security adapter class to iHub Visualization Platform client.

## Accessing the necessary Java classes

The Visualization Platform library, com.actuate.iportal.jar, contains the security extension Java classes. This library is located in the lib subdirectory of the Visualization Platform installation. The upload security adapter interface, com.actuate.iportal.security.IUploadSecurityAdapter, in this library provides the framework for additional upload security. A valid upload security adapter implements this interface.

Specifically, the JRE needs to access the following JARs:

- <context root>\WEB-INF\lib\com.actuate.iportal.jar
- <context root>\WEB-INF\lib\org.apache.xerces\_2.9.0.v201005080400.jar
- <iHub Visualization Platform client installation directory>\lib\servlet-api.jar
- <iHub Visualization Platform client installation directory>\lib\jsp-api.jar

## Creating a custom security adapter class

Implement the upload security adapter interface to customize file verification. The upload security adapter requires access to the following libraries:

- javax.servlet.http.HttpServletRequest
- javax.servlet.ServletContext
- com.actuate.iportal.security

To process a secure upload request from iHub Visualization Platform client using an upload security adapter, the following methods are required:

- getErrorMessage()
- isFileTypeAllowed()
- verifyFile()

For example, to prevent any file type except plain text (.txt) from being uploaded, implement txt as the only valid file type for isFileTypeAllowed, as in the following class:

```
package com.actuate.iportal.security;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;

public class SecureUpload implements IUploadSecurityAdapter {

 public boolean isFileTypeAllowed(HttpServletRequest request,
 String fileType){
 if (fileType == null) return false;
 if (fileType.toLowerCase().trim().equals("txt")) return
 true;
 }
}
```

```

 else return false;
 }

 public boolean verifyFile(HttpServletRequest request, String
 fileName, String dstFolder){
 return true;
 }

 public String getErrorMessage(HttpServletRequest request) {
 String message = "Only plain text (.txt) files are
 permitted.";
 return message;
 }
}

```

When the upload security adapter requires file validation, iHub Visualization Platform client copies the file temporarily into the directory specified by TEMP\_FOLDER\_LOCATION parameter in web.xml.

## Deploying an upload security adapter

To deploy an upload security adapter, the iHub Visualization Platform client application must have access to the class compressed into a JAR file. To meet this requirement, compile the class, compress it into a JAR, and move it into the <context root>\WEB-INF\lib directory for your iHub Visualization Platform client application. Then, add the name of the class as the value for the UPLOAD\_SECURITY\_ADAPTER parameter in <context root>\WEB-INF\web.xml. Finally, restart the application service running iHub Visualization Platform client to activate this change.

### How to deploy an upload security adapter to iHub Visualization Platform client

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

```
javac SecureUpload.java
```

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

```
jar cvf SecureUpload.jar SecureUpload.class
```

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

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

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

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

- 5 Navigate to the parameter name UPLOAD\_SECURITY\_ADAPTER.

- 6 Change the param-value parameter of the UPLOAD\_SECURITY\_ADAPTER to the fully qualified class name for the upload security adapter class. Use an entry similar to this one:

```
<param-name>UPLOAD_SECURITY_ADAPTER</param-name>
<param-value>SecureUpload</param-value>
```

- 7 Save and close web.xml.
- 8 Restart the application server running iHub Visualization Platform client. For the default installation, restart the Actuate 11 Apache Tomcat for Visualization Platform Service.

## Understanding the upload security adapter interface

To implement a custom upload security adapter, create a class that implements the com.actuate.iportal.security.IUploadSecurityAdapter interface. This interface defines the following methods.

### getErrorMessage( )

**Syntax** String getErrorMessage( javax.servlet.http.HttpServletRequest request)

**Parameter** **request**

The request parameter sent from the iHub Visualization Platform client web application.

**Description** A method that returns a custom error string when either isFileTypeAllowed or verifyFile returns False.

**Returns** String. An error message. If null, iHub Visualization Platform client displays a generic default error message.

### isFileTypeAllowed( )

**Syntax** boolean isFileTypeAllowed( javax.servlet.http.HttpServletRequest request, string fileType)

**Parameters** **request**

The request parameter sent from the iHub Visualization Platform client web application.

**fileType**

String. The type of the upload file, as determined by file extension.

**Description** A required method used to do additional validation of the upload file.

**Returns** Boolean. True for an allowed file type and False otherwise.

## **verifyFile( )**

**Syntax** boolean verifyFile( javax.servlet.http.HttpServletRequest request, string filePath, string dstPath)

**Parameters** **request**

The request parameter sent from the iHub Visualization Platform client web application.

**filePath**

String. The path of the file stored on iHub Visualization Platform client. The default location is the directory specified by TEMP\_FOLDER\_LOCATION parameter in web.xml.

**dstPath**

String. The repository path to which the upload sends the file.

**Description** A required method used to do additional validation of the upload file.

**Returns** Boolean. True for successful file validation and False otherwise.

# 20

## Using Java Report Server Security Extension

This chapter consists of the following topics:

- About the Java Report Server Security Extension
- Implementing the Java RSSE interface
- About installing a Java RSSE application
- Using page-level security
- Report Server Security Extension (RSSE) API operations
- Report Server Security Extension (RSSE) API data types

---

## About the Java Report Server Security Extension

BIRT iHub System provides a SOAP-based API that supports running a Report Server Security Extension (RSSE) application as a web service. Using the Java RSSE framework, a developer can create an application that provides one of the following security features:

- External authentication  
Authenticates a user password using an interface to an external security system such as Lightweight Directory Access Protocol (LDAP) or Microsoft Active Directory. Users, roles, notification groups, access control lists (ACLs), and other information remain on the volume.
- External registration  
Manages users, roles, notification groups, access control lists (ACLs), and other information using an interface to an external security system such as Lightweight Directory Access Protocol (LDAP) or Microsoft Active Directory. The volume no longer manages this information.
- Page-level security  
Controls user access to sensitive information in a BIRT report by implementing page-level security. A page-level security application requires a BIRT Page Level Security option license.

The following sections describe how to build, install, and customize these Java RSSE security applications in the Actuate Information Delivery API development environment.

---

## Implementing the Java RSSE interface

BIRT iHub Integration Technology provides sample applications that demonstrate Java RSSE interface implementations. In the installation, each sample application is located in a separate subdirectory under the Java Report Server Security Extension directory.

Each sample application provides the following resources:

- The reference implementation of the RSSE interface.  
Table 20-1 lists the package for each sample application.

**Table 20-1** Sample application packages

RSSE application	Package
LDAP authentication	com.actuate11.rsse.authenticationSample
LDAP external registration	com.actuate11.rsse.ldapSample

**Table 20-1** Sample application packages

RSSE application	Package
Page Security	com.actuate11.rsse.aclSample

- The file, lib/rsse.jar, contains the RSSE interface and related classes. The documentation for the package, com.actuate11.rsse.interfaces, is in the Java Report Server Security Extension/docs folder. To implement a class for the RSSE interface, refer to this API reference.
- The object renderer package, com.actuate11.rsse.or, contains a set of helper classes for logging RSSE objects to a file. The package uses the open source logging tool, Apache log4j. Using the Apache log4j API, a developer can write log statements in the application code, then configure the logging level through a property file.  
To configure the logging level for a Java RSSE application, modify the property, log4j.logger.com.actuate11.rsse, in the file, log.properties. The file, log.properties, is in the application package.

Apache log4j supports logging at the following levels:

- FATAL describes a severe error event that typically causes the application to abort.
- ERROR describes an error event that typically allows the application to continue running.
- WARN provides an alert to a potential problem.
- INFO provides a general message that describes the application's progress.
- DEBUG provides information on an application event that is useful for debugging.
- ALL turns on all logging options.
- OFF turns off logging.

For more information about the Apache Logging Services Project and the log4j tool, see <http://logging.apache.org/>.

---

## About installing a Java RSSE application

To set up and run a Java RSSE application, perform the following tasks:

- Build the Java RSSE application.
- Install the Java RSSE application on a volume.
- Enable a web service to use the Java RSSE application.

## How to build a Java RSSE sample application

Install and use the Apache Ant tool to build a Java RSSE sample application. Go to the Apache Ant Project web site at <http://ant.apache.org/> to obtain the software and installation instructions.

In the BIRT iHub Integration Technology installation, each sample application subdirectory contains a file, build.xml. Using the project settings specified in the file, build.xml, Ant performs the following operations:

- Compiles the Java RSSE application source files
  - Creates a lib directory
  - Archives the compiled classes in a JAR file in the lib directory
- Table 20-2 lists the archive file generated for each Java RSSE sample application.

**Table 20-2** Archive files that are generated for the Java RSSE sample applications

RSSE application	Archive file
LDAP authentication	rsseAuthenticate.jar
LDAP external registration	rsseLdap.jar
Page Security	rsseAcl.jar

To build a sample application using the Ant tool, navigate to the application directory. At the command line, type:

```
ant
```

## Installing a Java RSSE application

A single SOAP-based RSSE application provides the security implementation for an entire iHub cluster. The RSSE application runs as a web service in the BIRT iHub servlet container.

The default location for an RSSE web service application is \$SERVER\_HOME /web/webapps/acrsse. This location must be accessible to all nodes in the cluster.

### How to install a Java RSSE application

Install a Java RSSE application to run on BIRT iHub by performing the following tasks:

- 1 Make a copy of the \$SERVER\_HOME/web/webapps/acrsse directory. For example, copy the directory to the following location:

```
$SERVER_HOME/web/webapps/myacrsse
```

- 2 Copy the application archive file to the lib directory of the BIRT iHub servlet container in the following location:

```
$SERVER_HOME/web/webapps/myacrsse/WEB-INF/lib
```

- 3 Extract the file, class.properties, from the application archive file, to the following location:

```
$SERVER_HOME /web/webapps/myacrsse/WEB-INF
/classes/com/actuate11/rsse/wsdl
```

If necessary, create the subdirectories, /com/actuate11/rsse/wsdl, manually or use the archive extraction tool to create the subdirectories when extracting the class.properties file.

- 4 Using a source code editor, open the class.properties file and change its single line of code to reference the main class of the application in the archive file:

```
class=com.actuate11.rsse.mySampleApp.SampleRSSE
```

## Configuring and deploying an LDAP configuration file

To use a Java RSSE sample application that utilizes LDAP for external user authentication or registration, configure and deploy an LDAP configuration file to the BIRT iHub etc directory before enabling the web service on the volume.

### How to configure and deploy an LDAP configuration file for external authentication

To configure and deploy the LDAP configuration file for external authentication perform the following operations:

- 1 Using a source code editor, create the LDAP configuration file by typing the following code, substituting the values appropriate for the LDAP server installation such as:

- Name of the LDAP server
- Port number where the LDAP server listens
- UserBaseDN, including the attributes for the organizational unit, ou, and domain components, dc

```
<!-- ldapconfig_$volumeName.xml -->
<!-->
<Config>
 <!--The name of the LDAP server.-->
 <Server>servername.actuate.com</Server>
 <!--The port number where the LDAP server listens.-->
 <Port>389</Port>
 <!--The base DN used for user queries.-->
 <UserBaseDN>ou=actuate users, dc=actuate, dc=com
 </UserBaseDN>
</Config>
```

- 2** Save the file to the following location, naming the file, ldapconfig\_{\$volumeName}.xml, changing \$volumeName to the volume name:  
\$SERVER\_HOME\etc\

### How to configure and deploy an LDAP configuration file for external registration

Install the Java RSSE external registration example on BIRT iHub by performing the following tasks:

- 1** Using a source code editor, create an LDAP configuration file and copy or type the following code, substituting the values appropriate for the LDAP server installation:

```
<!-->
<Config>
 <!-- Name of the LDAP server. -->
 <Server>servername</Server>
 <!-- Port number where the LDAP server listens. The
 default port is 389. -->
 <Port>389</Port>
 <!-- LDAP distinguished name that the RSSE application uses
 for a query operation to the LDAP server. The Open
 Security application uses this account to validate users,
 roles, ACLs, and other Encyclopedia user information.
 Account with READ privilege is sufficient. -->
 <QueryAccount>uid=admin, ou=Administrators,
ou=TopologyManagement, o=NetscapeRoot</QueryAccount>
 <!-- Password for the LDAP account specified by the
 QueryAccount parameter. -->
 <QueryPassword>actuate</QueryPassword>

 <!-- LDAP distinguished name that the RSSE application uses
 to locate the LDAP user object, including attributes for
 the organizational unit, ou, and domain components, dc. -->
 <UserBaseDN>ou=AcUsers, dc=actuate, dc=com</UserBaseDN>

 <!-- Name of LDAP object class that the Actuate open
 security application uses to find Actuate user names. -->
 <UserObject>inetorgperson</UserObject>

 <!-- Actuate role attribute that indicates that an LDAP user
 object can perform Encyclopedia volume administration. -->
 <AdminRole>AcAdmin</AdminRole>

 <!-- LDAP role object name that maps to the Encyclopedia
 volume Operator role. -->
 <OperatorRole>AcAdmin</OperatorRole>
```

```

<!-- LDAP role object that maps to the All role in the
 Encyclopedia volume. -->
<AllRole>All</AllRole>

<!-- LDAP distinguished name that the RSSE application uses
 to locate the LDAP role object. -->
<RoleBaseDN>ou=AcRoles,dc=actuate,dc=com</RoleBaseDN>

<!-- LDAP object class that the Actuate open security
 application uses to find Actuate role names. -->
<RoleObject>groupofuniqueNames</RoleObject>

<!-- LDAP distinguished name that the RSSE application uses
 to locate the LDAP Actuate notification group object.
 GroupBaseDN can be the same as the role DN, if Group
 information is not separately maintained. -->
<GroupBaseDN>ou=groups,dc=actuate,dc=com</GroupBaseDN>

<!-- LDAP object class that the Actuate open security
 application uses to find Actuate notification group
 names. -->
<GroupObject>groupofuniqueNames</GroupObject>

<!-- Name of the LDAP group used for notifications of all job
 requests made in the iHub. The base DN is obtained from
 GroupBaseDN. -->
<GroupToNotify>specialGroup</GroupToNotify>

<!-- LDAP attribute used to retrieve the EmailAddress
 property of the user. No default value. -->

<EmailAddressAttr>mail</EmailAddressAttr>

<!-- LDAP attribute used to retrieve the license option
 property of the user. No default value. -->
<LicenseOptionsAttr>actuateLicenseOptions
 </LicenseOptionsAttr>

<!-- LDAP attribute used to retrieve the HomeFolder property
 of the user. No default value. -->
<HomeFolderAttr>actuateHomeFolder</HomeFolderAttr>

<!-- LDAP attribute used to retrieve the AttachReportInEmail
 property of the user. -->
<AttachReportInEmailAttr>actuateEmailForm
 </AttachReportInEmailAttr>

```

```

<!-- Permitted values are "included" or "linked". The default
 value is "linked". -->
<AttachReportInEmailDefault>linked
</AttachReportInEmailDefault>

<!-- LDAP attribute used to retrieve the email preferences,
 SendEmailForSuccess and SendEmailForFailure properties of
 the user. For some object classes such as inetorgperson,
 an e-mail attribute exists in the standard LDAP schema. -->
<SendEmailAttr>actuateEmailWhen</SendEmailAttr>

<!-- Permitted values are "never", "always", "failures", or
 "successes". -->
<SendEmailDefault>never</SendEmailDefault>

<!-- LDAP attribute used to retrieve the notification
 preferences, SendNoticeForSuccess and SendNoticeForFailure
 properties of the user. -->
<SendNoticeAttr>actuateFolderWhen</SendNoticeAttr>

<!-- Permitted values are "never", "always", "failures",
 "successes". -->
<SendNoticeDefault>always</SendNoticeDefault>

<!-- LDAP attribute used to retrieve the
 SuccessNoticeExpiration property of the user. The default
 value causes BIRT iHub to delete notices according to
 volume settings.-->
<SuccessNoticeExpirationAttr>
 actuateSuccessNoticeExpiration
</SuccessNoticeExpirationAttr>

<!-- Value to use for SuccessNoticeExpirationAttr when LDAP
 does not contain a value for that attribute. The value is
 the number of minutes. The default value of 0 (zero)
 causes BIRT iHub to delete notices according to volume
 settings. A value of -1 means that BIRT iHub keeps notices
 indefinitely. -->
<SuccessNoticeExpirationDefault>0
</SuccessNoticeExpirationDefault>

<!-- LDAP attribute used to retrieve the
 FailureNoticeExpiration property of the user. The default
 value causes BIRT iHub to delete notices according to
 volume settings. -->
<FailureNoticeExpirationAttr>actuateFailNoticeExpiration
</FailureNoticeExpirationAttr>

```

```

<!-- Value to use for FailureNoticeExpirationDefault when
 LDAP does not contain a value for that attribute. The
 value is the number of minutes. The default value of 0
 (zero) causes BIRT iHub to delete notices according to
 volume settings. A value of -1 means that BIRT iHub keeps
 notices indefinitely. -->
<FailureNoticeExpirationDefault>0
</FailureNoticeExpirationDefault>

<!-- LDAP attribute used to retrieve the privilege template
 for a user. Value is a comma-separated list of user or
 role privileges. A user permission is a user name followed
 by "=" and a string of 0 (zero) or more permission
 characters. A role permission is a role name followed by a
 "~" and a string of permission characters.
 Permissible characters and their meanings are:
 "r" = read
 "w" = write
 "e" = execute
 "d" = delete
 "v" = visible
 "s" = secure read (page level read)
 "g" = grant
 Examples: bob=rwed, viewing only~rv -->
<PrivilegeTemplateAttr>actuateDefaultPriv
</PrivilegeTemplateAttr>

<!-- Value to use for PrivilegeTemplateAttr when LDAP does
 not contain a value for that attribute. -->
<PrivilegeTemplateDefault/>

<!-- LDAP attribute used to retrieve the MaxJobPriority
 property of the user. The default value is 500. The
 permissible range is 0-1000. -->
<MaxJobPriorityAttr>actuateMaxPriority
</MaxJobPriorityAttr>

<!-- Value to use for MaxJobPriority when LDAP does not
 contain a value for that attribute. Default is 500. -->
<MaxJobPriorityDefault>500</MaxJobPriorityDefault>

<!-- LDAP attribute used to retrieve the ViewPreference
 property of the user. -->
<ViewPreferenceAttr>actuateViewingPref
</ViewPreferenceAttr>

```

```

<!-- Value to use for ViewPreferenceAttr when LDAP does not
 contain a value for that attribute. Permissible values are
 "default" and "dhtml".-->
<ViewPreferenceDefault>default</ViewPreferenceDefault>
<!-- LDAP attribute used to retrieve the channel subscription
 list of the user. The values are specified as a comma-
 separated list. -->
<ChannelSubscriptionListAttr>actuateChannelList
</ChannelSubscriptionListAttr>

<!-- Value to use for ChannelSubscriptionListAttr when LDAP
 does not contain a value for that attribute. The value is
 a comma-separated lists of channel names or is empty. -->
<ChannelSubscriptionListDefault/>
<ConnectionPropertyList>
 <ConnectionProperty>
 <Name>username</Name>
 <Value>testUser</Value>
 </ConnectionProperty>
 <ConnectionProperty>
 <Name>password</Name>
 <Value>mypassword</Value>
 </ConnectionProperty>
</ConnectionPropertyList>
<!-- LDAP attributes used when externalizing
 ConnectionPropertyList, containing username and password.
 Typically used when implementing pass-through security. Do
 not include the ConnectionPropertyList if not
 externalizing these properties. -->
<ConnectionPropertyList>
 <ConnectionProperty>
 <Name>username</Name>
 <Value>testUser</Value>
 </ConnectionProperty>
 <ConnectionProperty>
 <Name>password</Name>
 <Value>mypassword</Value>
 </ConnectionProperty>
</ConnectionPropertyList>
</Config>

```

- 2** Save the file to the following location, naming the file,  
ldapconfig\_{\$volumeName}.xml, changing \$volumeName to the volume name:  
\$SERVER\_HOME\etc\

#### How to configure external user registration

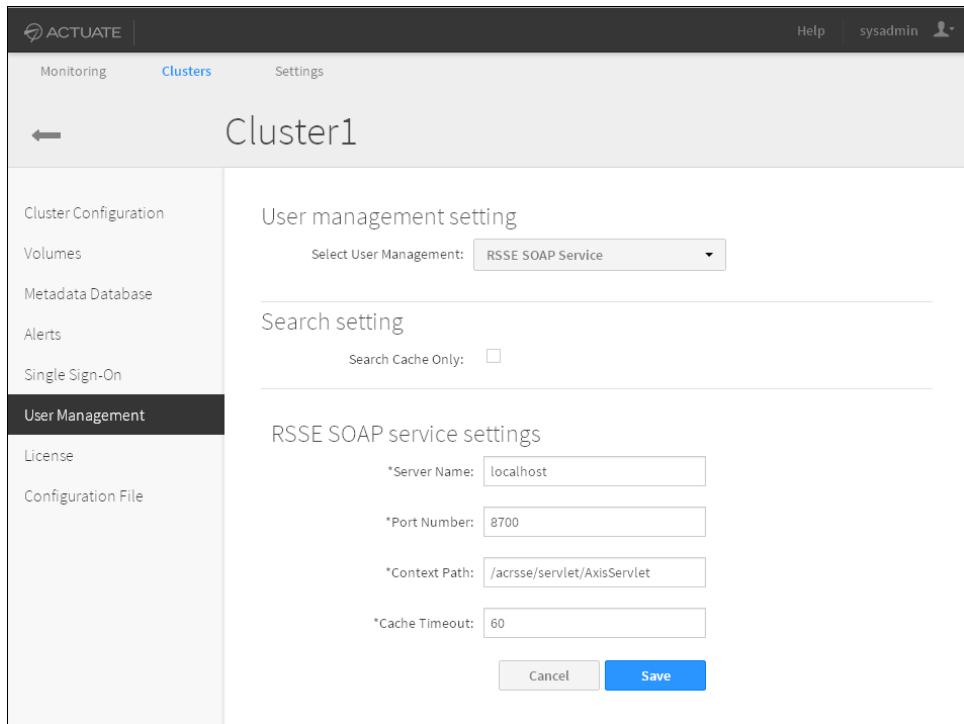
To use a Java RSSE application that utilizes LDAP for external user registration, the system administrator must enable an Open Security web service in

System Console—User Management—RSSE Soap Service Settings, then restart the system. Also, configure an LDAP server database to contain the volume's user information. For more information about configuring the LDAP server database, see the LDAP server documentation.

### **How to configure an RSSE SOAP Service**

To configure an RSSE SOAP service that uses an RSSE web service application, perform the following tasks:

- 1** In System Console, choose Clusters, then choose Edit for an existing cluster.
- 2** Choose User Management.
- 3** In User Management Setting—Select User Management, choose RSSE SOAP Service.
- 4** In RSSE SOAP service settings, configure the following properties for the RSSE SOAP Service, as shown in Figure 20-1:
  - Server Name  
Machine name of the server that runs the RSSE web service.
  - Port Number  
Port number for the RSSE web service.
  - Context Path  
Specifies the location of the RSSE web service for iHub to use when sending messages to the web service. The default path is /acrsse/servlet/AxisServlet.
  - Cache Timeout  
Number of minutes before BIRT iHub deletes cached data



**Figure 20-1** Configuring RSSE SOAP service settings

## Installing the page-level security application

To install the Java RSSE page-level security sample application on BIRT iHub, deploy an external access control list (ACL) file with the application. Perform this operation before enabling the web service on the volume.

For more information about deploying the ACL file in the page-level security application installation, see “How to install the Java RSSE page-level security application,” later in this chapter.

To complete the installation, perform the following steps:

- 1 Load a sample executable report to the volume.
- 2 Run the executable report to create a report document.
- 3 Configure permissions for these report files to test the sample application installation.

## Migrating a Java RSSE application to a new Actuate release

When migrating a Java RSSE application from an older release of Actuate software to a newer release, the application may require recompilation using the libraries of the new release.

When modifying the software, update any references of the older release to reference the new release, and place all relevant JAR files that contain classes for the new version into their proper locations.

---

## Using page-level security

Using the iHub Report Server Security Extension (RSSE) framework, a developer can create an RSSE service that manages page-level security in a BIRT design by retrieving a user access control list (ACL) externally.

By default, when a secure report asks for the ACL of a user, the volume returns a list that includes the user ID and the roles in which the user is a member.

Frequently, the information in BIRT iHub security does not match the information in a database used by a secure design. An RSSE page security application can translate a BIRT iHub ACL to a design-specific ACL.

### How to install the Java RSSE page-level security application

BIRT iHub Integration Technology contains an example of how external page-level security works using Java RSSE in the subdirectory, Page\_Security\_Example. For information about BIRT page-level security, see the Actuate BIRT documentation.

To install the page-level security sample application on BIRT iHub, deploy an external ACL file with the application. Perform this operation before enabling the web service.

To include the ACL file provided with the sample application in the build, perform the following operations:

- 1 Copy the file, user.acls, located in the Page\_Security\_Example directory to the following location:

```
/com/actuate11/rsse/aclSample
```

- 2 Using a source code editor, in Page\_Security\_Example directory, open the file, build.xml, and perform the following operations:

- 1 In build.xml, navigate to the buildACL element specifying the contents of the file, rsseAcl.jar.

- 2 Modify the fileset list to contain the following line of code:

```
<include name="com/**/*.acls" />
```

The buildACL element looks like the following example:

```
<target name="buildACL" depends="buildACL.clean, compileACL">
 <mkdir dir="lib"/>
 <jar jarfile="lib/rsseAcl.jar">
 <fileset dir=".">
 <include name="com/**/*.class" />
 <include name="com/**/*.properties" />
 <include name="com/**/*.acls" />
 </fileset>
 </jar>
</target>
```

**3** Build the application using the Apache Ant tool.

For more information about building a Java RSSE application using Apache Ant, see “How to build a Java RSSE sample application,” earlier in this chapter.

**4** Copy the file, rsseAcl.jar, to the lib directory of the BIRT iHub servlet container and configure the class.properties file.

For more information about copying the archive file for a Java RSSE application to the lib directory for the BIRT iHub servlet container and configuring the class.properties file, see “How to install a Java RSSE application,” earlier in this chapter.

**5** Configure the open security as a web service.

For more information about enabling an RSSE application to run as a web service, see “How to configure an RSSE SOAP Service,” earlier in this chapter.

## Creating an access control list (ACL)

The file, user.acls, stores a user access control list (ACL) using the following format:

```
Username=acl1, acl2, ..
```

The user name field matches the name of user in the volume. An equal ('=') sign separates the user name from the ACL list. An ACL list can contain zero, one, or more ACL specifications, as shown in the following code example:

```
user1=acl1, acl2, acl3, acl4
user2=acl5, acl6, acl7, acl8
user3=acl9
user4=acl10
```

If there is more than one ACL specification in the list, separate each ACL using a comma. The scanner reading the users.acls file eliminates any white space or backslash.

All the user name specifications in the example are legal. A list can contain users that do not appear in the volume. The information for these users is ignored.

## Deploying a report to a volume

Test page-level security by deploying the sample design to the volume. For example, a report may shows information about the sales reps in the following city offices:

- NYC
- Boston
- Philadelphia

User1 has access to the pages with information about NYC office, user2 to the Boston office, and user3 to the Philadelphia office. The file, user.acls, contains the following access control list specifications:

```
user1=NYC
user2=Boston
user3=Philadelphia
```

To deploy the sample design to a volume, perform the following steps:

- 1 Using BIRT iHub Visualization Platform, log in to the volume as Administrator.
- 2 In Documents, choose Upload to upload the design to the volume.
- 3 In Documents, from the design file drop-down list, choose Run to execute the design.

On Parameters, enter parameters as appropriate. Choose Finish to complete the running of the report and generate an output document.

The administrator is able to see all three offices.

User1 can only see the information for the NYC office, as shown in Figure 20-2.

## Authenticate



**Figure 20-2** Example of document output

To change the assignments in the file, user.acls, wait for the volume cache time-out period. Alternatively, put the volume offline, restart the BIRT iHub application container, then take the volume online again before checking to see if the changes are effective.

---

## Report Server Security Extension (RSSE) API operations

This section describes RSSE API operations. Note that TargetVolume and OrgId are set in the SOAP header of each request going to an RSSE service.

OrgId is an optional value to filter the users for a specific volume, which can be configured in System Console. How this value is used depends on the RSSE implementation.

---

## Authenticate

Verifies that the user is authorized to access the BIRT iHub System. Implement Authenticate for external user authentication and external user registration.

**Request elements** **User**  
The name of the user logging on to BIRT iHub.

**Password**  
The user password.

**Credentials**  
Additional credentials for authenticating the user.

**UserSetting**

Specifies whether to return the user properties. If True, returns the user properties.

<b>Response elements</b>	<b>UserAndProperties</b> The user name and properties.
--------------------------	-----------------------------------------------------------

## DoesRoleExist

Verifies whether the role exists in the external directory. BIRT iHub can call this function to clear references to deleted roles.

<b>Request elements</b>	<b>RoleName</b> The name of the role to verify.
-------------------------	----------------------------------------------------

<b>Response elements</b>	<b>Exists</b> Indicates whether the role exists. If True, the role exists.
--------------------------	-------------------------------------------------------------------------------

## DoesUserExist

Verifies whether the user exists in the external directory. BIRT iHub can call this function to clear references to deleted users.

<b>Request elements</b>	<b>UserName</b> The name of the user to verify.
-------------------------	----------------------------------------------------

<b>Response elements</b>	<b>Exists</b> Indicates whether the user exists. If True, the user exists.
--------------------------	-------------------------------------------------------------------------------

## GetConnectionProperties

Retrieves the connection properties for a user or role from an external data source for a pass-through security operation. In pass-through security, an information object's DCD file sets the securityPolicy to TranslatedCredential. The proxy user name and password settings, specifying the user login credentials in the DCD, contain empty quotes and are ignored by the implementation.

<b>Request elements</b>	<b>FileName</b> The fully qualified name of an information object's data connection definition (DCD) file.
-------------------------	---------------------------------------------------------------------------------------------------------------

<b>UserName</b>	The name of the user or role.
-----------------	-------------------------------

<b>Response elements</b>	<b>ConnectionProperties</b> The requested name and value pairs.
--------------------------	--------------------------------------------------------------------

---

## GetTranslatedRoleNames

Maps the external security role names to Actuate security role names. Either use GetTranslatedRoleNames in conjunction with the external registration security level, or use the same role names for the external and Actuate roles.

For example, a user with the Actuate Administrator security role can manage all items in a volume. If the Administrator role in the external security system has a different meaning, GetTranslatedRoleNames can map the external security role to an Actuate role with a different name.

<b>Request elements</b>	<b>GetTranslatedRoleNames</b> The external translated role names that map to the Actuate security role names.
<b>Response elements</b>	<b>TranslatedRoleNames</b> The names that Actuate uses for external security roles.

---

## GetTranslatedUserNames

Maps the external security user names to Actuate security user names. Either use GetTranslatedUserNames in conjunction with the external registration security level, or use the same user names for the external and Actuate users.

For example, a user with the Actuate Administrator privilege can manage all items in a volume. If the Administrator user in the external security system has a different meaning, GetTranslatedUserNames can map the external security user to an Actuate user with a different name. Deprecated in BIRT iHub Release 3.

<b>Request elements</b>	<b>GetTranslatedUserNames</b> The external translated user names that map to the Actuate security user names.
<b>Response elements</b>	<b>TranslatedUserNames</b> The names that Actuate uses for external security users.

---

## GetUserACL

Retrieves the user ACL. GetUserACL applies only if using page-level security. Page-level security controls printing, navigating, and all aspects of user viewing. Page-level security requires the Page Level Security option on BIRT iHub.

<b>Request elements</b>	<b>UserName</b> The name of the user whose ACL to retrieve.
<b>Response elements</b>	<b>ACL</b> The list of pages of a document to which the user has access.

---

## GetUserProperties

Retrieves the user properties from an external directory. Regardless of security level implementation, implement GetUserProperties when the user properties are stored in an external security source.

**Request elements** **User**  
The name of the user whose properties to retrieve.

**ResultDef**  
The properties to retrieve. Can contain user property names, roles, and PrivilegeTemplate.

**Response elements** **ArrayOfUserAndProperties**  
The user properties.

---

## GetUsersToNotify

Retrieves the list of users to notify about completed jobs.

**Request elements** **GetUsersToNotify**  
The list of users to notify about completed jobs.

**Response elements** **Users**  
The list of users to notify.

---

## PassThrough

Calls the RSSE for general purposes such as changing or refreshing the internal library state. If implemented, the RSSE calls PassThrough in response to the BIRT iHub receiving the Information Delivery API CallOpenSecurityLibrary request.

The RSSE passes the ReturnCode as a response to CallOpenSecurityLibrary, RSSE does not interpret the parameter.

**Request elements** **Input**  
The input parameter string.

**Response elements** **Output**  
The output parameter string.

**ReturnCode**  
The integer parameter that the caller of CallOpenSecurityLibrary interprets.

---

## SelectRoles

Searches for roles that match the specified criteria. Required if using an external registration security level.

SelectRoles can also retrieve a user roles. To retrieve a user roles, specify a name in UserName. SelectRolesResponse then returns the list of the user roles.

The SelectRoles SOAP message invokes the SelectRolesOfUser method within the Java code, and does not invoke the SelectRoles method. iHub does not use the SelectRoles method to link a user account to a role.

**Request elements****QueryPattern**

The string match.

**UserName**

The name of a user whose information to retrieve.

**FetchSize**

The maximum number of records to retrieve and return in a result set. The default value is 500.

**Response elements****Roles**

The list of roles matching the search criteria.

**TotalCount**

The number of entries in the search result set.

---

## SelectUsers

Retrieves the names of users that match the specified criteria. For example, to retrieve the names of all users in the Sales group, specify Sales in GroupName.

SelectUsers is required if using an external registration security level.

**Request elements****QueryPattern**

The string match.

**RoleName**

The name of the role whose members to retrieve.

**GroupName**

The name of the group whose members to retrieve.

**FetchSize**

The maximum number of records to retrieve and return in a result set. The default value is 500.

**Response elements** **Users**  
 The list of users matching the search criteria.

**TotalCount**  
 The number of entries in the search result set.

---

## Start

Initializes the RSSE. Implement Start to initialize RSSE. Called once for each cluster

**Request elements** **ServerHome**  
 The path to the BIRT iHub installation, for example  
 C:\Program Files\Actuate\Server on Windows.

**Volume**  
 The name of the volume.

**LogFile**  
 The path to the log file for RSSE activity.

**Version**  
 The BIRT iHub version number.

**Response elements** **IntegrationLevel**  
 The integration level of external security. One of the following values:

- External\_Authentication
- External\_Registration
- None

**ExternalProperties**  
 The following external user or role properties are included in the response:

- EmailAddress  
 The user e-mail address
- HomeFolder  
 The user home folder
- EmailForm  
 The form of the e-mail attachment, included or linked
- EmailWhen  
 The type of notification to use for a completed job
- FolderWhen  
 An indicator of when the user uses the Completed folder

## Stop

- SuccessNoticeExpiration  
The number of minutes that elapse before the Encyclopedia service deletes a successful job notice
- FailNoticeExpiration  
The number of minutes that elapse before the Encyclopedia service deletes a failed job notice
- DefaultObjectPrivileges  
The privileges that the user has by default on the objects the user creates
- MaxPriority  
The maximum request priority the user can set when creating a report printing or generation request
- ViewPreference  
The user web viewing preference, default or DHTML

### **RSSEVersion**

The version of RSSE.

### **UserACLExternal**

Specifies whether the user access list is stored externally. Applies only if using page-level security.

### **ConnectionPropertyExternal**

Specifies whether the user connection properties are retrieved externally from the RSSE. If True, the connection properties are retrieved externally. In this case, BIRT iHub directs requests to set connection properties to the RSSE and does not use GetConnectionProperties.

### **SelectUsersOfRole**

Applies only under external registration. Specifies whether the RoleName element in SelectUsers is implemented. The setting indicates whether BIRT iHub enables this feature. The default value is False.

### **SupportGetTranslatedUserNames**

Specifies whether to support translation of user names from external source.

---

## Stop

Stops the Report Server Security Extension service. Implement Stop to close RSSE and free system resources.

<b>Request elements</b>	<b>Stop</b>
	Closes RSSE and free system resources.

<b>Response elements</b>	<b>StopResponse</b> Contains status information on results of stop operation.
--------------------------	----------------------------------------------------------------------------------

---

## Report Server Security Extension (RSSE) API data types

This section describes the RSSE API data types. Some data types have the same name as data types within the IDAPI, but do not have the same content.

---

### Arrays of data types

Data type definitions can be grouped into arrays. Each array has a specific definition for that data type.

The following data types have arrays defined and used in the RSSE:

- Permission
- PropertyValue
- String
- UserAndProperties

---

### ErrorMessage

The contents of a error message, including the request name, error code, and description.

<b>Elements</b>	<b>RequestName</b> The request name or call included in the error message.
	<b>ErrorCode</b> The error code.
	<b>Description</b> The description associated with the error code.

---

### Permission

A complex data type that describes a user access rights.

## PropertyValue

### Elements    **RoleName**

The role name.

### **UserName**

The user name.

### **AccessRight**

The privileges the user or role has on an object. One or more of the following characters representing a privilege:

- D—Delete
- E—Execute
- G—Grant
- V—Visible
- S—Secured Read
- R—Read
- W—Write

---

## PropertyValue

A complex data type that describes a name-value pair.

### Elements    **Name**

The name portion of the name-value pair.

### **Value**

The value portion of the name-value pair.

---

## TranslatedRoleNames

A complex data type that describes the role names RSSE uses that match external role names.

### Elements    **Administrator**

The Administrator role name.

### **Operator**

The Operator role name.

### **All**

All other role names.

---

## TranslatedUserNames

A complex data type that describes the administrator name RSSE uses that matches the external administrator name.

**Elements** **Administrator**

The Administrator user name.

---

## User

A complex data type describing an RSSE user and their attributes.

**Elements** **Name**

The username used to connect to iHub.

**EmailAddress**

The user e-mail address.

**HomeFolder**

The users's home folder.

**LicenseOptions**

The user license options.

**ViewPreference**

The user viewer, Default or DHTML.

**MaxJobPriority**

The maximum priority that the user can assign to a job.

**SendNoticeForSuccess**

Specifies whether the BIRT iHub sends success notices to the user.

**SendNoticeForFailure**

Specifies whether the BIRT iHub sends failure notices to the user.

**SuccessNoticeExpiration**

Specifies the minimum number of minutes success notices remain in the Completed folder. To set the user success notices to never expire, set the value to 0xffffffff.

**FailureNoticeExpiration**

Specifies the minimum number of minutes failure notices remain in the Completed folder. To set the user failure notices to never expire, set the value to 0xffffffff.

## UserAndProperties

### **SendEmailForSuccess**

Specifies whether the BIRT iHub sends success notices in an e-mail message to the user.

### **SendEmailForFailure**

Specifies whether the BIRT iHub sends failure notices in an e-mail message to the user.

### **AttachReportInEmail**

Specifies whether to attach a report to an e-mail completion notice.

---

## UserAndProperties

A complex data type describing an RSSE user and properties, such as privileges and roles.

### **Elements**

#### **User**

The reference to the user object that contains the details to use to connect to iHub.

#### **PrivilegeTemplate**

Permission list as specified in privilege template.

#### **Channels**

Assigned channel names.

#### **Roles**

Assigned role names.

# Part **Seven**

---

**Working with usage and error logging**



# 21

## Using Actuate logging and monitoring APIs

This chapter contains the following topics:

- About usage logging and error logging extensions
- Developing usage and error logging extensions
- Customizing the usage logging extension
- Customizing the error logging extension
- About the usage log
- About the error log
- About BIRT iHub usage and error log consolidator

---

## About usage logging and error logging extensions

BIRT iHub System provides a monitoring framework that logs BIRT iHub usage and error information. You can use this information to understand how BIRT iHub uses system resources and troubleshoot problems.

BIRT iHub Integration Technology provides usage and error logging extensions that retrieve and write log data to files. The usage and error logging extensions are DLLs on a Windows platform and shared libraries on a UNIX system. BIRT iHub Integration Technology provides the customizable source code for the usage and error logging extensions as reference implementations.

---

## Interpreting AC\_SERVER\_HOME

This chapter makes reference to the AC\_SERVER\_HOME environment variable. In a default BIRT iHub installation on Windows, performed using the graphical installer, in which the install folder is C:\Actuate\BIRTiHubVisualization, AC\_SERVER\_HOME represents the following path:

C:\Actuate\BIRTiHubVisualization\modules\BIRTiHub\iHub

In a default BIRT iHub installation on Linux, in which the install folder is /home/actuate/iHub3, AC\_SERVER\_HOME represents the following path:

/home/actuate/iHub3/modules/BIRTiHub/iHub

---

## Developing usage and error logging extensions

The usage and error logging extensions are open framework applications. These reference implementations log the information that the BIRT iHub monitoring framework captures to files. A developer can customize the way the DLL or shared library handles the usage and error log information. For more information about these usage and error log extensions, refer to the readme files in <BIRT iHub Integration Technology install folder>/ServerIntTech3 /User Activity Logging Extension and Error Logging Extension.

BIRT iHub installs compiled versions of the usage and error logging extensions in AC\_SERVER\_HOME/bin. In Windows, the DLL applications files are UsrActivityLoggingExt.dll and ErrorLoggingExt.dll. In Linux and UNIX, the shared library applications files are libUsrActivityLoggingExt.so and libErrorActivityLoggingExt.so.

A usage or error log file is a comma-separated values (CSV) file. The default name for a usage log file is usage\_log\_1.csv. The default name for an error log file is error\_log\_1.csv.

In Actuate Release 10, the usage and error logs for each volume were written to separate directories. In Release 11 and iHub, the usage and error logs for all volumes are consolidated in one directory, AC\_SERVER\_HOME /UsageErrorLogs/primary, and written to one usage and one error log. The directory for the usage and error log files is not configurable.

## Configuring usage and error logging

The administrator sets up usage and error logging by configuring property attributes of the <UsageAndErrorLogging> element in acserverconfig.xml. Each property attribute name starts with the name of the log type.

Usage logs consist of the following types:

- Admin
- Deletion
- Eii (Data Integration)
- Generation (Factory)
- Printing
- Viewing

There is only one type of error log, Error.

The <UsageAndErrorLogging> element contains the following properties for each log type:

- LogLevel

For usage logs, the possible values for LogLevel are Standard or Detail. For deletion, printing, and viewing logging, Standard and Detail provide the same level of information. For Factory logging, detailed information includes report parameters. Logging detailed Factory information, instead of standard Factory information, causes performance degradation.

For the error log, the possible values are:

- Information

Logs informational messages to help track BIRT iHub behavior.

- Warning

Logs warning errors that typically do not impact normal BIRT iHub operation.

- Severe

Logs errors that can cause BIRT iHub to abort execution if you do not rectify the cause of the error. A severe error does not typically cause BIRT iHub to abort execution immediately.

- Fatal
  - Logs critical errors from which BIRT iHub cannot recover. A fatal error typically causes BIRT iHub to abort execution immediately.
- LogEnabled
  - The possible values for LogEnabled are true or false. True enables logging, false disables logging.
- LoggingExtensionName
  - Name of the logging extension. The default usage logging extension is UsrActivityLoggingExt. The default error logging extension is ErrorLoggingExt.

Use the example shown in Listing 20-1 to configure the logging properties in acserverconfig.xml

---

**Listing 20-1** Configuring logging properties in acserverconfig.xml

---

```
<System
...
 CustomEventServiceConnectionTimeout="300">
 <UsageAndErrorLogging
 EiiLogLevel="Standard"
 AdminLogLevel="Standard"
 EiiLogEnabled="true"
 ErrorLogLevel="Information"
 AdminLogEnabled="true"
 ErrorLogEnabled="true"
 ViewingLogLevel="Standard"
 DeletionLogLevel="Standard"
 PrintingLogLevel="Standard"
 ViewingLogEnabled="true"
 DeletionLogEnabled="true"
 GenerationLogLevel="Standard"
 PrintingLogEnabled="true"
 GenerationLogEnabled="true"
 ErrorLoggingExtensionName="ErrorLoggingExt"
 UsageLoggingExtensionName="UsrActivityLoggingExt"/>
 <SMTPServers/>
</System>
```

---

## Customizing the usage logging extension

To customize the usage logging extension, you must install BIRT iHub Integration Technology. BIRT iHub Integration Technology provides the C and C++ source code that you modify to customize the usage logging extension. The usage

logging extension source code file, usagelogext.c, installs in <BIRT iHub Integration Technology install folder>/ServerIntTech3 /User Activity Logging Extension.

The usagelogext.c source code implements the following functionality:

- Specifies the name and extension of the usage log file, and the location or path to the file

These properties are defined as constants. For example:

```
#define USAGELOG_FILE_NAME "usage_log"
#define USAGELOG_FILE_EXT ".csv"
```

The AcStartUsageLog function implements this functionality.

- Writes information about every transaction that BIRT iHub captures to a log file

The AcLogUsage function implements this functionality.

- Stops logging information and releases the resources the extension uses

The AcStopUsageLog function implements this functionality.

- Specifies whether the extension is multithread-safe

The AcIsThreadSafe function implements this functionality. If the extension is not multithread-safe, AcIsThreadSafe must return False. The reference implementation is not multithread-safe.

---

## Customizing the error logging extension

To customize the error logging extension, you must install BIRT iHub Integration Technology. BIRT iHub Integration Technology provides the C and C++ source code that you modify to customize the error logging extension. The error logging extension source code file, errorlogext.c, installs in <BIRT iHub Integration Technology install folder>/ServerIntTech3 /Error Logging Extension.

The errorlogext.c source code implements the following functionality:

- Specifies the name and extension of the log file, and the location or path to the log file

These properties are defined as constants. For example:

```
#define ERRORLOG_FILENAME "error_log"
#define ERRORLOG_FILE_EXT ".csv"
```

The AcStartErrorLog function implements this functionality.

- Writes the information about every error that BIRT iHub encounters to a log file  
The Ac.LogError function implements this functionality.
  - Stops logging information and releases the resources the extension uses  
The AcStopErrorLog function implements this functionality.
  - Specifies whether or not the extension is multithread-safe  
The AcIsThreadSafe function implements this functionality. If the extension is not multithread-safe, AcIsThreadSafe must return False. The reference implementation is not multithread-safe.
- 

## About the usage log

The usage log, `usage_log.csv`, is a comma-separated values (CSV) file. The usage log records the following events:

- Report viewing
- Report printing
- Report generation
- Report deletion
- Administrative
- Data integration

## About types of recorded events

For each type of event, you can set the logging level to Standard or Detail. If you are using the default usage logging extension, `UsrActivityLoggingExt`, the logging level does not affect how the file records the following types of events:

- Report viewing
- Report printing
- Report deletion

If you set the logging level for report generation or factory events to Detail, the usage log includes report parameters. Setting the logging level to Detail for report generation events decreases performance.

## Understanding a usage log entry

Each usage log entry is a comma-separated list containing up to 40 fields of information about an event. The following example describes a delete user event:

3272649170,5,1,3272649170,3272649170,---,0,Administrator,3,  
enl2509,enl2509,enl2509,User,testUser,----,-,----,-,----,-,----,-,  
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

The usage log organizes the entry fields into the following information groups:

- Fields 1 through 10 contain general information:
    - Fields 1, 4, and 5 contain the log file time stamp, start time, and finish time. The time is in seconds since 00:00:00, Jan. 1, 1901, GMT.
    - Field 2 contains the event type. The numeric values in Table 20-1 indicate the event types.

**Table 20-1** Event types and the corresponding event values

Event type	Event value
ReportGeneration	1
ReportPrinting	2
ReportViewing	3
ReportDeletion	4
Admin	5
Query	6
Search	7

- Field 3 contains the event result. The value for the event result is either 1 or 0, indicating success or failure.
  - Fields 6 through 8 contain report output information, indicating the file name, version, and file size. The report output group information appears only with report events. A dash indicates a field is not used.
  - Fields 9 and 10 contain execution information, indicating the user name and the BIRT iHub subsystem where the operation executed. The numeric values in Table 20-2 indicate the BIRT iHub subsystems.

**Table 20-2** BIRT iHub subsystems and the corresponding ID numbers

Subsystem	ID number
ReportEngine	1
ViewEngine	2
EncycEngine	3

**Table 20-2** BIRT iHub subsystems and the corresponding ID numbers

Subsystem	ID number
IntegrationEngine	4
Cache	5

- Fields 11 through 25 contain operational information in string format, including the volume, BIRT iHub, and cluster names. Fields 26 through 40 contain operational information in numeric format.  
The values in these fields depend on the value for the event type in field 2. Table 20-3 summarizes some of the information available for each event type at Standard level.

**Table 20-3** Examples of information that is available about the different types of events

Event type	Event value	Operation data available
Report generation	1	<p>String fields 11 through 21 display the following information:</p> <ul style="list-style-type: none"> <li>- , executable name, executable version, volume name, server name, cluster name, resource group name, node running request, page count, job name, request ID</li> </ul> <p>A dash indicates a field is not used.</p> <p>Numeric fields 26 through 29 display the following information:</p> <ul style="list-style-type: none"> <li>number of pages, submit time, job type, job priority</li> </ul>
Report printing	2	<p>String fields 11 through 18 display the following information:</p> <ul style="list-style-type: none"> <li>page numbers printed, volume name, printer name, server name, cluster name, node sent to, file type, server request id</li> </ul> <p>Numeric fields 26 through 29 display the following information:</p> <ul style="list-style-type: none"> <li>number of pages printed, submit time, job type, job priority</li> </ul>
Report viewing	3	<p>String fields 11 through 18 display the following information:</p> <ul style="list-style-type: none"> <li>output format, report page numbers, volume name, server name, cluster name</li> </ul> <p>Numeric field 26 displays the number of pages viewed.</p>

**Table 20-3** Examples of information that is available about the different types of events

Event type	Event value	Operation data available
Administrative	5	<p>String fields 11 through 13 display the following information: volume name, server name, cluster name</p> <p>Numeric field 26 displays an operation ID for an administration event. The following list provides the event name for each operation ID:</p> <ul style="list-style-type: none"><li>■ 1 Create</li><li>■ 2 Delete</li><li>■ 3 Modify</li><li>■ 4 Login</li></ul>
Actuate Integration service	6	<p>String fields 11 through 14 display the following information: volume name, server name, cluster name, server request id</p> <p>Numeric fields 26 and 27 display the following information: request wait time, request generation time</p>
Search	7	<p>String fields 11 through 15 display the following information: report format, page numbers, volume name, server name, cluster name</p> <p>Numeric field 26 displays the number of pages viewed.</p>

## About the error log

The error log, `error_log.csv`, is a comma-separated values (CSV) file. If you use the default error logging extension, `ErrorLoggingExt`, you can set the logging level to:

- Information  
The error log records messages that trace BIRT iHub behavior.
- Warning  
The error log records warnings. The errors do not necessarily affect the operation of BIRT iHub.
- Severe  
The error log records errors that can result in BIRT iHub failure if you do not correct them.
- Fatal  
The error log records critical errors from which BIRT iHub cannot recover and that can result in failure.

## Understanding an error log entry

Each error log entry is a comma-separated list containing up to 12 fields about an error-related event. The following example describes an error in a submit job event:

```
3272648796,2,3230,SubmitJob,Administrator,"Invalid start time or
end time.",enl2509,enl2509,enl2509,-,-,-
```

The error log organizes the entry fields into the following information groups:

- Fields 1 through 9 contain general information:
  - Field 1 contains the log file time stamp. The time is in seconds since 00:00:00, Jan. 1, 1901, GMT.
  - Field 2 contains the error severity level, an integer between 1 and 4. The numeric values in Table 20-4 indicate the level.
  - Field 3 contains the Error ID code.

**Table 20-4** Error severity levels and the corresponding values

Error severity level	Value
Information	1
Warning	2
Severe	3
Fatal	4

- Field 4 contains the Service name, indicating the subsystem where the error occurred such as the Factory, Encyclopedia, View, or Request service.
- Field 5 indicates the volume user.
- Field 6 contains the error message.
- Field 7 contains the volume name.
- Field 8 contains the BIRT iHub cluster name.
- Field 9 contains the BIRT iHub node name.
- Depending on the error, fields 10 through 12 can contain information such as a file name and ID number. A dash indicates a field is not used.

Table 20-5 summarizes some of the information available in fields 10 through 12 for an error log entry at Standard level.

**Table 20-5** Information that is available for error log entries at the Standard level

Type of error	Operation data available
Cluster master failover	Fields 10 and 11 display the following data: <ul style="list-style-type: none"><li>■ Original cluster master</li><li>■ New cluster master</li></ul>
Volume user activity	Fields 10 through 12 can contain error parameters such as the following items: <ul style="list-style-type: none"><li>■ Object name</li><li>■ ID number</li></ul>
Volume failover	Fields 10 and 11 contain the following data: <ul style="list-style-type: none"><li>■ Primary server</li><li>■ Backup server used</li></ul>
Volume online or offline	Fields 10 and 11 contain the following data: <ul style="list-style-type: none"><li>■ Volume name</li><li>■ Operation type either online or offline</li></ul>
BIRT iHub node start or stop	Field 10 contains the BIRT iHub name
Service enable or disable	Fields 10 and 11 contain the following data: <ul style="list-style-type: none"><li>■ Server name</li><li>■ List of services</li></ul>
Archive service error	Fields 10 through 12 contain error parameters
Volume job purging field 4 is Job Purge	Fields 10 through 12 contain error parameters
Volume health monitoring field 4 is Volume Health Monitor	Fields 10 through 12 contain error parameters

## About BIRT iHub error messages

Table 20-6 lists the general categories of BIRT iHub error messages.

**Table 20-6** Categories of BIRT iHub error messages

Error ID range	Error description
0001 - 1000	System errors such as Out of memory or Low thread count

**Table 20-6** Categories of BIRT iHub error messages

Error ID range	Error description
1001 - 3000	BIRT iHub errors such as Corrupt volume or Transient storage full Within this error category, the following sub-categories exist: <ul style="list-style-type: none"><li>■ 1001 - 2000 Actuate internal datastore</li><li>■ 2001 - 3000 Actuate internal</li></ul>
3001 - 6000	User errors such as Permission denied Within this error category, the following sub-categories exist: <ul style="list-style-type: none"><li>■ 3001 - 4000 Encyclopedia engine</li><li>■ 4001 - 5000 Report engine</li><li>■ 5001 - 6000 View engine</li></ul>
6001 - 12000	<ul style="list-style-type: none"><li>■ 6001 - 7000 SOAP engine</li><li>■ 7001 - 8000 Process management daemon</li><li>■ 8001 - 9000 Cluster engine</li><li>■ 10001 - 11000 Server configuration</li><li>■ 11001 - 12000 XML parsing</li></ul>
12001 - 13000	Viewing server errors
13000 - 14000	AcMail exceptions
100001 - 100600	Actuate Information service
100601 - 100699	Actuate Caching service
100700 - 150000	Shared by Actuate Information service and Actuate Caching service

---

## About BIRT iHub usage and error log consolidator

The log consolidator application is a Java application that reads data from an BIRT iHub usage or error log file and uses JDBC to add the information to a database. In an BIRT iHub cluster, you must install and run the log consolidator application on each BIRT iHub node to consolidate the cluster's usage and error log information in a database. Before running the log consolidator application, you must install the following components:

- BIRT iHub
- Log consolidator application files

- Log consolidator configuration file
- Database used by the log consolidator application

BIRT iHub installs the required JAR files for the log consolidator application in \$ACTUATE\_HOME/Jar/UsageAndErrorConsolidator. These files include:

- usageanderrorconsolidator.jar  
The com.actuate.consolidator application class and properties files.
- Java Architecture for XML Binding (JAXB) JAR files  
JAXB provides a framework that supports run-time mapping between XML and Java objects.
- ojdbc14.jar  
The supported Oracle JDBC Driver. The reference implementation uses Oracle as the example database.
- naming-java.jar  
Contains the handler for the Java namespace.

The log consolidator application also uses the following Microsoft Windows Registry keys or UNIX environment variables set by the BIRT iHub installation process:

- On Windows, make sure the following registry keys exist:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432node\Actuate\Common\22.0
 \AC_JAVA_HOME
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432node\Actuate\Common\22.0
 \AC_JRE_HOME
```

On a machine running any of the Release 11 versions of Actuate iServer System, the location of the AC\_JAVA\_HOME and AC\_JRE\_HOME registry keys is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432node\Actuate\Common\11.0
```

The log consolidator application uses java.exe from:

```
AC_JRE_HOME\bin
```

- On UNIX, make sure the following environment variables exist:

```
AC_JAVA_HOME
AC_JRE_HOME
```

The log consolidator application uses java.exe from:

```
$AC_JRE_HOME/bin
```

In BIRT iHub Integration Technology, the UsageAndErrorConsolidator directory contains additional files that you must use to complete the installation of the log consolidator application:

- /DBScripts contains the SQL script, CreateActuateLogTables.sql, which creates the tables in the Oracle database used by the Actuate log consolidator sample application. A readme.txt file describes this SQL script file.
- /jar contains the JAR files required by the log consolidator application.
- /Setup contains the following files that startup and shutdown the log consolidator application:
  - consolidatorconfig.xml is the sample consolidator configuration file that specifies settings such as the following items:
    - Database driver, URL, encoding, schema, user name, and password
    - Usage and error log details, such as the file names, refresh interval, number of logs, and whether a log file is enabled
  - The UNIX version uses the scripts, start\_consolidator.sh and stop\_consolidator.sh, to start up and shut down the log consolidator application.
  - The Windows version uses a setup application, consolidatorwin.exe, that installs the log consolidator application as a Windows service and starts and stops the application.
  - A readme.txt file describes how to use these components.
- /src contains the following items:
  - usageanderrorconsolidator.jar, the JAR file for the log consolidator application
  - consolidatormake.xml, an Ant build file
  - com.actuate.consolidator, the log application source code

### How to install the log consolidator application

1 Edit the following settings in consolidatorconfig.xml:

- JDBC driver name
- URL, specifying the type of JDBC driver and database connection information, including host name, port, and database instance (SID) or service name, using the following syntax:  
`jdbc:oracle:thin:@// [HOST] [:PORT] [:SID/SERVICE]`
- Database login information, including schema, user name and password
- Refresh interval

Measured in seconds. The default value is 10 seconds.

You may need to change the refresh interval depending on the amount of logging your system performs. The log consolidator application commits transactions to the database every 10 seconds or when the number of transactions exceeds 80, whichever occurs first.

- Usage and error log settings:

- Log file names
  - Number of log files for each type of file

The log file names and number of files must match the actual BIRT iHub configuration.

If you change a BIRT iHub setting, you must also change the corresponding consolidator configuration setting and restart BIRT iHub and the log consolidator application.

The following code example shows the default settings for consolidatorconfig.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LogConsolidator>
 <Database>
 <DriverName>oracle.jdbc.OracleDriver</DriverName>
 <URL>jdbc:oracle:thin:@dbsrv4-w2k:1521:Oran9i</URL>
 <Encoding>UTF8</Encoding>
 <Schema>Users</Schema>
 <DatabasePropertyList>
 <Properties>
 <Name>UserName</Name>
 <Value>actest</Value>
 </Properties>
 <Properties>
 <Name>Password</Name>
 <Value>systest</Value>
 </Properties>
 </DatabasePropertyList>
 </Database>
 <Consolidator>
 <RefreshInterval>10</RefreshInterval>
 <UsageLogEnabled>true</UsageLogEnabled>
 <ErrorLogEnabled>false</ErrorLogEnabled>
 <UsageLogProperties>
 <LogFileName>usage_log</LogFileName>
 <NumberOfLogFile>2</NumberOfLogFile>
 </UsageLogProperties>
 <ErrorLogProperties>
```

```
<LogFileName>error_log</LogFileName>
<NumberOfLogFiles>2</NumberOfLogFiles>
</ErrorLogProperties>
</Consolidator>
</LogConsolidator>
```

- 2** Copy the configuration file, consolidatorconfig.xml, to the following BIRT iHub directory:

\$AC\_SERVER\_HOME/etc

- 3** Install and run the startup and shutdown files after setting up the database:

- On a UNIX system, the following Actuate scripts start and stop the consolidator application:
  - start Consolidator.sh
    - Starts the log consolidator application. The script takes \$AC\_SERVER\_HOME as an argument and uses it to set the following path variables:
      - Configuration file
      - JAR file directory
      - CLASSPATH
  - stop Consolidator.sh
    - Stops the log consolidator application. The script takes \$AC\_SERVER\_HOME as an argument and uses it to read the log consolidator application PID from \$AC\_SERVER\_HOME/etc/consolidator.pid and kills the process. Add this script to BIRT iHub script stop\_srver.sh to stop the consolidator application whenever you stop the BIRT iHub.

- On a Windows system, the consolidatorwin.exe utility installs and removes the application as a Windows service, and starts and stops the consolidator application. The BIRT iHub installation process installs and runs the utility in the following directory:

\$AC\_SERVER\_HOME\bin

Use the consolidator.exe utility to install and configure the consolidator application. This utility assumes it is running in the BIRT iHub \bin directory.

The consolidatorwin.exe utility supports the following command line syntax:

```
consolidatorwin [-H/-?] [-SserviceType] [-UUserName]
[-Ppassword]
```

The command-line arguments specify the following options:

- ❑ -H requests help on usage
- ❑ -S specifies the following types of service:
  - ❑ auto adds the log consolidator as the Actuate Usage and Error Logging Consolidator 9 service that starts automatically when Windows restarts.
  - ❑ manual adds the log consolidator as the Actuate Usage and Error Logging Consolidator 9 service that requires manual startup when Windows restarts.
  - ❑ console starts the consolidator at a Windows command prompt.
  - ❑ remove stops the service.
- ❑ -U specifies the username  
The Windows user starting the consolidator application. Actuate recommends using the same user as the user that starts BIRT iHub.
- ❑ -P specifies the password  
The user password.

The following command adds the consolidator application as a Windows service that starts automatically when Windows starts:

```
consolidatorwin -Sauto -UUsername -PPassword
```

### **How to configure the log consolidator database**

- 1 Configure a database server machine, Oracle server, and database.
- 2 Using Oracle SQL\*Plus, log in as the system database administrator.
- 3 Run the CreateActuateLogTables.sql script.

The following command runs the CreateActuateLogTables.sql script from the install folder in the BIRT iHub Integration Technology installation for Windows:

```
SQL> @"C:\<BIRT iHub Integration Technology install folder>
 \ServerIntTech3\UsageAndErrorConsolidator\DBScripts
 \CreateActuateLogTables.sql";
```

CreateActuateLogTables.sql drops the ActuateLog and ActuateLogUser users, performs cascading deletes on all their objects, including sequences, tables, and

indexes, and recreates these objects. The script creates the following database objects:

- Tables to contain the usage and error log data  
INSERT statements add predefined codes and descriptions for the event, file, job, object, operation, output format, service, and status types, after creating the tables.
- Sequence generators to provide the values for usage and error event IDs when inserting log data  
Each usage and error event ID is a unique value.
- Indexes to contain primary and foreign key columns in the tables:
  - Primary key constraints on columns containing predefined codes ensure that these values are unique and not null.
  - Foreign key references on columns containing predefined codes ensure that these values are consistent with the values in the primary keys.

The ActuateLog schema contains the following tables and indexes:

- AcAdminEvent  
Contains the log records for administration operation events, including the following data:
  - Event ID
  - Object type code, indicating a User, Role, Channel, Group, File, or Folder object
  - Object operation code, indicating a Create, Delete, Modify, Login, Logout, or Download operation
  - Object name, version name, size, and attribute
  - Old and new values

Table 20-7 shows the structure of the AcAdminEvent table.

**Table 20-7** Structure of the AcAdminEvent table

Column	Data type	Constraint	References	Key
EventId	INTEGER	AcApEv _AdEvId _Idx	AcEvent .EventId	Primary
ObjectTypeCode	INTEGER		AcObject Type .Object TypeCode	

**Table 20-7** Structure of the AcAdminEvent table

Column	Data type	Constraint	References	Key
ObjectOperationCode	INTEGER		AcObject Operation .Object Operation Code	
ObjectName	VARCHAR2(1000)			
ObjectVersionName	VARCHAR2(255)			
ObjectSize	INTEGER			
ObjectAttribute	VARCHAR2(50)			
OldValue	VARCHAR2(2000)			
NewValue	VARCHAR2(2000)			

- **AcApplicationEvent**

Contains the log records for application events, including the following data:

- Event ID
  - Executable name
  - Executable version, indicating an RPTDOCUMENT, RPTDESIGN, or other file type
  - Job type code, indicating an Async, Persistent, and Transient job type
  - Resource group ID
  - Dispatch node, indicating the volume, system, and server
  - Output format code, indicating PDF, XLS, HTML, or other output format
- Table 20-8 shows the structure of the AcApplicationEvent table.

**Table 20-8** Structure of the AcApplicationEvent table

Column	Data type	Constraint	References	Key
EventId	INTEGER	AcApEv _ApEvId _Idx	AcEvent .EventId	Primary
ExecutableName	VARCHAR2(1000)			
ExecutableVersion	VARCHAR2(100)		AcFileType .FileTypeCode	
FileTypeCode	INTEGER			

(continues)

**Table 20-8** Structure of the AcApplicationEvent table (continued)

Column	Data type	Constraint	References	Key
Parameters	VARCHAR2(2000)			
JobName	VARCHAR2(100)			
JobTypeCode	INTEGER		AcJobType .JobTypeCode	
JobSubmittedTimestamp	DATE			
ResourceGroupId	INTEGER		AcResource Group .Resource GroupId	
DispatchNode	INTEGER		AcSystem Component .System ComponentId	
RequestId	VARCHAR2(100)			
RequestWaitTime	INTEGER			
RequestRunningTime	INTEGER			
OutputName	VARCHAR2(1000)			
OutputVersion	VARCHAR2(100)			
OutputFormatCode	INTEGER		AcOutput Format .Output Format Code	
OutputSize	INTEGER			
PageCount	INTEGER			
PageNumbersViewed	VARCHAR2(50)			

- AcErrorEvent

Contains the log records for error events, including the following data:

- Error event ID
- System component ID, indicating the volume, system, and server
- User name
- Error code, category, severity, parameters, and message

Table 20-9 shows the structure of the AcErrorEvent table.

**Table 20-9** Structure of the AcErrorEvent table

Column	Data type	Constraint	References	Key
ErrorEventId	INTEGER	AcErEv _ErEvId _Idx	AcEvent .EventId	Primary
EventTimestamp	DATE			
SystemComponentId	INTEGER		AcSystem Component .System ComponentId	
UserName	VARCHAR2(255)			
ErrorCode	INTEGER			
ErrorCategory	VARCHAR2(255)			
ErrorSeverity	INTEGER			
ErrorParameter1	VARCHAR2(50)			
ErrorParameter2	VARCHAR2(50)			
ErrorParameter3	VARCHAR2(50)			
ErrorMessage	VARCHAR2(255)			

- AcErrorLogOffset
  - Contains the following error log data:
    - File offsets
    - Volume names
    - Last update timestamp

Table 20-10 shows the structure of the AcErrorLogOffset table.

**Table 20-10** Structure of the AcErrorLogOffset type

Column	Data type/Values	Constraint	References	Key
FileIndex	NUMBER			
FileOffset	NUMBER			
VolumeName	VARCHAR2(50)	NOT NULL		Primary
LastUpdateTimeStamp	NUMBER			

- AcEvent

Contains the log records for events, including the following data:

- Event ID
- Event timestamp
- System component ID, indicating the volume, system, and server
- Event type code, indicating a Generate, Print, View, Delete, Admin, Query, or Search event type
- Start and end timestamps
- Status code, indicating Success or Failure
- Service type code, indicating a Factory, View, Encyclopedia, Integration, or Cache service type

Table 20-11 shows the structure of the AcEvent table.

**Table 20-11** Structure of the AcEvent table

Column	Data type	Constraint	References	Key
EventId	INTEGER	AcSt_StCo_Idx		Primary
EventTimestamp	DATE			
SystemComponentId	INTEGER		AcSystem Component .SystemComponent Id	
UserName	VARCHAR2(50)			
EventTypeCode	INTEGER		AcEventType .EventTypeCode	
StartTimestamp	DATE			
EndTimestamp	DATE			
StatusCode	INTEGER		AcStatus .StatusCode	
ServiceTypeCode	INTEGER	EventId	AcServiceType .ServiceType Code	

■ **AcEventType**

Contains the codes and descriptions for event types, including the following data:

- Event type code, including the following predefined values:  
1 through 7

- Event type description, including the following predefined values:  
Generate, Print, View, Delete, Admin., Query, Search
- Table 20-12 shows the structure of the AcEventType table.

**Table 20-12** Structure of the AcEventType table

Column	Data type/Values	Constraint	References	Key
EventTypeCode	INTEGER1-7	AcEvTy_EvTyCo_Idx		Primary
EventTypeDescription	VARCHAR2(50)			

- AcFileType  
Contains the codes and descriptions for the following file type data:
  - File type codes, including the following predefined values:  
1 through 39
  - File types, including the following predefined values:  
UNKNOWN, DOX, DCD, HTM, HTML, ICD, IOB, PDF, ROL, ROP, ROS, ROV, ROW, RPTDESIGN, RPTDOCUMENT, RPTLIBRARY, RPTTEMPLATE, RTF, SMA, TXT, XLS

Table 20-13 shows the structure of the AcFileType table.

**Table 20-13** Structure of the AcFileType table

Column	Data type/Value	Constraint	References	Key
FileTypeCode	INTEGER1-39	AcFiTy_FiTyCo_Idx		Primary
FileType	VARCHAR2(20)			

- AcJobType  
Contains the codes and descriptions for the following job type data:
  - Job type codes, including the following predefined values:  
1 through 3
  - Job type descriptions, including the following predefined values:  
Async, Persistent, Transient

Table 20-14 shows the structure of the AcJobType table.

**Table 20-14** Structure of the AcJobType table

Column	Data type/Value	Constraint	References	Key
JobTypeCode	INTEGER1-3	AcJoTy_JoTyCo_Idx		Primary
JobTypeDescription	VARCHAR2(20)			

- AcObjectOperation
 

Contains the codes and descriptions for the following object operation data:

    - Object operation codes, including the following predefined values:  
1 through 6
    - Object operation descriptions, including the following predefined values:  
Create, Delete, Modify, Login, Logout, Download
- Table 20-15 shows the structure of the AcObjectOperation table.

**Table 20-15** Structure of the AcObjectOperation table

Column	Data type/Value	Constraint	References	Key
ObjectOperationCode	INTEGER	AcObOp_ObOpCo_Idx		Primary
ObjectOperationDescription	VARCHAR2(20)			

- AcObjectType
 

Contains the codes and descriptions for the following object type data:

    - Object type codes, including the following predefined values:  
1 through 6
    - Object type descriptions, including the following predefined values:  
User, Role, Channel, Group, File, Folder
- Table 20-16 shows the structure of the AcObjectType table.

**Table 20-16** Structure of the AcObjectType table

Column	Data type/Values	Constraint	References	Key
ObjectTypeCode	INTEGER	AcObTy_ObTyCo_Idx		Primary
ObjectTypeDescription	VARCHAR2(20)			

- AcOutputFormat
 

Contains the codes and descriptions for the following output format data:

  - Output format codes, including the following predefined values:  
1 through 42
  - Output format descriptions, including the following predefined values:  
UNKNOWN, PDF, XLS, ROW, DHTML, HTML, HTM, RTF, REPORTLET,  
XMLDISPLAY, XMLCOMPRESSEDDISPLAY, DHTMLRAW, DHTMLLONG,  
CSS, ANALYSIS, EXCELDISPLAY, EXCELEDATA, EXCELDATADUMP,  
RTFFULLYEDITABLE, UNCSV, TSV, EXCEL, XMLDATADUMP,  
XMLREPORTLET, XMLCOMPRESSEDREPORTLET, XMLCOMPRESSEDEXCEL,

XMLCOMPRESSEDPDF, XMLCOMPRESSEDRTF, XMLSTYLE, XMLDATA,  
RPTDOCUMENT, RPTLIBRARY, RPTTEMPLATE

Table 20-17 shows the structure of the AcOutputFormat table.

**Table 20-17** Structure of the AcOutputFormat table

Column	Data type/Values	Constraint	References	Key
OutputFormatCode	INTEGER1-42	AcOuFo_OuFoCo_Idx		Primary
OutputFormat	VARCHAR2(30)			
Description				

- AcResourceGroup
  - Contains the codes and descriptions for the following resource group data:
  - Resource group ID, including the following predefined value:  
0
  - Resource group name, including the following predefined value:  
NULL

Table 20-18 shows the structure of the AcResourceGroup table.

**Table 20-18** Structure of the AcResourceGroup table

Column	Data type/Values	Constraint	References	Key
ResourceGroupId	INTEGER	AcReGr_ReGrId_Idx		Primary
ResourceGroupName	VARCHAR2(128)			

- AcServiceType
  - Contains the codes and descriptions for the following service type data:
  - Service type code, including the following predefined values:  
1 through 5
  - Service type description, including the following predefined values:  
Factory, View, Encyclopedia, Integration, Cache

Table 20-19 shows the structure of the AcServiceType table.

**Table 20-19** Structure of the AcServiceType table

Column	Data type/Values	Constraint	References	Key
ServiceTypeCode	INTEGER	AcSeTy_SeTyCo_Idx		Primary
ServiceTypeDescription	VARCHAR2(50)			

- AcStatus

Contains the codes and descriptions for the following status type data:

- Status codes, including the following predefined values:  
0 through 1
- Status descriptions, including the following predefined values:  
Failure, Success

Table 20-20 shows the structure of the AcStatus table.

**Table 20-20** Structure of the AcStatus type

Column	Data type/ Values	Constraint	References	Key
StatusCode	INTEGER	AcSt_StCo _Idx		Primary
StatusDescription	VARCHAR2(20)			

■ **AcSystemComponent**

Contains the log records for system components, including the following data:

- System component ID
- Volume, system, and server names

Table 20-21 shows the structure of the AcSystemComponent table.

**Table 20-21** Structure of the AcSystemComponent table

Column	Data type/ Values	Constraint	References	Key
SystemComponentId	INTEGER	AcSt_StCo _Idx		Primary
VolumeName	VARCHAR2(50)			
SystemName	VARCHAR2(50)			
ServerName	VARCHAR2(50)			

■ **AcUsageLogOffset**

Contains the following usage log data:

- File offsets
- Volume names
- Last update timestamp

Table 20-22 shows the structure of the AcUsageLogOffset table.

**Table 20-22** Structure of the AcUsageLogOffset type

Column	Data type/ Values	Constraint	References	Key
FileIndex	NUMBER,			
FileOffset	NUMBER			
VolumeName	VARCHAR2(50)	NOT NULL		Primary
LastUpdateTime	NUMBER			
Stamp				

The ActuateLog schema contains the following indexes listed in Table 20-23.

**Table 20-23** Indexes in the ActuateLog schema

Index	Table.Column(s)
AcApEv_AdEvId_Idx	AcAdminEvent.EventId
AcApEv_ApEvId_Idx	AcApplicationEvent.EventId
AcApEv_ExNa_Idx	AcApplicationEvent.ExecutableName
AcApEv_FiTyCo_Idx	AcApplicationEvent.FileTypeCode
AcApEv_JoTyCo_Idx	AcApplicationEvent.JobTypeCode
AcAdEv_ObNa_Idx	AcAdminEvent.ObjectName
AcApEv_OuNa_Idx	AcApplicationEvent.OutputName
AcAdEv_ObTyCo_ObOpCo_Idx	AcAdminEvent.ObjectTypeCode, ObjectOperationCode
AcErEv_ErCo_Idx	AcErrorEvent.ErrorCode
AcErEv_ErEvId_Idx	AcErrorEvent.ErrorEventId
AcErEv_ErSe_Idx	AcErrorEvent.ErrorSeverity
AcErEv_EvTi_Id	AcErrorEvent.EventTimestamp
AcErEv_SyCoId_Idx	AcErrorEvent.SystemComponentId
AcErEv_UsNa_Idx	AcErrorEvent.UserName
AcEv_EvId_Idx	AcStatus.EventId
AcEv_SyCoId_Idx	AcEvent.SystemComponentId
AcEv_StCo_Idx	AcEvent.StatusCode
AcEv_SyTyCo_Idx	AcEvent.ServiceTypeCode
AcEv_EvTi_Idx	AcEvent.EventTimestamp

(continues)

**Table 20-23** Indexes in the ActuateLog schema (continued)

Index	Table.Column(s)
AcEv_EvTyCo_Idx	AcEvent.EventTypeCode
AcEv_UsNa_Idx	AcEvent.UserName
AcEvTy_EvTyCo_Idx	AcEventType.EventTypeCode
AcFiTy_FiTyCo_Idx	AcFileType.FileTypeCode
AcJoTy_JoTyCo_Idx	AcJobType.JobTypeCode
AcObOp_ObOpCo_Idx	AcObjectOperation.ObjectOperationCode
AcObTy_ObTyCo_Idx	AcObjectType.ObjectTypeCode
AcOuFo_OuFoCo_Idx	AcOutputFormat.OutputFormatCode
AcReGr_ReGrId_Idx	AcResourceGroup.ResourceGroupId
AcSeTy_SeTyCo_Idx	AcServiceType.ServiceTypeCode
AcSt_StCo_Idx	AcStatus.StatusCode
AcSyCo_SyCoId_Idx	AcSystemComponent.SystemComponentId

# 22

## Actuate logging extension functions

This chapter provides an alphabetical list of the functions of the Actuate Usage Logging and Error Logging Extensions. It contains the following topics:

- About Usage Logging Extension functions
- About Error Logging Extension functions

## About Usage Logging Extension functions

The usage logging extension supports retrieving usage information that BIRT iHub captures and writing that information to a log file.

---

### AclsThreadSafe

Specifies whether the Usage Logging Extension is multithread-safe. If the extension is not multithread-safe, AclsThreadSafe must return False and BIRT iHub serializes all calls to the extension.

**Syntax** USAGELOGEXT\_API int AclsThreadSafe ( );

---

### AcLogUsage

Called by BIRT iHub for every transaction it logs.

**Syntax** USAGELOGEXT\_API void AcLogUsage (UsageInformation\* usagenfo);

**Parameter** **usagenfo**

The usage information that BIRT iHub captures.

---

### AcStartUsageLog

Specifies the path to the usage log file and the BIRT iHub to monitor. For example, if you write the usage information to a database, AcStartUsageLog provides a placeholder to open the database connection.

The server and cluster parameters are WideChar pointers. The function uses the data type WideChar for Unicode compatibility. Actuate pushes out all string data in UCS-2 format.

**Syntax** USAGELOGEXT\_API int AcStartUsageLog (const char\* serverHome, const WideChar\* server, const WideChar\* cluster);

**Parameters** **serverHome**

The path to the usage log file. The path must be \$AC\_SERVER\_HOME/bin.

**server**

The name of the BIRT iHub to monitor.

**cluster**

The name of the cluster of which the BIRT iHub is a member.

---

## AcStopUsageLog

Stops logging usage information and releases resources.

**Syntax** USAGELOGEXT\_API void AcStopUsageLog ( );

---

## About Error Logging Extension functions

This section provides an alphabetical list of the Error Logging Extension functions. Each entry includes a general description of the function, its syntax, and a description of its parameters.

The Error Logging Extension supports retrieving error information that BIRT iHub captures and writing that information to a log file.

---

## AclsThreadSafe

Indicates whether the Error Logging Extension is multithread-safe. If the extension is not multithread-safe, AclsThreadSafe must return False and BIRT iHub serializes all calls to the extension.

**Syntax** ERRORLOGEXT\_API int AclsThreadSafe ( );

---

## Ac.LogError

Called by BIRT iHub for every error it encounters.

**Syntax** ERRORLOGEXT\_API void Ac.LogError (ErrorInformation\* errorInfo);

**Parameter** **errorInfo**

The error information that BIRT iHub captures.

---

## AcStartErrorLog

Specifies the path to the error log file and the BIRT iHub to monitor. For example, if you write the error information to a database, AcStartErrorLog provides a placeholder to open the database connection.

The server and cluster parameters are WideChar pointers. The function uses the data type WideChar for Unicode compatibility. Actuate pushes out all string data in UCS-2 format.

## AcStopErrorLog

**Syntax**   ERRORLOGEXT\_API int AcStartErrorLog (const char\* serverHome,  
                                  const WideChar\* server, const WideChar\* cluster);

**Parameters**   **serverHome**

The path to the error log file. The path must be \$AC\_SERVER\_HOME/bin.

**server**

The name of the BIRT iHub to monitor.

**cluster**

The name of the cluster of which the BIRT iHub is a member.

---

## AcStopErrorLog

Stops logging error information and releases resources.

**Syntax**   ERRORLOGEXT\_API void AcStopErrorLog ( );

# Index

## Symbols

\_ (underscore) character 824  
; (semicolon) character 824  
: (colon) character 823, 865  
! (exclamation point) character 823  
? (question mark) character 824  
? wildcard character 566, 612  
. (period) character 824, 865  
, (comma) character 823  
" (double quotation mark) character 823  
{ } (curly brace) characters 861  
@ (at-sign) character 823  
\* (asterisk) character 823, 856  
\* wildcard character 567, 612  
/ (forward slash) character 823  
/ (slash) character 865  
\ (backslash) character 823  
& (ampersand) character 823  
# (number sign) character 823  
# wildcard character 567, 612  
% (percent) character 824  
+ (plus sign) character 824  
< (less than) character 823  
= (equals sign) character 823  
> (greater than) character 823  
\$ (dollar sign) character 823

## Numerics

3D charts 238

## A

ABIInfoObject element 724  
AbsoluteDate data type 700  
AbsoluteDate element 755  
AC\_JRE\_HOME variable 927  
AcAdminEvent table 932  
AcApplicationEvent table 933  
Accept directive 504  
AcceptEncoding element 812  
access control lists  
    applying 763  
    changing 902

creating 900  
deploying external 898  
getting templates for 646  
retrieving file or folder 645, 647  
retrieving users 899, 904  
storing externally 908  
translating 899  
access rights 327, 328, 769, 910  
    *See also* privileges  
access type attributes 735  
access types  
    getting 307, 316  
    job properties and 652  
    setting 310, 319  
accessing  
    Apache Axis client application 516  
    BIRT Interactive Crosstabs 390  
    BIRT Studio 865  
    BIRT Viewer 868  
    chart themes 110  
    cross tab elements 391  
    dashboard gadgets 50  
    HTML buttons 91  
    HTML5 charts 49  
    Information Console functionality 822  
    Interactive Crosstabs 38  
    Interactive Viewer 868  
    Java classes 877, 883  
    JavaScript API 38  
    JavaScript API class libraries 38, 39, 132  
    Microsoft .NET client application 600  
    plug-ins 504  
    proxy objects 502  
    report elements 23, 48  
    report viewer 50  
    reports 48  
    repository items 849  
    resources 42  
    RSSE sample applications 888  
    source code 38  
    third-party code libraries 517  
    web applications 846  
    web service applications 42

accessing (*continued*)  
  web services 492, 493, 498, 502  
  WSDL files 499

accessor methods 525

AccessRight element 769, 910

AccessRights element 772

accessToGrant parameter 858

AccessType element 735, 738, 763

AccessType property 648

AcChannelFilter variable 836

AcCreateUser class 554, 610

acDouble data type 701

AcDownloadFile\_Chunked class 588, 620

AcErrorEvent table 934

AcErrorLogOffset table 935

AcEvent table 935

AcEventType table 936

AcExecuteReport class 591

AcFilesFoldersFilter variable 849

AcFilesFoldersTypeFilter variable 849

AcFileType table 937

AcIsThreadSafe function  
  error logging extension 920, 945  
  usage logging extension 919, 944

AcJobType table 937

ACL element  
  GetFileACL operations 646  
  GetFileCreationACL operations 647  
  GetFileDetails operations 648  
   GetUserACL operations 904  
  NewFile type 763

ACL files 898, 899, 900  
  *See also* access control lists

ACL property 647

ACL templates 646

Ac.LogError function 920, 945

AcLogin class 532, 606, 609

AcLogUsage function 919, 944

ACLS. *See* access control lists

AcMail exceptions 926

acNull data type 701

AcObjectOperation table 938

AcObjectType table 938

AcOutputFormat table 938

AcResourceGroup table 939

AcRSSEPassThrough function 628

ACS. *See* Caching service

AcSelectFiles class 566, 612

AcSelectJavaReportPage class 592, 594

AcServiceType table 939

AcStartErrorLog function 919, 945

AcStartUsageLog function 919, 944

AcStatus table 939

AcStopErrorLog function 920, 946

AcStopUsageLog function 919, 945

AcSystemComponent table 940

action attributes (Ping) 669

Action Details window 105

action element 669

action parameter 854

actions 825, 828  
  completing 199  
  performing 106  
  triggering 103

Activate element 775, 781

Activate property 684, 691

Active Directory servers 888

active tab (dashboards) 152

active tabs (dashboards) 156

ActualHeadline element 753

ActualOutputFileDialog element  
  JobNotice type 748  
  JobProperties type 753  
  JobSearch type 758

ActualOutputFileName element  
  JobNotice type 748  
  JobProperties type 753  
  JobSearch type 758

ActualOutputFileSize element 748

Actuate API service 529

actuate class 38, 40, 132, 136

Actuate Java Components 41

Actuate namespace 132, 607

ActuateAPI class 535, 608, 609

ActuateAPI interface 529–530

ActuateAPI service 498

ActuateAPIEx interface 535

ActuateAPILocator class 529, 536

ActuateAPILocatorEx class 536

ActuateBuildNumber element 813

ActuateControl class 534  
  create user operations and 559, 560

ActuateController class 591, 593

ActuateLog schema 932, 941

ActuateSoapBindingStub class 528  
ActuateSoapPort interface 528, 605  
ActuateSoapPort\_address attribute 529  
ActuateVersion element 813  
AcUploadFile class 574, 618  
AcUploadFile.java 578  
AcUsageLogOffset table 940  
acviewer container 391  
ad hoc parameters  
    defining 765  
    generating query output and 213  
    testing for 207, 217  
ad hoc queries 213  
AddArchiveRules element 791  
addAttachmentPart function 577  
addAttribute function 448  
AddChannelNotificationById element 795  
AddChannelNotificationByName element 794  
AddChildRolesById element 800  
AddChildRolesByName element 799  
AddDependentFilesById element 790  
AddDependentFilesByName element 790  
addDimension function 123, 412  
AddFileCreationPermissions element 802  
AddGroupNotificationById element 795  
AddGroupNotificationByName element 794  
adding  
    bookmarks 48, 715  
    chart titles 238  
    cross tabs 411  
    custom security adapters 43, 876, 877–878, 879  
    dashboards 50, 146  
    data cubes 121  
    data items 242  
    data object data sources 22, 23  
    data service components 76, 179  
    data sets 24  
    data sorters 126, 176, 472  
    display names. *See* display names  
    file types 719, 738  
    filter conditions 163, 438, 723  
    filters 106, 163  
    Flash objects 246, 250  
    folders 720, 836  
    HTML buttons 91  
HTML5 charts 49, 255  
interactive features 103, 104, 124, 338  
label elements 281  
licensing options 802  
output messages 560  
page breaks 462, 465  
parameter components 66, 184  
parameter groups 210, 221, 231  
passwords 223  
privilege filters 326  
rendering options 364  
report components 136  
report elements 358  
Report Explorer components 56, 297  
report viewer components 44, 334  
scroll panels 366  
sort conditions 176, 472  
standard charts 234  
tables 285  
text elements 293  
upload security adapters 882, 883–885  
URI parameters 823, 824  
URL parameters 42  
users  
    tutorial for 557  
users to Encyclopedia 721  
users to volume 554, 571  
users to volumes 610, 615  
web pages 38, 39  
addLevel function 425  
AddLicenseOptions element 802  
addMeasure function 123, 413  
addMember function 431, 457  
AddOutputFilePermissions element 795  
addParameter function 575  
AddParentRolesById element 800  
AddParentRolesByName element 799  
addPoint function 269, 270  
AddRequiredFilesById element 790  
AddRequiredFilesByName element 790  
addressing e-mail 559  
addSeries function 256, 262  
AddToGroupsById element 801  
AddToGroupsByName element 801  
addTotal function 445, 476  
AddUserNotificationById element 794  
AddUserNotificationByName element 794

addUsers function 571, 615  
Admin event type 921, 923  
Administrat element 626  
Administrat function 556  
Administrat objects 556  
Administrat operations 626, 701  
    *See also* administration operations  
Administrat requests 555, 556, 570, 611, 615  
Administrat responses 557, 572  
administration applications 492, 553, 610  
administration operation classes 570, 615  
administration operations  
    adding users and 557  
    containing multiple requests 570, 615  
    creating users and 721  
    defining batch operations and 570, 614  
    defining transaction operations and 570,  
        614, 785  
    developing 553, 610, 626, 701  
    handling exceptions for 556, 616  
    running composite 572, 616  
    running single 556  
    viewing logging entries for 932  
administrative events 923, 932  
Administrator element  
    ExternalTranslatedUserGroupNames  
        type 733  
    ExternalTranslatedUserNames type 732  
    TranslatedRoleNames type 910  
        TranslatedUserNames type 911  
administrator names 911  
Administrator user groups 732, 733  
administrators  
    *See also* administration operations  
    creating Login requests for 690  
    managing iHub services and 641, 643  
    managing iHub System and 509  
AdminOperation arrays 556, 611, 616  
AdminOperation class 555  
AdminOperation element 626  
AdminOperation operations 701  
AdminOperation requests 555, 570, 611, 615  
AdminRights element 668  
advanced sort feature 373  
ageDays parameter  
    execute report page 842  
    submit job page 858  
ageHours parameter  
    execute report page 842  
    submit job page 859  
aggregate data. *See* aggregation  
aggregate functions. *See* aggregation  
    functions  
aggregation  
    building cross tabs and 118, 125  
    building data cubes and 120  
    enabling or disabling 374  
Aggregation data type 703  
aggregation functions  
    adding totals and 125  
    changing 452  
    getting 452, 480  
    setting 454, 481  
aging intervals 842, 858  
aging rules. *See* archiving rules  
AIS. *See* Integration service  
alerts 183  
Alias element 717  
All element 733, 773, 910  
ALL logging level 889  
All user groups 733  
AllowExport element 717  
AllowViewTimeParameter element 740  
alphaNumericSorterSet array 172  
analyzing data 120, 390  
animation (charts) 270  
Apache Ant utility 516, 890  
Apache Axis client application 516, 530, 534  
Apache Axis code libraries 516, 517  
Apache Axis environments 493, 496  
Apache Axis Java-based framework 577  
Apache Axis Javadocs 517  
Apache Axis TCPMonitor utility 531  
Apache log4j utility 517, 889  
Apache Logging Services Project 889  
Apache Tomcat servers 42  
api package 3  
appContext objects 11  
appendToJobStatus method 14, 15  
application context objects 11  
application context roots 822  
application events 933  
application programming interfaces  
    iHub environment 10

application programming interfaces (APIs)  
*See also* Information Delivery API  
data objects and 22  
RSSE applications and 888, 889  
system information and 916

application servers 41, 42

application services 40

application/dime media type 504

application/soap media type 504

applications 868  
accessing 846  
accessing client 516, 600  
accessing RSSE sample 888  
building security 888, 899, 901  
building user interfaces for. *See* user interfaces  
compiling code for 25  
consolidating log information and 926  
creating proxy objects for 502  
creating users and 557, 563  
deploying 874  
developing 10  
developing administration 553, 610  
developing batch or transaction 570, 614  
developing IDAPI. *See* IDAPI applications  
developing login 532, 533, 536, 606  
developing RSSE. *See* RSSE applications  
developing web. *See* web applications  
displaying data and 76, 77, 120  
displaying reports and 39, 44, 48, 66  
establishing connections to 40, 42, 43  
extracting subsets of data for 49  
getting version information for 138  
installing RSSE sample 898, 899  
integrating with BIRT iHub 492, 493  
integrating with iHub System 600  
integrating with web services 493  
loading class libraries from 38  
logging out of 142  
managing users and 897  
providing security for 41–43  
retrieving browser instance for 17  
retrieving volume names for 18  
running sample 517, 530  
sharing authentication information for 42  
testing 564  
testing connections for 140

uploading files and 577, 585  
applyFilter parameter 848, 850  
applyOptions function 266, 413  
arc function 275  
archive application 665  
archive files  
*See also* jar files; war files  
archive policies 843, 859  
archive rules. *See* archiving rules  
archive service command 665  
archive service errors 925  
archive settings 794  
archiveBeforeDelete parameter  
execute report page 843  
submit job page 859  
ArchiveLibrary element 665  
ArchiveLibrary property 664  
ArchiveOnExpiration element 703  
ArchiveOnExpire element 745  
archivePolicy parameter  
execute report page 843  
submit job page 859  
ArchiveRule data type 703, 815  
ArchiveRule element 763  
ArchiveRule objects 706  
*See also* archiving rules  
ArchiveRule property 692  
ArchiveRuleInherited element 745  
ArchiveRules element 648  
ArchiveRules property 574, 618, 647  
ArchiveServiceCmd element 665  
archiving operations  
getting schedules for 665  
scheduling 804  
setting parameters for 794  
starting 635  
stopping 635  
testing for 814  
archiving report objects 843, 859  
archiving rules  
getting 647  
inheriting 703  
removing 791  
setting attributes of 703, 706  
setting file specific 574, 618, 692  
area charts 263  
Argument data type 703

arguments. *See* command line arguments; parameters  
Arguments class 533  
array definitions 703  
ArrayOfAggregation data type 705  
ArrayOfArchiveRule data type 706  
ArrayOfArgument data type 706  
ArrayOfAttachment data type 706  
ArrayOfBookMark data type 706  
ArrayOfCapabilities data type 706  
ArrayOfChannel data type 706  
ArrayOfChannelCondition data type 706  
ArrayOfColumnDefinition data type 707  
ArrayOfColumnSchema element 776  
ArrayOfComponent data type 707  
ArrayOfComponentIdentifier data type 707  
ArrayOfCounterInfo data type 707  
ArrayOfDataExtractionFormat data type 707  
ArrayOfDataFilterCondition data type 707  
ArrayOfDataRow data type 708  
ArrayOfDataSortColumn data type 708  
ArrayOfDate data type 708  
ArrayOfDocumentConversionOptions data type 708  
ArrayOfFieldDefinition data type 708  
ArrayOfFile data type 708  
ArrayOfFileCondition data type 567, 613, 708  
ArrayOfFileContent data type 709  
ArrayOfFileType data type 709  
ArrayOfFilterCriteria data type 709  
ArrayOfGroup data type 709  
ArrayOfGroupCondition data type 709  
ArrayOfGrouping data type 709  
ArrayOfInt data type 709  
ArrayOfILOCacheDBIndexConstraint data type 709  
ArrayOfJobCondition data type 710  
ArrayOfJobNotice data type 710  
ArrayOfJobNoticeCondition data type 710  
ArrayOfJobProperties data type 710  
ArrayOfJobScheduleCondition data type 710  
ArrayOfJobScheduleDetail data type 710  
ArrayOfLicenseOption data type 711  
ArrayOfLong data type 710  
ArrayOfMDSSInfo data type 711  
ArrayOfNameValuePair class 593  
ArrayOfNameValuePair data type 711  
ArrayOfParameterDefinition data type 711  
ArrayOfParameterValue data type 711  
ArrayOfPendingSyncJob data type 711  
ArrayOfPermission data type 711  
ArrayOfPrinter data type 712  
ArrayOfPrinterOptions data type 712  
ArrayOfPropertyValue data type 712  
ArrayOfRecord data type 712  
ArrayOfResourceGroup data type 712  
ArrayOfResourceGroupSettings data type 712  
ArrayOfResultSetSchema data type 712  
ArrayOfResultSetSchema element 640, 653  
ArrayOfRole data type 713  
ArrayOfRoleCondition data type 713  
ArrayOfRunningJob data type 713  
ArrayOfServerInformation data type 713  
ArrayOfServerResourceGroupSetting data type 713  
ArrayOfService data type 713  
ArrayOfString data type 713  
ArrayOfString parameter 569  
ArrayOfUser data type 714  
ArrayOfUserAndProperties element 905  
ArrayOfUserCondition data type 714  
ArrayOfUserGroup data type 714  
ArrayOfUserGroupCondition data type 714  
arrays  
    data type definitions and 703, 909  
    viewer objects and 67  
ascending sort order 176, 177, 474  
AssignedToUserId element 809  
AssignedToUserName element 809  
AssignedToUsersById element 799  
AssignedToUsersByName element 799  
AssignRolesById element 801  
AssignRolesByName element 801  
AssignUserGroupsByID element 802  
AssignUserGroupsByName element 801  
asynchronous commands (volumes) 635  
asynchronous jobs 672, 686, 774  
asynchronous mode 686  
asynchronous operations 106  
asynchronous resource groups 658, 774  
AsyncResourceGroupList element 658  
Attachment data type 714, 815

Attachment objects 580, 706  
AttachmentPart class 575  
attachments  
  chunked transfer encoding and 587  
  downloading 632  
  embedding in responses 581  
  embedding in SOAP messages 633  
  selecting specific pages as 678  
  sending components as 650  
  sending data as 575  
  sending files as 573, 736  
  sending reports as 634  
  sending to multiple locales 714  
  setting attributes of 714  
  setting content IDs for 574, 618  
  setting content type for 574, 618  
  setting HTTP version for 504  
  setting output formats for 689

AttachReportInEmail element  
  JobInputDetail type 746  
  SubmitJob operations 689  
  User type 805, 912  
  UserField type 807

attribute declarations 493

attributes  
  *See also* properties  
  binding definitions and 497  
  data type definitions and 496  
  HTTP headers and 504  
  SOAP requests and 510  
  WSDL elements and 525

Attributes element 650

authenticate function  
  actuate class 41, 43, 137  
  security adapter 875, 877, 879

Authenticate operations 902

authentication  
  accessing applications and 43, 874  
  accessing corporate networks and 874  
  accessing external security systems  
    and 888  
  customizing 5, 874, 877  
  issuing URIs and 826  
  loading resources and 42  
  logging in to Information Console  
    and 875, 876  
  logging in to web services and 43, 137

logging into iHub System and 532, 606, 902  
authentication application 888, 890  
authentication exceptions 143  
authentication IDs 15, 827  
  duration of 532, 606  
  getting system 507, 537  
  setting system 537

authentication information  
  prompting for 41  
  requesting 330  
  sharing 42  
  SOAP headers and 507  
  unloading 43, 142

authentication requests 143

AuthenticationException class 143

AuthenticationException objects 143

authexpired action 828

AuthId element  
  Login operations 532, 606, 667  
  SOAP headers and 507, 741  
  SystemLogin operations 691

AuthId variable 535, 608

AuthorizationIsExternal element 814

auto suggest delays 190

auto suggest threshold values 213, 219

auto suggestion list controls 220

auto suggestion lists 191, 206

autoarchive policies 843, 859

AutoArchiveSchedule element 665

AutoArchiveSchedule property 664

autoarchiving. *See* archiving operations

autosave feature 148, 152

autosuggest list controls 765, 766

AutoSuggestThreshold element 766

axes values (charts)  
  data points and 272  
  multiple series and 238  
  testing for 259, 263

axis labels (charts) 264

Axis servers. *See* Apache Axis environments

axis type values (cross tabs) 428

axis types (cross tabs)  
  changing 125, 429  
  getting 427, 446  
  setting 428, 429, 447

## B

bandwidth 210  
banner elements 847  
bar charts 263  
base class 38  
Base64Binary data type 715  
BasedOnFileId element 629  
BasedOnFileName element 629  
batch applications 570, 614  
batch file (Apache Axis) 516  
batch operations 313  
batch requests 570, 615  
beans. *See JavaBeans*  
BeanSerializer objects 526  
BETWEEN operator 163, 438, 845  
binary data 672  
binary files 864, 868  
binding definitions (WSDL) 497, 498  
binding element 494, 497  
BIRT APIs 10, 22  
BIRT Data Analyzer. *See Data Analyzer*  
BIRT design files 508, 509  
    *See also* design files  
BIRT Designer 90  
BIRT Designer Professional 121  
    accessing iHub API and 11  
    returning unexpected values 14  
BIRT engine 11  
BIRT iHub System  
    logging in to 536  
BIRT iHub system  
    developing administration applications for 553  
    logging in to 532, 606  
BIRT iHub. *See iHub System*  
BIRT report files 657  
    *See also* report files  
BIRT reports 592, 677  
    *See also* reports  
BIRT scripting api package 4  
BIRT Studio  
    accessing 865  
    logging in to 865  
    starting 864  
BIRT Studio servlet 864

BIRT Viewer 38  
    accessing 868  
    linking to 868  
    navigating through reports and 868  
    *See also* report viewer  
BIRT Viewer servlet 868, 869  
BIRT Viewer view properties 593  
BIRT\_HOME variable 22  
BirtException class 29  
bln element 723, 787  
BookMark data type 715  
bookmark names 170, 281, 293  
Bookmark objects 706  
bookmark parameter 76  
BookmarkList element 637  
bookmarks  
    accessing cross tabs and 391, 401, 416, 467  
    adding Flash objects and 247, 341, 359  
    creating 48, 715  
    displaying report elements and 45, 76, 171, 345  
    displaying Reportlets and 344, 351  
    getting chart instance for 338, 358  
    getting chart names for 236  
    getting data item for 242, 341, 359  
    getting gadgets associated with 251, 343, 360  
    getting labels associated with 281, 343, 360  
    getting page numbers for 655, 656  
    getting report content for 339, 341, 344  
    getting text element for 293, 344, 361  
    navigating through reports and 347  
    retrieving from designs 637  
    retrieving result sets and 169  
    returning table objects for 106, 286, 344, 360  
    returning viewer objects for 345  
    sending requests for 170, 171  
BookMarkType element 715  
BookMarkValue element 715  
Boolean data type 715, 725, 765  
Boolean element 725, 779  
Boolean expressions 163, 164, 438, 439  
Boolean parameter 779  
Boolean values 787  
bottom N filter feature 384

BOTTOM\_N operator 163, 438  
BOTTOM\_PERCENT operator 163, 438  
browse file page 825, 834, 836  
browse page. *See* browse file page  
browsefile action 828, 829  
browseportletfile action 829  
BrowserPanel class 356  
browsers. *See* web browsers  
browsing 836  
build files (RSSE) 890  
build numbers 783, 813  
build paths 11  
build tool (Apache Axis) 516  
buildACL element 899, 900  
building  
    Java classes 525  
bullet gadgets 253  
bundling report files 634, 689, 745  
Business Intelligence and Reporting Tools.  
    *See* BIRT  
button constants (navigation) 188  
button controls 765  
button elements 91, 188, 220  
button events 91  
byte array input streams 594

## C

C/C++ applications 918, 919  
C# applications 502, 600, 603  
cache  
    open security 796  
    SOAP messages and 505  
    transient reports 642  
cache database. *See* Caching service database  
Cache Timeout property 897  
Cache-Control directive 505  
Caching element 783  
Caching service 669, 783, 926  
calculated columns 374  
    *See also* computed columns  
Call parameters 589  
callback functions  
    authenticating users and 42, 43  
    closing Interactive Crosstabs and 405  
    connecting to web services and 42, 140  
    defined 38

displaying dashboards and 51  
displaying data and 77  
displaying reports and 44, 57, 59  
downloading parameters and 185  
filtering data and 49  
handling exceptions and 143, 145, 181  
hiding report elements and 48  
retrieving parameters and 67, 186, 196  
retrieving result sets and 77, 174, 179, 338  
sorting data and 50  
callback parameter 137, 140  
CallOpenSecurityLibrary operations 628, 905  
canceljob action 828, 829  
CancelJob operations 628  
canceljob page. *See* request drop page  
CancelJobStatus data type 715  
cancelled jobs 680  
cancelreport action 829  
CancelReport operations 628  
Capabilities data type 716  
Capabilities element  
    GetCapabilities operations 637  
    GetCapabilitiesByCategory  
        operations 638  
    GrantUserGroupCapabilities  
        operations 665  
    Login operations 668  
    RevokeUserGroupCapabilities  
        operations 674  
capabilities list 638  
Capabilities objects 706  
CapabilityCategories element 667  
CapabilityNames element 716  
cascading parameter names 213, 219, 766  
cascading parameters  
    changing 199  
    getting values 203  
    testing for 207  
CascadingGroupName element 656  
CascadingParentName element 766  
case sensitivity 825, 858, 868  
    file names 676  
    iHub schemas 493  
    passwords 804  
    servlet names 864  
    user names 804

Categories element 637, 638  
Categories list 637, 638  
Category element 638, 716  
category series (charts)  
  drilling through 235  
  getting maximum value of 257  
  getting minimum value of 258  
  setting value range for 260  
categoryData variable 106  
cbFail parameter 848  
cbSuccess parameter 848  
cell attributes 722  
Cell element 723  
cells (empty) 414, 462, 464  
changeMeasureDirection function 125, 414  
changes, undoing or redoing 385  
ChangesPending element 780  
changing  
  access control lists 902  
  aggregation functions 452  
  chart subtype 240  
  chart themes 110  
  chart titles 235, 239  
  charts 49  
  cross tabs 122, 124, 391  
  data 375  
  data cubes 122  
  data series 259  
  file or folder privileges 791  
  gadget types 377  
  iHub server configurations 780  
  label elements 284  
  logging configurations 929  
  parameters 184, 199, 210  
  passwords 804  
  report designs 396  
  report output 66  
  reports 376  
  servlets 864  
  tables 48  
  text elements 48, 383  
  user interface options 48  
  viewer servlets 868  
  volumes 626  
Channel data type 716  
ChannelCondition data type 716  
ChannelField data type 716  
channelID parameter 837  
channelName parameter 838, 840, 848  
channels  
  deleting 726  
  removing notifications from 837  
  sending notifications to 848  
  updating 788  
Channels element 912  
ChannelSearch data type 716  
character codes (URLs) 865  
character data. *See* strings  
character encoding 823, 824  
character encoding. *See* encoding  
character encryption. *See* encryption  
character patterns  
  filter expressions 163, 164, 845, 846  
  search expressions 736, 743, 749, 806  
character sets 504, 824  
character strings. *See* strings  
character substitution 823  
characters  
  access control lists and 900  
  passwords and 804  
  search conditions and 736  
  SOAPAction directive and 505  
  text string limitations for 815  
  user names and 804  
charset attribute 504  
chart areas 103  
chart bookmarks 236, 338, 358  
chart builder 103  
Chart class 234  
chart dimension constants 238  
chart elements  
  adding 234, 255  
  determining type 237  
  displaying 240  
  getting bookmarks for 338, 358  
  hiding 238  
  setting size 239  
chart engine API 3, 5  
chart gadgets 253  
chart IDs 236, 237  
chart interactive features 103  
chart legends 103  
chart objects 234  
  *See also* charts

chart package 3  
chart properties  
  enabling or disabling 374  
chart subtypes 240, 374  
chart themes 110  
chart titles 235, 236, 238, 239  
chart types 263  
chart wizard launcher  
  *See also* chart builder  
CHART\_DIMENSION\_2D constant 238  
CHART\_DIMENSION\_2D\_WITH\_DEPTH constant 238  
CHART\_SUBTYPE\_PERCENTSTACKED constant 240  
CHART\_SUBTYPE\_SIDEBYSIDE constant 240  
CHART\_SUBTYPE\_STACKED constant 240  
charts  
  *See also* Flash charts; HTML5 charts  
  accessing themes for 110  
  adding interactive features to 103, 104  
  changing subtype of 240  
  changing titles for 235, 239  
  clearing data filters for 235  
  creating 234  
  developing. *See* charting APIs  
  displaying 240  
  drilling through 235, 236  
  getting bookmark name for 236  
  getting embedded 649  
  getting HTML element for 236  
  getting page associated with 237  
  rendering to specific output type 593  
  selecting subtypes for 374  
  setting filters for 239  
  setting number of dimensions for 238  
  setting title for 238  
  submitting requests for 240  
check boxes 220, 765, 849  
ChildUserId element 809  
ChildUserName element 808  
chunked attachments 504, 587, 619  
chunked messages 588, 594, 620  
circle function 276  
class definitions 29, 877  
class files 602, 889, 927  
class libraries 38, 132, 141, 877  
class names 525, 604, 739  
class reference 132, 135, 393  
classes  
  accessing auxiliary 533  
  accessing Java 877  
  accessing Java security extension 883  
  building mashup pages and 91  
  building RSSE applications and 889, 890  
  calling remote services and 528  
  connecting to web applications and 132  
  creating administration operations and 570, 615  
  creating login operations and 532, 609  
  creating proxy objects and 502  
  developing with 38, 391  
  downloading files and 587, 619  
  encoding/decoding SOAP messages and 516  
  executing reports and 591  
  generating C# 502, 603  
  generating code libraries for 493, 517, 600  
  generating data objects and 22  
  generating from WSDL types 526  
  generating Java 525  
  generating schemas for 516  
  implementing Actuate API service 529  
  Java event handlers and 10  
  mapping Java types and 526  
  setting search conditions and 566, 611  
  uploading files and 573, 617  
classpaths 11  
  Design Engine API applications 25  
clearFilter parameter 848  
clearFilters function  
  Chart class 235  
  FlashObject class 246  
  Gadget class 250  
  Table class 286  
  XTabAnalyzer class 414  
ClearSystemPrinters element 803  
client applications. *See* applications  
ClientChart class 255  
ClientChart objects 49, 255  
client-initiated requests 504  
ClientOption class 262  
ClientOption objects 49, 262  
ClientPoint class 266

ClientSeries class 269  
client-side error constants 195, 304  
client-side errors 181, 436  
ClientVersion element 667  
clipping rectangles 277  
clipRect function 277  
CloseInfoObject operations 629  
closing  
  HTTP sessions 43  
  information objects 629  
Cluster element 784  
cluster engine error messages 926  
cluster master failover errors 925  
cluster node lock violations 780  
cluster nodes 753, 925  
clusters  
  getting information about 661  
  installing log consolidator for 926  
  running logging extensions for 944, 945  
  sending SOAP requests to 502, 509  
  setting as system type 784  
  viewing error log entries for 925  
code  
  accessing for auxiliary classes 533  
  accessing JavaScript API 38  
  adding chart interactive features and 105  
  calling remote services and 528  
  compiling 25  
  constructing requests and 76  
  creating run configuration for 31  
  displaying cross tabs and 123, 127, 390  
  displaying dashboards and 51  
  displaying parameters and 67, 68  
  displaying reports and 45, 48  
  embedding 50  
  enabling user interface options and 48, 129  
  generating C# 603  
  generating data objects and 22  
  generating Java 516, 525  
  hiding user interface options and 129  
  initializing HTTP sessions and 40  
  initializing HTTPS sessions and 42  
  log consolidator application and 928  
  logging extensions and 919  
  logging information and 889  
  login classes and 609  
registering event handlers and 122  
RSSE applications and 889, 890  
running 91  
SOAP messages and 502, 525  
writing event handlers and 12, 13  
code emitter 516, 517, 524  
code libraries 493, 516, 517, 600  
collapse/expand feature 375  
Collation element 770  
CollationOption element 751, 771  
colon (:) character 865  
color printers 751, 771, 772  
ColorMode element 771  
ColorModeOptions element 771  
column editing feature 375  
Column element 724  
column headers  
  *See also* column names  
column headings 118  
  *See also* column names  
column index values  
  accessing result sets and 175  
  displaying cross tabs and 417  
  getting 175, 286, 368  
column mirror starting level  
  getting 461  
  setting 414, 460, 464  
column name parameter 106  
column names  
  displaying charts and 106  
  displaying parameters and 213, 227, 230, 363  
  filtering data and 165, 166  
  getting list of 170, 174  
  running queries and 219  
  setting aliases for 717  
  setting display names for 717  
  sorting data and 176, 177  
column schema objects 707  
column schemas 717, 724  
column types  
  parameter definitions and 213, 220  
  parameter value objects and 227, 230  
COLUMN\_AXIS\_TYPE constant 428  
ColumnDefinition data type 716  
ColumnDetail data type 717

columnMirrorStartingLevel parameter 414, 460  
ColumnName element  
    DataFilterCondition type 723  
    DataSortColumn type 724  
    ParameterDefinition type 766  
columnPageBreakInterval parameter 460  
columns  
    adding to cross tabs 461  
    changing data in 375  
    describing file 567, 613, 736  
    exporting 717  
    extracting data from 631  
    filtering values in 235, 723  
    getting 286, 417  
    hiding 48, 289, 717  
    matching top or bottom values in 163  
    moving 379  
    reordering 292, 381  
    resizing 375  
    retrieving data values in 175, 213  
    retrieving index values for 175, 286, 368  
    retrieving result sets and 77, 171, 175  
    selecting 368  
    setting page breaks on 462, 464, 465  
    setting size 815  
    setting type 717  
    showing calculated values in 374, 455  
    showing summary data in 125  
    showing table 291  
    sorting on 49, 176, 177, 631  
Columns element 630, 631  
ColumnSchema data type 717  
ColumnSchema objects 707  
ColumnType element 766  
comma-separated values files. *See* CSV files  
Command element 635  
command line arguments  
    duplicate names and 571, 615  
    log consolidator 931  
    login applications 533, 607  
    upload operations and 585  
command prompt  
    running applications from 563  
command status attributes 636  
commands (volumes) 635  
commercial model API JAR files 22  
commit function 396  
compiling 25  
Completed folder 907  
completed jobs 905, 907  
completed jobs page 848  
completion notices 673, 688, 912  
    *See also* notifications  
CompletionTime element  
    JobField type 744  
    JobNotice type 748  
    JobNoticeField type 749  
    JobProperties type 753  
Component element  
    CubeExtraction operations 630  
    DataExtraction operations 631  
    GetCubeMetaData operations 640  
    GetJavaReportEmbeddedComponent operations 650  
    GetJavaReportTOC operations 650  
    GetMetaData operations 653  
    GetPageNumber operations 656  
    SelectJavaReportPage operations 677  
component names (reports) 141  
ComponentIdentifier data type 717  
components  
    getting embedded 594, 649  
    getting page number for 656  
    logging information for 940  
    retrieving data from 631  
    sending as attachments 650  
ComponentType data type 718  
composite messages 502  
compound documents 587, 620, 632  
computed columns 453, 455  
    *See also* calculated columns  
computed measures 453, 455  
Concise mode (Ping) 670  
Condition element  
    FileSearch type 737  
    JobNoticeSearch type 750  
    JobScheduleSearch type 756  
    JobSearch type 757  
    UserGroupSearch type 808  
    UserSearch type 809  
ConditionArray element  
    FileSearch type 737  
    JobNoticeSearch type 750

ConditionArray element (*continued*)  
  JobScheduleSearch type 756  
  JobSearch type 758  
  UserGroupSearch type 808  
  UserSearch type 810  
conditions. *See* filter conditions; search conditions  
configuration error messages 926  
configuration template names 781  
configurations  
  changing 780  
  changing servlets and 864  
  firewalls and 875  
  iHub servers and 780  
  initiating actions and 828  
  IPSE applications and 879  
  LDAP servers and 891, 892, 897  
  log consolidator application and 928  
  log consolidator database and 931  
  resource groups and 659, 684  
  RSSE applications and 889, 890  
  upload security adapters and 884  
Connect action 669  
connection definition files. *See* database connection definition files  
connection exceptions 145  
connection handles. *See* ConnectionHandle element  
connection parameters 40, 139, 670  
connection properties  
  externalizing 814, 908  
  getting 638, 903  
  setting 683  
connection strings 671  
ConnectionException class 145  
ConnectionException objects 145  
ConnectionHandle element  
  CubeExtraction operations 631  
  DataExtraction operations 632  
  ExecuteReport operations 591, 635  
  GetJavaReportEmbeddedComponent operations 650  
  GetJavaReportTOC operations 651  
  GetPageCount operations 655  
  PendingSyncJob type 767  
  RunningJob type 777  
  SelectJavaReportPage operations 678  
SOAP headers and 508, 741  
WaitForExecuteReport operations 693  
ConnectionHandle variable 535, 608  
ConnectionProperties element  
  GetConnectionProperties operations 639, 903  
  Ping operations 670  
  SetConnectionProperties operations 684  
ConnectionPropertiesAreExternal element 814  
ConnectionPropertyExternal element 908  
connections  
  accessing Information Console and 41  
  accessing volumes and 41, 827, 855, 881  
  authenticating users and 41, 137  
  closing 142  
  configuring log consolidator 928  
  displaying dashboards and 153  
  getting properties for 638, 903, 908  
  handling errors for 145  
  loading JavaScript classes and 91  
  logging in to web applications and 42, 43, 179  
  opening 38, 40, 139  
  ping operations and 670  
  preserving 508, 777  
  protecting data and 874–875  
  setting properties for. *See* connection properties  
  testing 140  
ConnectionString property 671  
consolidator application. *See* log consolidator application  
Constants class 195, 304  
ContainedFiles element 633  
containerID parameter 51  
content components 44, 48, 56, 168  
Content element  
  DownloadFile operations 633  
  ExportParameterDefinitionsToFile operations 636  
  ExtractParameterDefinitionsFromFile operations 636  
  FileContent type 736  
  SelectFiles operations 676  
  UploadFile operations 692  
content panels (viewer) 356, 366, 370, 371

content type (messages) 504  
content variable 392, 574, 618  
ContentData element 715  
ContentEncoding element 714  
ContentEncoding property 650  
contentID attribute 574  
ContentId element 714  
ContentItemList element 676  
Content-Length directive 505  
ContentLength element 714  
ContentLength property 650  
Content-Type directive 504  
ContentType element 714, 739  
context menus 384  
context objects 11  
Context Path property 897  
context roots 822  
ContextPath property 593  
control commands (volumes) 635  
control type attributes 734, 765  
control type UI values 204  
control types 214, 220  
controls (HTML buttons) 91  
ControlType element 765  
conversion options 641, 686, 730  
ConversionOptions element 641, 690, 747  
convert function 196  
convertDate function 197  
ConvertUtility class 196  
cookies 826, 849  
CopyFile element 702, 786  
CopyFile operations 718  
CopyFromLatestVersion element 692  
CopyFromLatestVersion variable 574, 618  
copying  
    access rights 328  
    file properties 574, 618  
    Information Console directory  
        structures 61, 64, 72, 82, 119  
    JAR files 11  
CounterId element 719  
CounterIDList element 639  
CounterInfo data type 719  
CounterInfo objects 707  
CounterInfoList element 637, 639  
CounterName element 719  
counters 637, 639, 719

CounterValue element 719  
counting  
    items in folders 649  
    items in lists 656  
    pages in files 308  
    pages in reports 345, 399, 655  
    search results records 738, 758, 810  
CountLimit element  
    FileSearch type 738  
    GetFileACL operations 646  
    GetFileCreationACL operations 647  
    GetParameterPicklist operations 656  
    JobNoticeSearch type 750  
    JobScheduleSearch type 757  
    JobSearch type 758  
    UserGroupSearch type 809  
    UserSearch type 810  
create folder page 825, 834, 836  
CreateActuateLogTables.sql script 928, 931  
createCall function 575, 588  
CreateChannel element 785  
CreateChannel operations 719  
CreateDatabaseConnection operations 629  
CreatedByUserId element 646  
CreatedByName element 646  
CreateFileType element 702, 786  
CreateFileType operations 719  
createfolder action 829  
CreateFolder element 702, 786  
CreateFolder operations 720  
CreateGroup element 785  
CreateIOCache operations 629  
CreateNewVersion element 811  
CreateParameterValuesFile operations 629  
CreateQuery operations 630  
CreateResourceGroup operations 630  
CreateRole operations 721  
CreateUser application 557, 563  
CreateUser element 701, 785, 786  
createUser function 554, 555  
CreateUser operations 557, 610, 721  
CreateUserGroup element 701, 786  
CreateUserGroup operations 721  
creating  
    access control lists 900  
    auto suggestion lists 191  
    batch applications 570, 614

creating (*continued*)  
bookmarks 48, 715  
cross tabs 121, 411  
custom security adapters 43, 876, 877–878, 879  
dashboards 50, 146  
data cubes 24, 121  
data objects 22  
data service components 76, 179  
data sets 24  
data sorters 126, 176, 472  
data sources 23  
debugging messages 14, 15  
display names. *See* display names  
filter conditions 163, 438, 723  
filters 106, 163  
Flash gadgets 250  
Flash objects 246  
folders 720, 836  
HTML buttons 91  
HTML5 charts 49, 255  
IDAPI applications. *See* applications  
Java event handlers 10, 13  
JavaScript event handlers 10, 12  
job requests 686  
label elements 281  
LDAP configuration files 891, 892  
log consolidator tables 928, 932  
Login requests 690  
output messages 560  
parameter components 66, 184  
parameter groups 210, 221, 231  
passwords 223  
print jobs 672  
privilege filters 326  
report designs 25, 46, 52, 54, 69, 71, 119  
report explorer 56, 297  
report files 762  
report parameters 764  
requests 76  
resource groups 630, 774  
result sets 174  
scroll panels 366  
SOAP headers 507–509  
SOAP messages 496, 502, 503  
sort conditions 176, 472  
standard charts 234  
tables 285  
text elements 293  
transaction applications 570, 571, 614, 615  
upload security adapters 882, 883–885  
URIs 822, 823  
URL parameters 42  
user groups 807, 808  
users 554, 610, 721  
    tutorial for 557  
viewer components 44, 334  
web pages 38, 39  
web-based applications 132, 493  
WSDL schemas 493–499  
credentials  
    evaluating 879  
    external data sources 903  
    external users 42, 137, 667  
    iHub system 880, 902  
Credentials element 667, 902  
credentials parameter 137  
cross tab bookmarks 391, 401, 416, 467  
cross tab elements 122, 390, 391, 419  
cross tab filter objects 438  
cross tab gadgets 397, 404  
cross tab layout view 484  
cross tab objects 391, 411  
cross tab report elements 391, 411  
cross tab Reportlets 407  
cross tabs  
    accessing 390  
    adding dimensions to 123, 412, 425  
    adding measures to 123, 413, 452  
    changing 122, 124, 391  
    creating 121, 411  
    displaying 118, 122, 390, 467  
    drilling through 127, 415, 431  
    enabling interactive features for 124  
    filtering data in 414, 422, 438  
    getting bookmarks for 401, 416  
    getting columns in 417  
    getting empty cell values for 462  
    getting level values in 459  
    getting measure direction for 463  
    getting page breaks for 462, 463  
    getting rows in 418  
    handling errors for 435  
    handling events for 122, 402, 434

hiding detail data in 129, 419  
loading 390  
pivoting elements in 125, 419  
removing dimensions from 123, 419  
removing measures from 420  
removing summary data in 126  
rendering 409  
reordering elements in 125, 420, 421  
retrieving data for 121, 124, 417, 418  
selecting elements in 434  
setting display options for 413, 460  
setting empty cell values in 414, 464  
setting level values for 459  
setting measure direction for 465  
setting page breaks for 464, 465, 466  
sorting data in 49, 126, 422, 472  
submitting changes to 423  
switching measure direction in 414  
updating 127  
viewing detail data in 423  
viewing summary data in 125, 445, 476, 480

crossContext parameter 42  
cross-platform applications 492, 502  
Crosstab class 122, 411  
Crosstab objects 391, 411  
crosstab variable 392  
CSS files  
    *See also* cascading style sheets  
CSS formats 811  
CSS position attribute 399, 406  
CSV element 742  
CSV files 678, 916  
CSV formats 742  
cube view 485  
CubeExtraction operations 630  
cubes  
    analyzing data and 120  
    changing 122  
    creating 24, 121  
    extracting data from 630  
    generating 23  
    getting metadata for 640  
    loading 390  
cur element 722, 787  
Currency data type 724, 764  
Currency element 724, 779

Currency parameter 779  
CurrentTransientReportTimeout element 642  
custom event web service 731  
custom events 721, 730, 732  
custom security adapters 43, 877  
CustomEvent data type 721  
CustomEvent element 730, 732  
customizing  
    auto suggestion lists 191  
    file verification 883  
    logging extensions 918, 919  
    security adapters 876–878, 879  
    upload security adapters 882–884  
    URL parameters 42, 137  
    user authentication 5, 874, 877  
    user interfaces 372, 484  
    user logins 5, 874  
    web pages 38  
cylinder gadgets 253

## D

da action 828  
daemonURL parameter 851  
Daily data type 722  
Daily element 755  
dashboard action 828  
Dashboard class 50, 146  
dashboard components 141  
dashboard definitions 147, 150, 156  
dashboard event constants 157  
dashboard files 50, 51, 657  
dashboard metadata 150  
dashboard names 148, 152  
dashboard objects 146  
dashboard page 834  
dashboard page fragment (HTML) 152  
dashboard tab names 156, 159, 161  
dashboard tab objects 161  
dashboard tab toolbar 154  
dashboard tab type constants 161  
dashboard tabs  
    constructing 161  
    determining gadget specific 160  
    getting active 156  
    getting array of 156  
    getting names 159, 161, 162

dashboard tabs (*continued*)  
    getting type 161  
    setting as active 152  
dashboard templates 148, 154  
DASHBOARD\_MAX constant 353  
DASHBOARD\_NORMAL constant 353  
DashboardDefinition class 156  
dashboards  
    accessing gadgets in 50  
    adding tabs to 161  
    changing gadgets in 158  
    closing 149  
    creating 50, 146  
    determining status of 402  
    displaying 50  
    downloading 147  
    getting active tab for 148, 156  
    getting content for 150  
    getting tabs in 156  
    handling events for 150, 157, 158  
    loading 50  
    saving 148, 149, 151  
    setting active tab for 151, 152  
    setting auto save delay for 152  
    setting size of 153, 154  
    setting viewing mode for 353  
    setting web service connection for 153  
    showing personal 155  
    showing tab toolbar in 154  
    submitting requests for 154  
    viewing cross tabs and 397  
    viewing gadgets and 160, 404

data  
    aggregating. *See* aggregation  
    analyzing 120, 390  
    changing 375  
    displaying 77, 460  
    downloading 168  
    extracting. *See* data extraction operations  
    filtering 772  
    hiding 129, 244, 419  
    localizing. *See* locales  
    preventing loss of 39  
    prompting for 210, 229, 232, 767  
    protecting corporate 874  
    restricting access to 874  
    returning from multiple sources 506  
    returning from web services 38, 76, 179  
    returning subsets of 49, 174  
    selecting 368  
    submitting requests for 169  
    summarizing 118  
    updating 208

Data Analytics module 392  
Data Analytics viewer 405  
Data Analyzer  
    adding toolbars. *See* Data Analyzer  
        toolbars  
    enabling or disabling 376  
data analyzer component 141  
    *See also* Data Analyzer  
data charts viewer  
    *See also* charts  
data cubes. *See* cubes  
data elements  
    *See also* data items  
    adding 23, 242  
    creating 25, 46, 52, 54, 69, 71, 119  
    getting 359  
data extraction format objects 707  
data extraction operations 376, 631, 640  
data fields 118  
    *See also* columns  
data filters. *See* filters  
data handlers 575  
data hierarchies (cubes) 121, 122, 125  
data item IDs 243  
data item objects 242  
data items 242, 243, 244  
    *See also* data  
data mart functions 23  
data object classes 22  
data object values files  
    exporting parameter definitions in 636  
    retrieving parameters in 636, 657  
    submitting jobs and 687  
data objects 22, 28  
data point arrays 272  
data point options 266, 269  
data points (charts) 266, 269  
data repositories 864  
    *See also* volumes  
        displaying content 56, 297  
    getting type 880

web services and 41  
data repository access rights 327, 328  
data repository file paths 59, 148  
data repository type constants 333  
data repository types 331, 333  
data rows. *See* rows  
data series (charts)  
    adding 256, 262, 269  
    changing 259  
    deleting 259, 270  
    displaying values 103  
    drilling through 235, 236  
    getting number of run-time 257  
    getting values for 257, 258  
    managing 269  
    removing 271  
    replacing 271  
    setting values for 260, 261  
    setting visibility of 259, 264, 272  
data series names 106  
data series objects 269, 270  
data service class 76, 90, 179  
data service components 76, 141  
data service objects 179  
data services 179  
data set fields. *See* fields  
data sets  
    adding 24  
    generating programmatically 23  
data sorters. *See* sorters  
data sources  
    combining data from multiple 506  
    creating 23  
    generating 23  
    specifying 581  
data streams 594  
data transfer protocols 492, 502  
data type definitions 495–496, 703, 909  
data type parameters 576  
data types  
    assigning to columns 717  
    building C# classes and 603  
    building JavaBeans and 525  
    computed columns 453, 455  
    defining 495, 695, 724  
    naming 495, 787  
    parameter definitions 213, 214, 220, 764  
report parameters 228, 231, 733  
RSSE applications 909  
setting cell specific 722  
specifying required sequence for 496  
DataACL element 690, 747  
database connection definition files 903  
database connection properties. *See*  
    connection properties  
database drivers. *See* drivers  
Database property 671  
database schemas. *See* schemas  
database types 671  
DatabaseConnectionDefinition data type 722  
DatabaseEnvironment property 671  
DatabaseList property 671  
databases  
    *See also* data sources  
    consolidating logging information  
        and 926, 928, 931  
    pinging 671  
    retrieving access control lists from 899  
    storing user information and 897  
DataCell data type 722  
DataExtraction operations 376, 631, 640  
DataExtractionFormat data type 723  
DataExtractionFormat objects 707  
DataExtractionFormats element 641  
DataExtractionRef element 631  
DataFetchHandle element 629  
DataFilterCondition data type 723  
DataFilterCondition objects 707  
DataItem class 242  
datamart functions 23  
DataRow data type 723  
DataRow objects 708  
DataRows element 742  
DataSchema data type 724  
DataSchema element 741  
DataService class 76, 90, 179  
DataSortColumn data type 724  
DataSortColumn objects 708  
DataSource property 671  
DataSourceType data type 724  
DataSourceType element 766, 767  
DataType data type 724  
DataType element  
    ColumnSchema type 717

**DataType** element (*continued*)  
    **FieldDefinition** type 733  
    **ParameterDefinition** type 764  
**Date** data type 725, 764  
**Date** element 724, 779  
Date objects 708  
date parameters 779  
date stamps 844  
date values 163, 197, 700, 755  
date-and-time values 787  
**DateOnly** element 725, 779  
**DateOnly** parameter 779  
**DatesExcluded** element 755  
**DateTime** data type 724, 725  
**dateToDelete** parameter  
    execute report page 843  
    submit job page 859  
**dbl** element 722, 787  
**dbn** element 722, 787  
**DBType** property 671  
**DCD.** *See* database connection definitions  
**DEBUG** logging level 889  
debugging  
    event handlers 14, 15  
    files 746  
debugging messages 14, 15  
**DebugInstruction** element 746  
Decimal data type 725  
decimal values 725  
**DecomposeCompoundDocument** element 632  
**DecomposeCompoundDocument** variable 587, 620  
decomposed documents 587  
default authentication 874  
default character set 504  
default encoding 824  
default iHub server URL 138  
default parameters 137  
default printer 772, 813  
default request options 138  
default resource group 632  
default settings 42  
default values  
    downloading 186  
    field definitions and 734  
    getting 215  
    overwriting 634  
    setting 221, 765  
default web service URL 138  
**DefaultEventLagTime** element 731  
**DefaultEventPollingDuration** element 731  
**DefaultEventPollingInterval** element 731  
**DefaultFailureNoticeExpiration** element 814  
**DefaultObjectPrivileges** property 908  
**DefaultOutputFileACL** element 652, 654  
**DefaultOutputFileACL** property 651, 654  
**DefaultPrinterName** element 805, 807, 813  
**DefaultPrinterName** property 803  
**DefaultSuccessNoticeExpiration** element 814  
**DefaultTableValues** element 766  
**DefaultValue** element 734, 765  
**DefaultValueIsNull** element 766  
**DefaultViewingPreference** element 813  
**DefaultViewingPreference** property 803  
definitions element 493, 494  
**DelayFlush** element 508, 741  
**DelayFlush** variable 535, 608  
delays 190  
delete events 920, 921  
delete file status page 834  
delete job page 825, 834, 837  
delete privilege 769  
delete status page 825, 834, 837  
**DeleteChannel** element 785  
**DeleteChannel** operations 726  
**DeleteDatabaseConnection** operations 632  
deletefile action 828, 830  
**DeleteFile** element 702, 786  
**DeleteFile** operations 726  
**DeleteFileType** element 702, 786  
**DeleteFileType** operations 727  
**DeleteGroup** element 785  
**DeleteGroup** operations 727  
**DeleteIOCache** operations 632  
deletejob action 828, 830  
**DeleteJob** element 702, 786  
**DeleteJob** operations 727  
deletejobnotice action 828, 830  
**DeleteJobNotices** element 702, 786  
**DeleteJobNotices** operations 728  
deletejobschedule action 828, 830  
**DeleteJobSchedule** element 702, 787  
**DeleteJobSchedule** operations 728

DeleteResourceGroup operations 632  
DeleteRole operations 728  
DeleteUser element 701, 785  
DeleteUser operations 729  
DeleteUserGroup element 701, 786  
DeleteUserGroup operations 729  
deleting  
  archived reports 844, 861  
  archiving rules 791  
  authentication information 43, 142  
  channels 726  
  data groups 289  
  data series 259, 270, 271  
  dimensions 123, 419  
  duplicate values 270  
  event handlers 122, 189, 402  
  file dependencies 811  
  file types 727  
  files 726  
  folders 726, 841  
  job schedules 728  
  jobs 727, 837, 841  
  measures 420  
  notification groups 727  
  notifications 728, 837  
  report files 841  
  resource groups 632  
  security roles 799  
  summary data 126  
  system printers 803  
  user groups 729  
  users 729, 788  
dependent files. *See* file dependencies  
DependentFileDialog element 737  
DependentFileName element 737  
deploying  
  applications 874  
  external access control lists 898  
  LDAP configuration files 891  
  reports 901–902  
  security adapters 878, 879, 884  
Deployment Kit 38, 41, 143  
descending sort order 176, 177, 474  
Description element  
  CreateFolder operations 720  
  ErrorMessage type 909  
  File type 734, 736  
  InstallApp operations 666  
  LicenseOption type 759  
  NewFile type 763  
  Printer type 769  
  ResourceGroup type 774  
  ServerInformation type 780  
  User type 804  
  UserField type 806  
  UserGroup type 807  
  UserGroupField type 808  
Description property  
  CopyFromLatestVersion element 692  
  ResourceGroup element 691  
  ResultDef element 648  
  UploadFile requests 574, 618  
descriptions (limits for) 815  
design configuration objects 22  
design engine 23  
Design Engine API applications 25  
Design Engine API extension 22  
design engine API package 3, 5  
design engine classes 25  
design files 869  
  accessing 492  
  getting parameters from 657  
  naming 350, 351, 407  
DesignConfig objects 22  
designs 868  
  committing changes to 396  
  creating 25, 46, 52, 54, 69, 71, 119  
  opening 866  
  running 66, 509  
  saving 381  
  saving viewer content as 348  
destination attributes (Ping) 668  
Destination element 668  
destination parameter 855  
DestinationURL element 666  
destroy function  
  ClientPoint class 267  
  ClientSeries class 270  
  HTML5Chart Renderer class 277  
detail data 129, 291, 419, 423  
detail pages 825, 834, 838  
developing  
  administration applications 553, 610  
  batch or transaction applications 570, 614

developing (*continued*)  
charts. *See* charting APIs  
data objects 22, 28  
IDAPI applications. *See* IDAPI applications  
login applications 532, 533, 536, 606  
RSSE applications. *See* RSSE applications  
web applications 10, 132, 493  
    web pages 38  
development environments 493, 502, 600  
development languages 492  
DHTML formats 811  
DHTMLPageCaching element 813  
DHTMLPageCaching property 803  
DHTMLPageCachingExpiration property 803  
DHTMLPageCachingExpirationAge element 813  
diagnostic information 668, 853, 855  
    *See also* Ping operations  
dialog boxes  
    displaying cross tabs and 390  
    exporting data and 353  
    loading 90, 141  
    printing reports and 354  
    viewing result sets and 353  
Dialog class 90  
dialog components 141  
dialog event constants 357  
Dimension class 122, 125, 425  
dimension index values 125, 421, 428, 430  
dimension names 426, 428  
dimension objects 425  
dimensions  
    adding levels to 425, 429, 448, 451  
    adding to cross tabs 123, 412, 425  
    changing axis type 125, 421, 429  
    creating data cubes and 120  
    drilling through 127, 415, 431  
    expanding or collapsing members in 127, 423  
    filtering 440, 442, 443  
    generating summary values for 125  
    getting level values in 459  
    getting levels 427, 441  
    hiding detail data in 129, 419  
    naming 428  
    removing 123, 419  
reordering 125, 420  
setting axis type for 428, 429  
setting index values for 428, 430  
sorting values 472, 473, 474  
viewing charts and 238  
DIRECTION\_HORIZONTAL value 460, 463, 465  
DIRECTION\_VERTICAL value 460, 463, 465  
DirectOnly element 637  
directories  
    accessing client applications and 516, 600  
    copying 61, 64, 72, 82, 119  
    creating download 594  
    creating folders for 720  
    installing log consolidator and 927  
    logging usage information and 917  
    preserving workspace 745  
    running RSSE applications and 888  
    searching 566, 612  
directory paths  
    Java event handlers and 11  
    temporary files 16  
directory paths. *See* paths  
Disabled element 774, 776  
Disabled property 691  
disableIV function 336  
disk space 855  
dispatch node (log consolidator) 933  
display formats. *See* formats  
display names  
    columns 717  
    parameter definitions 214, 215, 220, 221  
    parameter values 200, 228, 231  
    scalar parameters 733  
displayData function 77  
displayFilterIcon property 593  
displayGroupIcon property 593  
displaying  
    aggregate values 118  
    columns 291  
    cross tabs 118, 122, 390, 467  
    current jobs 856  
    dashboards 50  
    data 77, 460  
    data cubes 120, 485  
    data items 244  
    data series 259, 264, 272

dates 725  
debugging messages 14  
error messages 842  
failed jobs 848  
file properties 568  
Flash objects 246, 249, 254  
folders 302, 850  
HTML output 593  
HTML5 charts 273  
Interactive Crosstabs features 128  
label elements 283, 343  
login page 822, 850  
parameter groups 192  
PDF documents 593, 812  
pending jobs 857  
report elements 48  
report executables 850  
report items 378  
report parameters 67, 184, 210, 860  
Reportlets 344  
reports 44, 508, 868  
standard charts 240  
summary data 125, 445, 476, 480  
table elements 291  
table of contents 370, 371, 383  
tables 45, 106, 285  
text 294, 295, 344  
toolbars 383, 485  
tooltips 382

DisplayName element  
BookMark type 715  
FieldDefinition type 733  
ParameterDefinition type 765  
ParameterValue type 767

displayName element 717  
displayname variable 214  
displayReport function 59  
DisplayType element 739  
distributed iHub System. *See* clusters  
distribution location settings 794  
div tag 39, 44, 56, 390  
DllPath property 671  
DLLs 600, 916  
do directive 825  
do\_update page. *See* options page  
DOCTYPE tag 39  
document conversion options 641, 686, 730

document files 838, 850  
getting names 187  
naming 193, 350, 351, 407  
document output formats (Word) 337  
Document Type Definitions 868  
documentation ix, 276  
*See also* help collections  
DocumentConversionOptions data type 730  
DocumentConversionOptions objects 708  
documents  
*See also* reports  
downloading 587, 620, 632  
getting embedded components in 649  
getting parameters for 657  
getting table of contents for 650  
running Interactive Crosstabs and 390  
saving 348, 381  
viewing 868

DoesRoleExist operations 903  
DoesUserExist operations 903  
Domain element 509, 667  
domains 42  
Done element 732  
Double data type 725, 730, 764  
Double element 725, 779  
Double parameter 779  
double values 701, 730, 787  
.dov files. *See* data object values files  
download operations 632  
*See also* downloading  
download result set dialog 353  
downloadDashboard function 147  
DownloadEmbedded element  
DownloadFile operations 633  
ExportParameterDefinitionsToFile  
operations 636  
GetJavaReportEmbeddedComponent  
operations 650  
GetJavaReportTOC operations 650  
SelectJavaReportPage operations 678  
DownloadEmbedded variable 587, 620  
downloadfile action 828  
DownloadFile class 587, 619  
DownloadFile operations 587, 619, 621, 632  
DownloadFile requests 589, 590, 620  
DownloadFile responses 590

downloading  
  *See also* download operations  
binary files 864, 868  
compound documents 587, 620, 632  
dashboards 147  
data 168  
  report files 587, 619, 621, 632  
  report parameters 66, 67, 68, 185, 186  
reports 168, 337  
result sets 76, 179, 338, 353  
  WSDL files 601  
downloadParameters function 185  
downloadParameterValues function 67, 68, 186  
downloadReport function 337  
downloadResultSet function  
  DataService class 76, 179  
  Viewer class 337  
DownloadTransientFile operations 633  
drawing elements (Highcharts) 275  
drill function 127, 414  
drillDown function 415  
drillDownCategory function 235  
drillDownSeries function 235  
Driller class 431  
Driller objects 127, 431  
drilling 127  
drillUp function 415  
drillUpCategory function 235  
drillUpSeries function 236  
driver names 739  
DriverName element 739  
drivers  
  consolidating logging information and 927, 928  
  ping operations and 670  
  polling 746  
  running open server 669  
  setting timeout intervals for 745  
DriverTimeout element 745  
drop pages 825, 834, 841  
drop-down list controls 734, 765  
DroppedFromUsersById element 799  
DroppedFromUsersByName element 799  
DropRolesById element 802  
DropRolesByName element 801  
DropUserGroupsByID element 802

DropUserGroupsByName element 801  
DSTAMP variable 517  
DTD (Document Type Definition) syntax 868  
dtm element 722, 787  
Duplex element 751, 771, 772  
DuplexOptions element 771  
duplicate names 506, 721, 761  
duplicate requests 555  
duplicate values 270  
duplicating. *See* copying  
duplication suppression feature 382  
DurationSeconds element 744, 753  
dynamic filter parameters  
  converting values 196  
  defining 222  
  getting column names for 213, 227  
  getting column type for 213, 227  
dynamic filter queries 213  
dynamic filters 207  
dynamic link libraries 600, 916

## E

EasyScript 124  
Echo action 669  
editMeasure function 124, 416  
editTableRow action 830  
Effective element 639  
ElementFactory class 23  
elementFormDefault attribute 495  
elements. *See* report elements; XML elements  
ElementType element 715  
e-mail  
  addressing 559  
  directing to external users 907  
  sending attachments with. *See* attachments  
  sending notifications with 559  
  setting notification options for. *See* notifications  
EmailAddress element 804, 806, 911  
EmailAddress property 907  
EmailForm property 907  
EmailFormat element 689, 747  
EmailWhen property 907  
embeddedDownload variable 534  
EmbeddedObjPath element 812  
EmbeddedProperty element 784

EmbeddedRef element 650  
embedTemplate function 147  
empty cells 414, 462, 464  
emptyCellValue parameter 414, 460  
enableAdvancedSort function 373  
enableAggregation function 374  
EnableAutoParamCollection element 739  
enableCalculatedColumn function 374  
enableChartProperty function 374  
enableChartSubType function 374  
enableCollapseExpand function 375  
enableColumnEdit function 375  
EnableColumnHeaders property 678  
enableColumnResize function 375  
enableContentMargin function 375  
enableCrosstabView function 484  
enableCubeView function 485  
EnableCustomEventService element 731  
Enabled element 666  
enableDataAnalyzer function 376  
enableDataExtraction function 376  
enableEditReport function 376  
enableExportReport function 376  
enableFacebookComments function 377  
enableFilter function 377  
enableFilterSummaryView function 485  
enableFlashGadgetType function 377  
enableFormat function 377  
enableGroupEdit function 378  
enableHideShowItems function 378  
enableHighlight function 378  
enableHoverHighlight function 378  
enableIV function 338  
enableLaunchViewer function 379  
enableLinkToThisPage function 379  
enableMainMenu function 379  
enableMetaData property 593  
enableMoveColumn function 379  
enablePageBreak function 380  
enablePageBreak parameter 460  
enablePageNavigation function 380  
enableParameterPage function 380  
enablePrint function 380  
enableReorderColumns function 381  
enableRowResize function 381  
enableSaveDesign function 381  
enableSaveDocument function 381

enableShowToolTip function 382  
enableSort function 382  
enableSuppressDuplicate function 382  
enableSwitchView function 382  
enableTextEdit function 383  
enableTOC function 383  
enableToolBar function  
    Viewer.UIOptions class 48, 383  
    XTabAnalyzer.UIOptions class 485  
enableToolbarContextMenu function 384  
enableToolbarHelp function 384, 486  
enableToolbarSave function 486  
enableToolbarSaveDesign function 486  
enableToolbarSaveDocument function 487  
enableTopBottomNFilter function 384  
enableUndoRedo function 385  
encapsulation 39  
encode method 824  
encoding 39, 576, 823, 824  
encoding attribute 495  
encrypted tokens 667  
EncryptedPwd element 667, 804  
encryption 43  
encryption levels 690  
Encyclopedia engine 669  
Encyclopedia service 908  
Encyclopedia volumes  
    assigning resource groups to 774  
    defining attributes of 813  
    developing transaction operations for 785  
    naming 813  
    testing for 780  
    updating properties for 803  
end parameter 76  
EndTime element 773  
engine api package 3  
enterprise repository type 333  
EntityID element 666  
Envelope attribute 506  
environment variables 530, 927  
environments 10  
EQ operator 49, 106, 163, 438  
ERR\_CLIENT constant  
    parameter class 195  
    ReportExplorer class 304  
ERR\_CLIENT exception type  
    actuate.Exception class 181

ERR\_CLIENT exception type (*continued*)  
  ViewerException class 386  
  XTabAnalyzer.Exception class 435

ERR\_SERVER constant  
  parameter class 195  
  ReportExplorer class 304

ERR\_SERVER exception type  
  actuate.Exception class 181  
  ViewerException class 386  
  XTabAnalyzer.Exception class 435

ERR\_USAGE constant  
  parameter class 195  
  ReportExplorer class 304

ERR\_USAGE exception type  
  actuate.Exception class 181  
  ViewerException class 386  
  XTabAnalyzer.Exception class 435

error action 828

error callback functions 143, 145, 180

error codes 182, 436, 511, 783

error constants 195, 304

error descriptions 181, 435

error detail page 834, 838

error events 932, 934

error log consolidator 926

error log file names 916, 919

error log file types 916

error log files 919, 923, 924, 945

error log settings 929

error logging extension 916, 919, 945

ERROR logging level 889

error messages  
  BIRT iHub 925  
  displaying 842  
  error log files 924  
  Interactive Crosstabs 436  
  SOAP messages 509  
  uncategorized exceptions 182  
  viewer 387

error page 825, 835, 842

error parameters 925

error severity levels 924

errorcallback function 42, 43

errorCallback parameter 40, 138, 140

ErrorCode element 909

ErrorDescription element  
  CancelJob operations 628

CancelReport operations 629

ExecuteReport operations 635

GetSyncJobInfo operations 660

WaitForExecuteReport operations 693

ERRORLOG\_FILE\_EXT property 919

ERRORLOG\_FILENAME property 919

ErrorMessage data type 909

errors  
  Administrate operations and 556  
  download file operations and 621  
  duplicate element names and 506  
  failed SOAP requests and 511  
  iHub server status and 783  
  logging 924, 945  
  Login requests and 608  
  namespace declarations and 506  
  RSSE applications and 889, 909  
  upload file operations and 577, 618  
  viewing log file entries for 924, 925  
  web service requests and 40, 138, 140

escape characters 736

Event data type 730

event functions 91

event handlers  
  accessing iHub environment and 10, 11  
  creating HTML buttons and 91  
  creating interactive charts and 105  
  debugging 14  
  designing reports and 90  
  displaying cross tabs and 122, 402, 434  
  displaying dashboards and 150, 157, 158  
  displaying parameters and 189  
  displaying reports and 357  
  exceptions and 181  
  getting help for 10  
  navigating repository content and 58, 299, 305  
  registering 189, 402  
  removing 122, 189, 402  
  selecting report elements and 368  
  writing Java 10, 13  
  writing JavaScript 10, 12

event IDs 932

event model 10

event names 731

event status codes 731, 936

event type codes 921, 936

event type descriptions 937  
event types 731, 921, 936  
EventConstants class  
    Dashboard class 157  
    Parameter class 199  
    ReportExplorer class 305  
    Viewer class 357  
    XTabAnalyzer class 434  
EventName element 731, 754  
EventOptions data type 731  
EventParameter element 722, 754  
events  
    adding to HTML buttons 91  
    defining attributes of 730  
    defining options for 731  
    handling. *See* event handlers  
    interactive charts and 103  
    logging error information and 924, 936  
    logging usage information and 920, 921, 922  
    monitoring RSSE applications and 889  
    running Interactive Crosstabs and 122  
    running JavaScript API scripts and 90  
    running jobs and 743, 754  
    setting polling interval for 731  
    specifying custom 721, 732  
    viewing administration operation 932  
    viewing application 933  
    viewing error 934  
EventStatus element 731, 754  
EventType data type 731  
EventType element  
    Event type 731  
    JobProperties type 754  
    JobScheduleSearch type 757  
evt variable 106  
example .NET projects 600  
Excel formats 337, 689, 811  
Excel spreadsheets 364  
Exception class 122, 181, 435  
exception class definitions 29  
exception classes 143, 145, 386  
exception objects 181  
exception types  
    constructing exception objects and 181  
    getting 182, 437  
    testing for 182, 437

exceptions  
    authentication and 42, 43, 143  
    connections and 145  
    cross tabs and 122, 434, 435  
    data service components and 77  
    report parameters and 199  
    viewer and 357, 386  
executable file types 739  
executable files  
    accessing 600  
    attaching to responses 634  
    displaying 850  
    generating output for 852, 858  
    generating report files from 590, 629  
    getting parameters from 657  
    running log consolidator and 927  
    running report 66  
    selecting 849  
    submitting job requests for 687  
    third-party reports and 492  
    viewing log file entries for 933  
ExecutableFileName element 768, 778  
executableName parameter  
    execute report page 843  
    submit job page 859  
ExecutableVersionName element 768, 778  
ExecutableVersionNumber element 768, 778  
execute privilege 769  
execute report page 826, 835, 842  
EXECUTE\_REPORT\_WAIT\_TIME  
    parameter 842  
executedocument action 828, 830  
ExecuteQuery operations 633  
executereport action 828, 830  
ExecuteReport applications 591  
ExecuteReport class 591  
ExecuteReport element 509  
executeReport function 592  
ExecuteReport operations  
    assigning resource groups to 509  
    defining 590, 633  
    viewing logging information for 921  
ExecuteReportResponse class 591  
ExecuteReportResponse element 591  
ExecuteReportStatus data type 732  
ExecuteVolumeCommand operations 635

executing  
  reports 842  
executing BIRT Viewer servlet 868  
Execution element 773  
execution parameters 657, 773  
execution requests. *See* ExecuteReport  
  operations  
execution status attributes 635, 693  
ExecutionTimeout element 778  
Exists element 903  
expiration intervals 842, 858  
ExpirationAge element 703, 745  
ExpirationDate element 745  
ExpirationTime element 703  
expired jobs 680  
ExpireDependentFiles element 703  
explodePieSlice function 263  
explorerpane parameter 56  
export report dialog 353  
ExportBeforeViewing element 739  
exporting reports 337, 353, 376  
ExportParameterDefinitionsToFile  
  operations 636  
expressions  
  aggregating data values and 124  
  computing cross tab values and 124, 453,  
    455  
  creating EasyScript 124  
  filtering data and 163, 438  
extended character sets 824  
extended security credentials 880  
extensible markup language. *See* XML  
Extension property 720  
external access control lists 898, 899  
external authentication 888, 891, 902  
external data sources 796, 897, 903, 905  
external page-level security 899  
external registration application 888, 890, 896  
external security integration levels 907  
external security systems 888, 905  
external translated user names 732  
external user credentials 42, 137, 667  
external user groups 733  
external user names 796, 797, 904  
external user properties 665, 905  
external user registration  
  assigning security roles and 813  
authenticating users for 902  
configuring LDAP servers for 891, 892,  
  897  
described 888  
enabling 814  
getting user names for 906  
preparing volumes for 896  
searching roles for 906  
searching users for 681, 682  
external user roles 665, 681, 903  
external users 682, 903  
ExternalProperties element 907  
ExternalTranslatedRoleNames data type 732  
ExternalTranslatedUserGroupNames data  
  type 733  
ExternalTranslatedUserNames data type 732  
ExternalUserPropertyNames element 665  
ExternalUserPropertyNames property 664  
ExtractParameterDefinitionsFromFile  
  operations 636

## F

Facebook comments 377  
Facebook comments panel 354  
Factory process IDs 778  
Factory processes 781  
Factory service 856  
  assigning resource groups to 775  
  enabling 783  
  getting information about 641, 643  
  pinging 670  
  running jobs and 632, 767, 777  
FactoryPid element 778  
failed connections 145  
FAILED element 782  
Failed element 715, 732  
failed jobs 690, 732, 776, 848  
failed requests 143  
FailNoticeExpiration property 908  
failover errors 925  
failure notices 559  
  deleting 908  
  print operations 673  
  setting expiration time for 805, 807  
  submit job operations 688

FailureNoticeExpiration element  
  User type 805, 911  
  UserField type 807

FALSE operator 163, 438

fatal errors 924

FATAL logging level 889

Fault messages 511

feature maps 385, 487

FeatureOptions element 668

features 370, 372, 484

fetch direction (FileSearch) 317, 321

fetch handles  
  defined 568  
  file searches 318, 321, 567, 613  
  folder items 324, 325

fetch size (FileSearch) 318, 321

FetchDirection element  
  FileSearch type 738  
  GetFileACL operations 646  
  GetFileCreationACL operations 647  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
  UserGroupSearch type 809  
  UserSearch type 810

FetchHandle element  
  FileSearch type 738  
  GetFileACL operations 646  
  GetFileCreationACL operations 647  
  GetFolderItems operations 649  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
  SelectFiles operations 676  
  SelectJobs operations 679, 680  
  SelectJobSchedules operations 680  
  SelectUserGroups operations 682  
  SelectUsers operations 683  
  UserGroupSearch type 809  
  UserSearch type 810

FetchInfoObjectData operations 636

FetchSize element  
  FileSearch type 738  
  GetFileACL operations 645  
  GetFileCreationACL operations 647  
  GetParameterPicklist operations 656  
  JobNoticeSearch type 750

JobScheduleSearch type 757

JobSearch type 758

SelectRoles operations 906

SelectUsers operations 906

UserGroupSearch type 809

UserSearch type 810

Field element  
  FileCondition type 735  
  JobCondition type 743  
  JobNoticeCondition type 749  
  JobScheduleCondition type 754  
  UserCondition type 806  
  UserGroupCondition type 807

FieldControlType element 734

FieldDefinition data type 733

FieldDefinition objects 708

fields 118  
  *See also* columns

FieldValue data type 734

file access type attributes 735

file access types  
  getting 307, 316  
  job properties and 652  
  setting 310, 319

file attributes 57  
  *See also* file properties

File class 306

File data type 734, 815

file dependencies  
  expiring 703  
  getting 317  
  moving files and 761  
  removing 811  
  replacing existing files and 763, 811  
  searching for 321, 737  
  specifying 790

file descriptions 307, 310, 734, 739

file detail page 834, 838

file drop page 834, 841

File element 633, 648, 736

file events 730, 732, 736

file IDs 307, 310, 317, 320  
  *See also* FileId element

file index page 835, 846

file list page 849

file lists 313, 718, 849

file name extensions 307, 676, 739

file names  
  downloading files and 588, 620  
  getting 187, 308, 319, 344  
  moving files and duplicate 761  
  printing 581  
  saving report designs and 348  
  saving report documents and 348  
  search operations and 676  
  setting 311, 323, 734, 762  
  uploading files and 580  
  viewing reports and 350, 351

File objects 306, 708

file owner attribute 735

file owners  
  getting 308, 318  
  setting 311, 322

file paths. *See paths*

file properties  
  copying 574, 618  
  displaying 568  
  downloading files and 588, 620, 633  
  getting 566, 612, 647  
  setting 734  
  updating 788

file search objects 299, 301, 315

file size 308, 311

file size attribute 734, 737, 753

file stream objects 618

file streams 587

file system interface 297

file system repositories 41

file type attributes 738

file type events 736

file type icons 739, 792

file type objects 709

file types  
  adding 719, 738  
  deleting 727  
  duplicating 720  
  generating executable files and 590  
  getting conversion options for 641  
  getting extensions 307  
  getting parameters for 648  
  resource groups and 775  
  searching 676  
  setting 310, 508, 635, 734  
  updating 791, 792

uploading files and 882  
verifying 883  
viewing logging information for 937

FileAccess data type 735

FileCondition class 313

FileCondition data type 735

FileCondition objects 708

FileContent data type 736

FileContent objects 709

file-creation privileges 646, 802

FileCreationACL template 646

filedatasource parameter 77

FileEvent data type 736

FileEvent element 730, 732

FileField data type 736

filefoldersprivilege action 830

FileId element  
  DownloadFile operations 632  
  GetConnectionProperties operations 638  
  GetConnectionPropertyAssignees  
    operations 639  
  GetFileACL operations 645  
  GetFileDetails operations 647  
  SaveTransientReport operations 675  
  SetConnectionProperties operations 684  
  UploadFile operations 692

FileId variable 587, 620

FileName element  
  DownloadFile operations 632  
  GetConnectionProperties operations 638,  
    903  
  GetConnectionPropertyAssignees  
    operations 639  
  GetFileACL operations 645  
  GetFileDetails operations 647  
  Ping operations 670  
  SetConnectionProperties operations 683

filename parameter 855

FileName variable 587, 620

FileProperties element 630, 633

FileProperties variable 588, 620

files  
  *See also* report files  
  accessing 849  
  archiving 843, 859  
  assigning privileges to 735, 856  
  attaching to e-mail 689, 736

counting pages in 308  
creating 762  
debugging 746  
defining attributes of 734  
defining fields in 567, 613, 736  
deleting 726, 841  
displaying 850, 868  
embedding in SOAP messages 581  
filtering 849  
getting information about 838  
handling missing 789  
installing log consolidator and 928  
logging RSSE objects to 889  
monitoring 736  
naming 844, 853, 860  
naming. *See* file names  
navigating through 297  
overwriting 844, 853, 859  
reading 575  
referencing 306  
saving 844, 852  
searching 315, 735, 737, 857  
setting number of pages in 311  
setting properties for. *See* file properties  
specifying input 687  
specifying owner of 735, 737  
streaming 573, 590, 617, 784  
uploading  
    tutorial for 577  
    uploading and downloading binary 864  
    uploading third-party 573, 617  
Files and Folders page (Management Console) 586  
FileSearch class 315, 567, 613  
FileSearch data type 737  
FileSearch objects 299, 301, 315  
FileSearch type 567, 613  
FileType data type 738, 815  
FileType element  
    ArchiveRule operations 703  
    CreateFileType operations 720  
    DocumentConversionOptions type 730  
    File type 734, 736  
    GetDataExtractionFormats operations 640  
    GetDocumentConversionOptions operations 641  
    GetFileTypeParameterDefinitions operations 648  
    SOAP headers and 508, 741  
    TargetResourceGroup element and 509  
FileType objects 709  
FileType property 648  
FileType variable 567, 612  
FileTypes element  
    ResourceGroupSettings type 775  
    SelectFileTypes operations 677  
    ServerResourceGroupSetting type 781  
FileTypes property 684, 691  
Filter class 163, 438  
filter condition objects 707  
filter conditions  
    adding 163, 438, 723  
    getting operator in 441  
    getting values of 166, 442  
    matching character patterns and 163, 164, 845, 846  
    setting operator in 166, 443  
    setting values for 167, 443  
filter controls 765  
Filter element 656  
filter expressions 163, 438  
filter icons 593  
filter objects 106, 163, 438  
filter operators 163, 165, 166, 438  
filter parameter 848, 850  
filter strings 313  
filter summary view 485  
FilterCriteria data type 740  
filtering  
    data 772  
    file lists 313, 319, 322  
    jobs 848  
    report documents 849  
FilterList element 630, 631  
filters 848, 849, 850  
    adding gadgets and 250, 252  
    adding table elements and 286, 290  
    clearing 246, 286  
    creating 106, 163  
    determining if dynamic 207  
    displaying charts and 105, 235, 239  
    displaying cross tabs and 414, 422, 438  
    displaying Flash objects and 246, 248

filters (*continued*)  
enabling or disabling 377, 384  
getting column names for 165  
getting type for 440  
retrieving data and 49, 163  
setting column names for 166  
setting level names for 442, 443  
submitting requests and 170, 172  
finding data. *See* search operations  
Firefox browsers 106, 181  
firewalls 502, 874  
FirstPage element 732  
FirstTable parameter 45  
Flash charts 237, 248, 253  
Flash object elements 248  
Flash objects 246, 247, 341, 359  
FlashObject class 246  
floating-point numbers 730  
focus 349  
folder detail page 834, 838  
folder drop page 834, 841  
folder IDs 648, 836  
folder index page 835, 846  
folder labels 302  
folder list page 849  
folder lists 325, 718, 849  
folder names 298, 720, 825, 836  
folder parameter 850, 857  
folder paths. *See* paths  
FolderId element 648, 666  
FolderItems class 324  
FolderName element 648, 720  
folders  
accessing 849  
archiving contents 843  
assigning privileges to 856  
creating 720, 836  
deleting 726, 841  
displaying 302, 850  
duplicating 720  
getting access rights to 645  
getting files in 648, 675  
getting home 881  
getting names 298  
handling missing 789  
linking to 851  
moving 698, 761  
naming 300, 720  
navigating through 297, 836  
searching 566, 612, 857  
setting home 554, 805, 806  
specifying type 853  
updating 788  
viewing information about 838, 850  
folderType parameter 859  
FolderWhen property 907  
fonts 192  
forceLogin parameter 827, 875  
forceSoftRestart function 396  
forcing logins 42, 875  
foreign keys (log consolidator) 932  
format editing feature 377  
Format element 811  
format parameter 843  
formats  
*See also* output formats  
downloading reports and 337  
localizing data and 163, 492, 509  
retrieving information objects and 742  
FormName element 751  
forward definitions 828  
free disk space 855  
freeing resources 919, 920  
FrequencyInDays element 722  
FrequencyInMonths element 760  
FrequencyInWeeks element 814  
functions  
*See also* callback functions; methods  
Actuate API service 529  
Actuate JavaScript API 38, 39, 90, 135  
error logging extension 945  
Interactive Crosstabs JavaScript API 390  
usage logging extension 944

## G

g function 277  
Gadget class 250  
gadget elements 250, 360  
gadget IDs 397  
gadget names 159  
gadget objects 250  
gadget script objects 158  
gadget type change control 377

gadget types 253  
GADGET\_TYPE\_BULLET constant 253  
GADGET\_TYPE\_CYLINDER constant 253  
GADGET\_TYPE\_LINEARGAUGE  
  constant 253  
GADGET\_TYPE\_METER constant 253  
GADGET\_TYPE\_SPARK constant 253  
GADGET\_TYPE\_THERMOMETER  
  constant 253  
gadgets  
  accessing 50  
  adding to dashboards 150  
  changing 158, 377  
  displaying 254  
  filtering 250, 252  
  getting instance of 251, 343  
  getting title of 159  
  getting type 159  
  hiding 252  
  retrieving bookmarks for 251  
  setting size 253  
  specifying type 253  
GadgetScript class 158  
garbage collection 188  
generating  
  Apache Axis Javadoc 517  
  C# classes 502, 603  
  code libraries 493, 600  
  data objects 22, 28  
  debugging messages 14, 15  
  Java classes 525  
  JavaBeans 525  
  locale-specific data 492, 509  
  output 852  
  query output 213, 219, 230  
  report components 136  
  report object value files 629, 634  
  Reportlets 90  
  reports 590  
  source code 517  
Generation element 783  
generation events 922  
generation requests  
  assigning to resource groups 509  
  cancelling 715  
  getting status of 628, 635, 693  
  preserving connections for 508  
            retrying 776  
  setting status attributes for 732  
  setting wait intervals for 635  
  submitting jobs for 688  
getAccessRights function 327  
getAccessType function 307, 316  
getActiveTab function 148  
getActuateSoapPort function 529  
getActuateSoapPortAddress function 529  
getAggregationFunction function 452, 480  
GetAllCounterValues operations 637  
GetAllPaperSizes element 661  
getAttributes function 448  
getAuthenticationId method 15  
getAuthId function 537  
getAutoSuggestThreshold function 213  
getAxisType function 426, 446  
getBookmark function  
  Chart class 236  
  Crosstab class 416  
  DataItem class 242  
  FlashObject class 247  
  Gadget class 251  
  Label class 281  
  Request class 170  
  Table class 286  
  TextItem class 293  
GetBookmarks operations 637  
GetCapabilities operations 637  
GetCapabilitiesByCategory operations 637  
GetCapabilityCategories element 638  
GetCapabilityCategories operations 638  
getCascadingParentName function 213  
getCascadingParentValues function 203  
getCategoryCount function 256  
GetChannelACL operations 638  
getChart function 338  
getChartByBookmark function 358  
getChartHeight function 256  
getChartWidth function 256  
getChildData function 204  
getClientChart function 236  
getClientHeight function 339  
getClientOptions function 257  
getClientWidth function 339  
getColumn function 286, 417  
getColumnIndex function 368

getColumnMirrorStartingLevel function 461  
getColumnName function  
    Filter class 165  
    ParameterDefinition class 213  
    ParameterValue class 227  
    Sorter class 176  
getColumnNames function 174  
getColumnPageBreakInterval function 462  
getColumns function 170  
getColumnType function 213, 227  
getCondition function 316  
getConditionArray function 316  
GetConnectionProperties operations 638, 903, 908  
GetConnectionPropertyAssignees  
    operations 639  
GetContent operations 639  
getContentByBookmark function 339  
getContentByPageRange function 340  
getContentData function 594  
getContentMargin function 340  
getContentPanel function 370  
getControlType function 204, 214  
getCore function 257  
GetCounterValues operations 639  
getCountLimit function 317  
getCrosstabByBookmark function 467  
GetCubeMetaData operations 640  
getCurrentDisplayName function 214  
getCurrentPageContent function  
    Viewer class 48, 341  
    XTabAnalyzer class 397  
getCurrentPageNum function 341, 397  
getCurrentReportParameters function 159  
getCurrentValue function 204  
GetCustomFormat operations 640  
getDashboardName function 148  
getData function 243, 417  
GetDatabaseConnectionDefinition  
    operations 640  
GetDatabaseConnectionParameters  
    operations 640  
GetDatabaseConnectionTypes  
    operations 640  
GetDataExtractionFormats operations 640  
getDataItem function 341  
getDataItemByBookmark function 359  
getDataType function  
    Measure class 453  
    ParameterDefinition class 214  
    ParameterValue class 228  
getDefaultActiveTab function 156  
getDefaultIportalUrl function 138  
getDefaultRequestOptions function 138  
getDefaultValue function 204, 215  
getDefaultValueIsNull function 215  
getDependentFileId function 317  
getDependentFileName function 317  
getDescription function  
    actuate.Exception class 181  
    ReportExplorer.File class 307  
    XTabAnalyzer.Exception class 435  
getDimension function 432  
getDimensionName function 426  
getDisplayName function 215, 228  
GetDocumentConversionOptions  
    operations 641  
GetDynamicData operations 641  
getElement function 386, 436  
getEmbed function 594  
GetEmbeddedComponent operations 641  
getEmptyCellValue function 462  
getEnablePageBreak function 462  
getErrCode function 182, 436  
getErrorMessage function 387, 883, 885  
getExpression function 453  
getExtendedCredentials function 880  
GetFactoryServiceInfo element 642  
GetFactoryServiceInfo operations 641  
GetFactoryServiceJobs operations 643  
getFeatureMap function 385, 487  
getFetchDirection function 317  
getFetchHandle function 318, 324  
getFetchSize function 318  
getField function 313  
GetFileACL operations 645  
GetFileCreationACL operations 646  
getfiledetails action 830  
GetFileDetails operations 647  
getfiledetails page. *See* file detail page  
getFileType function 307  
GetFileTypeParameterDefinitions  
    operations 648  
getFilters function 170

getFilterType function 440  
getFlashObject function 341  
getFlashObjectByBookmark function 359  
getfolderitems action 831  
GetFolderItems operations 648  
getfolderitems page. *See* folder index page  
getFolderName function 298  
GetFormats operations 649  
getGadget function 343  
getGadgetByBookmark function 360  
getGadgetId function 397  
getGadgetName function 159  
getGadgetTitle function 159  
getGadgetType function 159  
getGrantedRoleId function 327  
getGrantedRoleName function 327  
getGrantedUserId function 327  
getGrantedUserName function 328  
getGroup function 215, 228  
getHeight function 343, 398  
getHelpText function 205, 216  
getHtmlDom function  
    Chart class 236  
    Crosstab class 417  
    DataItem class 243  
    FlashObject class 247  
    Gadget class 251  
    Label class 281  
    Table class 287  
        TextItem class 294  
getId function 307  
getIncludeHiddenObject function 318  
getIndex function 426, 449, 453  
GetInfoObject operations 649  
getInstanceId function  
    Chart class 237  
    DataItem class 243  
    FlashObject class 247  
    Gadget class 251  
    Label class 282  
    Table class 287  
        TextItem class 294  
getIportalUrl function 143  
getIServerUrl function 330  
getItemList function 324  
GetJavaReportEmbeddedComponent  
operations 649

GetJavaReportEmbeddedComponent  
    function 594  
GetJavaReportTOC operations 650  
getjobdetails action 828, 831  
GetJobDetails operations 651  
getjobdetails page. *See* request detail page  
getjobdetails.jsp 840  
getKey function 472  
getLabel function 282, 343  
getLabelByBookmark function 360  
getLatestVersionOnly function 298  
getLayout function 186  
getLeft function 398  
getLevelAttributeName function 440  
getLevelName function  
    Filter class 441  
    Level class 449  
    MemberValue class 458  
    Sorter class 473  
    SubTotal class 477  
getLevels function 427  
getLocale function 331  
getLocation function 477  
getMatch function 313  
getMaxRows function 170  
getMeasureDirection function 463  
getMeasureIndex function 481  
getMeasureName function 454  
getMember function 473  
getMembers function 432, 458  
getMessage function 182, 436  
getMessageContext function 590  
GetMetaData operations 653  
getMouseScrollingEnabled function 366  
getName function  
    File class 308  
    LevelAttribute class 451  
    NameValuePair class 200  
    ParameterDefinition class 216  
    ParameterValue class 228, 362, 469  
    Tab class 161  
getNameValueList function 205  
getNewAxisType function 427  
getNewIndex function 427, 454  
getnoticejobdetails action 831  
GetNoticeJobDetails operations 653  
getOperator function 165, 441

getOperatorList function 216  
getOptions function 364  
getOwner function 308, 318  
getPageContent function  
    Chart class 237  
    Crosstab class 418  
    DataItem class 243  
    FlashObject class 248  
    Gadget class 252  
    Label class 282  
    Table class 287  
    TextItem class 294  
getPageCount function 308  
GetPageCount operations 655  
GetPageNumber operations 655  
getPanInOutEnabled function 367  
getParameterGroupNames function 186  
getParameterMap function 197  
getParameterName function 205  
GetParameterPickList operations 656  
getParameterValues function 198, 398  
getParentData function 205  
getPassword function 877, 880  
getPickList function 206  
getportletfolderitems action 831  
getPosition function 216, 229, 398  
getPrivilegeFilter function 319  
getPromptParameter function 229  
getPromptText function 206  
getReportletBookmark function 344  
getReportName function 187, 344  
GetReportParameters operations 657  
getRepositoryType function 331, 880  
getrequesterjobdetails action 828, 831  
getRequestOptions function 143  
getRequiredFileId function 319  
getRequiredFileName function 319  
GetResourceGroupInfo operations 657  
GetResourceGroupList operations 658  
getResponseMessage function 590  
getResultDef function 298  
getRow function 288, 418  
getRowMirrorStartingLevel function 463  
getRowPageBreakInterval function 463  
getRunAsUser function 880  
getsavedsearch action 828, 833  
GetSavedSearch operations 659  
getSearch function 299  
getSelectedElement function 368  
getSelectNameValueList function 217  
getSelectValueList function 217  
getSerializer function 526  
getSeriesCount function 257  
GetServerResourceGroupConfiguration  
    operations 659  
getServerUrl function 880  
getServerWorkingDirectory method 16  
getShowToc function 370  
getSize function 308  
getSorters function 171  
getStartRow function 171  
GetStaticData operations 659  
GetStyleSheet operations 659  
getSuggestionList function 206  
GetSyncJobInfo operations 660  
GetSystemMDSInfo operations 660  
GetSystemPrinters operations 660  
GetSystemServerList operations 661  
GetSystemVolumeNames operations 661  
getTable function 344  
getTableByBookmark function 106, 360  
getTabName function 159  
getTabs function 156  
getTabTitle function 160  
getTabType function 161  
getTemplate function 148  
getText function 295, 344  
getTextByBookmark function 361  
getTextContent function 168  
getTimeStamp function 309  
getTitle function 162  
GetTOC operations 662  
getTop function 399  
getTotalCount function 325  
getTotalPageCount function 345, 399  
getTotals function 446, 477  
getTransientDocumentName function 187  
GetTranslatedRoleNames operations 904  
getType function  
    Chart class 237  
    Crosstab class 419  
    DataItem class 244  
    Exception class 182, 437  
    FlashObject class 248

Gadget class 252  
GrandTotal class 446  
Label class 283  
SubTotal class 478  
Table class 288  
TextItem class 295  
getUIConfig function 345  
getUIOptions function 345, 399  
getUrl function 145  
 GetUserACL operations 904  
getUserAgentString method 17  
 GetUserExtendedProperties operations 662  
 GetUserGroupExtendedProperties  
operations 662  
 GetUserGroupProductAccess operations 665  
 GetUserGroupProductAccessResponse  
element 666  
getUserHomeFolder function 881  
getUserId function 144  
 GetUserLicenseOptions operations 663  
getUserName function 877, 881  
getUserPermissions function 309  
 GetUserPreference operations 663  
 GetUserPrinterOptions operations 664  
 GetUserProperties operations 905  
getUserRoles method 13, 16, 17  
GetUsersToNotify operations 905  
getValue function  
    MemberValue class 458  
    NameValuePair class 201  
    ParameterValue class 229, 362, 470  
    ResultSet class 175  
getValueIsNull function 229, 363, 470  
getValues function 166, 441  
getVersion function 138, 309  
getVersionName function 309  
getView function  
    actuate class 106, 139  
    Viewer class 345  
    XTabAnalyzer class 400  
getViewerId function 361, 467  
getVolume function 331, 881  
GetVolumeList element 667  
getVolumeName method 12, 18  
getVolumeProfile function 331  
GetVolumeProperties operations 664  
getWidth function 347, 400  
getXAxisMax function 257  
getXAxisMin function 258  
getXTabBookmark function 400  
getXTabId function 401  
getYAxisMax function 258  
getYAxisMin function 258  
global constants 195, 304  
goto action 828  
gotoBookmark function 347  
gotoPage function 347  
grand totals 125, 445  
GrandTotal class 125, 445  
grant privilege 769  
GrantedRoleId element  
    GetFileACL operations 645  
    GetFileCreationACL operations 646  
    PrivilegeFilter type 772  
grantedRoleId value 327, 328  
GrantedRoleName element  
    GetFileACL operations 645  
    GetFileCreationACL operations 646  
    PrivilegeFilter type 772  
grantedRoleName value 327, 328  
GrantedUserGroupId element 645, 772  
GrantedUserId element  
    GetFileACL operations 645  
    GetFileCreationACL operations 646  
    PrivilegeFilter type 772  
grantedUserId value 327, 329  
GrantedUserName element  
    GetFileACL operations 645  
    GetFileCreationACL operations 646  
    PrivilegeFilter type 772  
grantedUserName value 328, 329  
GrantPermissions element 791  
GrantUserGroupCapabilities operations 665  
GrantUserGroupCapabilitiesResponse  
element 665  
GrantUserGroupProductAccess  
operations 663  
graphical user interfaces. *See* user interfaces  
graphics elements 278  
    *See also* images  
graphs. *See* charts  
GREATER\_THAN operator 163, 438  
GREATER\_THAN\_OR\_EQUAL  
operator 163, 438, 845

Group data type 740  
group editing feature 378  
Group element 764, 767  
group names 721  
groupBy function 288  
GroupCondition data type 740  
GroupField data type 740  
grouping data rows 91, 288  
Grouping data type 740  
GroupingEnabled element 652, 654  
GroupList element 796  
GroupName element 906  
GroupPromptText element 766  
groups  
*See also* notification groups; resource groups  
assigning privileges 768  
assigning to users 801  
creating user 721  
defining administrator 732  
defining attributes of 807, 808  
deleting 729  
external users and 733  
getting assigned privileges for 772  
getting available capabilities for 637, 638  
getting product family access of 665  
granting capabilities for 665  
granting product family access to 663  
removing in tables 289  
removing users 801  
revoking capabilities for 674  
revoking product access for 674  
searching 807, 808  
updating 798, 801  
validating 667  
GroupSearch data type 740  
GUI components  
*See also* user interfaces

## H

Header class 608  
Header data type 741  
header elements (SOAP messages) 496, 504  
*See also* SOAP headers  
Header type 507  
headers (HTML) 91

Headline element 687, 748, 749  
headline parameter  
  execute report page 843  
  output page 852  
  submit job page 859  
headlines 18, 748  
health monitoring errors (volumes) 925  
help 384, 486  
help text 205, 216, 222, 766  
helper classes (RSSE) 889  
HelpText element 766  
hexadecimal encoding 823  
hexadecimal values 701  
hidden files 318, 322  
hidden objects 738  
hidden parameters 733, 765  
hide function  
  Chart class 238  
  ClientSeries class 270  
  DataItem class 244  
  FlashObject class 248  
  Gadget class 252  
  Label class 283  
  Table class 289  
  TextItem class 295  
hide/show item feature 378  
hideColumn function 289  
hideDetail function 129, 289, 419  
hideNavBar function 187  
hideParameterGroup function 187  
hideParameterNames function 188  
hiding  
  chart elements 238  
  columns 48, 289, 717  
  data 129, 244, 419  
  Flash objects 248, 252  
  HTML5 charts 270  
  Interactive Crosstabs features 128  
  label elements 283  
  navigation bars 187  
  parameters 733  
  report parameters 187, 188, 217, 222  
  table of contents 371  
  tables 289  
  text items 295  
  toolbars 48  
Highcharts class 274

Highcharts documentation 276  
Highcharts drawing elements 275  
Highcharts objects 257, 274  
Highcharts point class 266  
Highcharts point configurations 272  
Highcharts renderer objects 275, 277, 280  
highlighting 378  
home folder  
    getting 881  
    setting 554, 805  
home folders 851  
home page 851  
homeFolder attribute 559  
HomeFolder element 805, 806, 911  
homeFolder parameter 847  
HomeFolder property 907  
Host directive 504  
Host property 671  
HostString property 671  
hover highlight feature 378  
HTML buttons 91  
HTML code 38, 280  
HTML elements  
    adding Flash objects to 247, 251  
    adding table elements to 287  
    adding text elements to 282, 294  
    creating cross tabs and 390, 417  
    displaying charts and 236  
    displaying data and 243, 334  
    displaying Interactive Crosstabs and 391, 394  
    displaying parameters and 184  
HTML files 508  
HTML formats 337  
HTML forms 184, 192  
HTML reports  
    displaying 593  
HTML5 charts  
    accessing 49  
    adding animation feature to 270  
    adding data series to 256, 262, 269  
    applying themes to 111  
    changing data series in 259  
    changing features 49  
    displaying 273  
    drawing functions for 275  
    getting number of run-time series in 257  
    getting options for 257  
    getting series values for 257, 258  
    getting size of 256, 257  
    getting specific instance of 236  
    hiding 270  
    instantiating 255  
    labeling axes values 264  
    pivoting axes values in 263  
    redrawing 259, 271, 272  
    removing data points in 267  
    removing duplicate values for 270  
    removing series from 259, 270, 271  
    rendering 271  
    replacing data series in 271  
    resizing drawing area for 280  
    selecting data points in 267  
    selecting series for 271  
    setting data point properties for 266  
    setting options for 262  
    setting series values for 260, 261  
    setting series visibility in 272  
    setting title of 260, 264  
    setting type 263  
    updating data points in 268  
    viewing data series in 259, 264  
HTTP connections 502, 504  
HTTP headers 503, 504  
HTTP requests 38  
HTTP sessions  
    closing 43  
    initializing 40, 41  
    providing security for 41, 42  
    running Interactive Crosstabs and 434  
    sharing information for 42  
HTTPS sessions 42  
hyperlinks  
    *See also URLs*  
    enabling or disabling 379  
    sending output files and 689  
hypertext markup language. *See* HTML code  
hypertext transfer protocol. *See* HTTP  
**I**  
icon groups 593  
icons  
    file types 739, 792

icu\_version.jar 11

**Id element**

- CopyFile operations 719
- DeleteFile operations 726
- DeleteJob operations 727, 728
- DeleteUser operations 729
- DeleteUserGroup operations 729
- File type 734
- MoveFile operations 761
- ObjectIdentifier type 763
- SelectFiles operations 676
- SelectJobs operations 679
- SelectJobSchedules operations 680
- SelectUserGroups operations 682
- SelectUsers operations 683
- UndeleteUser operations 788
- UpdateJobSchedule operations 793
- UpdateUser operations 797
- UpdateUserGroup operations 798
- User type 804
- UserGroup type 807

**ID parameter** 841

**IDAPI applications**

*See also* Information Delivery API

- accessing web services for 493, 498, 502
- binding to web services 527, 605
- calling remote services for 528
- combining multiple data sources and 506
- creating SOAP messages for 496, 502, 503
- creating WSDL schemas for 493–499
- defining operations for 623
- developing 492, 496, 530, 606
- downloading files and 588, 620
- generating code library for 600
- generating locale-specific data for 492, 509
- getting SOAP port for 529
- handling client interactions with 516, 534, 600
- integrating with external security sources 888
- managing third-party reports and 492
- mapping Java types for 526
- running reports and 591
- submitting AdminOperation requests and 555, 556, 611

**IDE (integrated development environment)** 493

**IdList element**

- CopyFile operations 718
- DeleteFile operations 726
- DeleteJob operations 727, 728
- DeleteUser operations 729
- DeleteUserGroup operations 729
- MoveFile operations 761
- SelectFiles operations 676
- SelectJobs operations 679
- SelectJobSchedules operations 680
- SelectUserGroups operations 682
- SelectUsers operations 683
- UndeleteUser operations 788
- UpdateJobSchedule operations 793
- UpdateUser operations 797
- UpdateUserGroup operations 798

**IfExists parameter** 853, 859

**IgnoreActiveJob element** 728

**IgnoreDup element**

- CreateFileType operations 720
- CreateFolder operations 720
- CreateUser operations 555, 721
- CreateUserGroup operations 721
- UpdateFileType operations 792
- UpdateUser operations 797
- UpdateUserGroup operations 798

**ignoreDup parameter** 571, 615

**IgnoreMissing element**

- DeleteFile operations 726
- DeleteFileType operations 727
- DeleteJob operations 727, 728
- DeleteUser operations 729
- DeleteUserGroup operations 730
- UndeleteUser operations 788
- UpdateFile operations 789
- UpdateFileType operations 792
- UpdateJobSchedule operations 793
- UpdateUser operations 797
- UpdateUserGroup operations 798

**iHub**

- running Information Console and 824
- sending requests over 822

**iHub API** 10, 14

**iHub API reference** 15

**iHub clusters.** *See* clusters

**iHub Integration Technology**

- Apache Axis clients and 516

logging extensions for 916, 918, 919, 943  
Microsoft .NET clients and 600  
RSSE interface and 888  
running sample applications 517  
**iHub** licensing options 663, 759, 810  
**iHub** processes 670  
**iHub** repository. *See* volumes  
**iHub** server URLs  
    getting 138, 330, 591  
    setting 332, 534  
**iHub** servers  
    assigning resource groups to 775, 781  
    authenticating users for 667  
    configuring 780  
    defining attributes of 780  
    directing SOAP messages to 504  
    getting environment information for 12  
    getting login names for 881  
    getting resource groups for 658  
    getting security credentials for 880  
    getting state of 661  
    getting URLs for 138, 330, 591  
    naming 780  
    publishing report files to 10  
    running BIRT reports and 14  
    running logging extensions for 944, 945  
    running RSSE applications on 889, 898  
    sending SOAP requests to 502, 505, 506,  
        509  
    setting as system type 784  
    setting response times for 635  
    setting state 782  
    setting status of 780, 782  
    setting URLs for 332, 534  
    setting version attributes for 780, 783  
**iHub** service type codes 936, 939  
**iHub** service type definition 783  
**iHub** services 853, 855, 856, 925  
    *See also* specific **iHub** service  
**iHub System**  
    developing administrative applications  
        for 492, 553, 610  
    getting printer information for 660  
    getting user licenses for 663, 810  
    installing log consolidator for 926, 928  
    integrating web services with 492, 493  
    integrating with client applications 502  
integrating with external security  
    sources 888  
logging error information for 919, 924, 945  
logging in to 667, 690, 881  
logging usage information for 918, 921,  
    944  
monitoring 916, 944, 945  
sending SOAP requests to 502  
setting licensing options for 759, 775  
setting security credentials for 902  
setting type attributes for 784  
testing components of 668  
updating user licensing for 802  
viewing error messages for 926  
viewing log entries for 924, 926, 940  
**iHub** volumes  
    getting names of 12, 18  
image components 649  
image elements 278  
image file names 594  
image function 278  
image IDs 594  
ImageMapURL formats 811  
images  
    getting from reports 594  
    rendering charts as 593  
    streaming 784  
immediate jobs 843, 859  
importing  
    exception class definitions 29  
IN operator 163, 439, 845  
InActive element 716  
IncludeHiddenObject element 738  
includeHiddenObject value 322  
inclusive range operators 163, 164, 845, 846  
index pages 835, 846  
index values  
    accessing result sets and 175  
    changing axis type and 125, 421  
    creating total objects and 126  
    displaying cross tabs and 417, 418  
    displaying tables and 288  
    getting column 175, 286, 368  
    getting data row 171  
    getting level 449  
    getting measure 453, 454, 481  
    incrementing 430

index values (*continued*)  
  setting data row 76, 172, 173  
  setting dimension 428, 430  
  setting level 449  
  setting measure 455, 456, 482  
  setting parameter position and 224  
indexes 427, 430  
INFO logging level 889  
InfoObject element 724  
InfoObjectData data type 741  
InfoObjectFormat data type 742  
Information Console  
  accessing functionality 822  
  connecting to 41  
  customizing pages for 38  
  displaying pages for 825  
  logging in to 822, 850, 876  
  logging out of 851  
  text string limitations for 815  
  viewing debugging messages and 14  
Information Console library 877, 883  
Information Console Security Extension 874  
  *See also* IPSE applications  
Information Console URL 143  
Information Console web application  
  copying directory structures for 61, 64, 72, 82, 119  
Information Delivery API  
  *See also* IDAPI applications  
  accessing WSDL schemas for 499  
  Apache Axis clients and 516, 524, 530  
  array definitions and 703  
  data type reference for 695  
  integrating with BIRT iHub 492  
  Microsoft .NET clients and 600, 603, 606  
  operations reference for 623  
  text string limitations for 815  
information object files 742  
information objects  
  accessing parameters in 724, 764  
  closing 629  
  external security systems and 903  
  retrieving data from 741  
  submitting job requests for 686  
Information service 926  
informational messages 924  
InheritedFrom element 703  
initialize function 40, 41, 43, 139  
input 210  
Input element 905  
input element 498  
input file IDs. *See* InputFileDialog element  
input file names. *See* InputFileName element  
input files  
  attaching to responses 634  
  executing reports and 591  
  specifying as data sources 581  
  submitting jobs and 687, 744, 752  
input message element 497  
input messages 497, 498, 527, 605  
  *See also* requests  
input parameters  
  callback functions and 38, 174  
  generic objects as 196  
  job schedules and 794  
  open security libraries and 905  
input stream reader 594  
input streams 594  
InputDetail element 652, 654  
InputDetail property 651, 653  
InputFile element 634  
InputFileDialog element  
  ExecuteReport operations 634  
  JobProperties type 752  
  JobScheduleSearch type 757  
  JobSearch type 758  
  PrintReport operations 672  
  SubmitJob operations 687  
InputFileName element  
  ExecuteReport operations 633  
  JobProperties type 752  
  JobScheduleSearch type 757  
  JobSearch type 758  
  PrintReport operations 672  
  SubmitJob operations 687  
InputFileVersionName element 744  
InputParameter element 628  
InstallApp operations 666  
installation  
  Apache Ant utility 516, 890  
  log consolidator application 926, 928  
  logging extensions 916  
  page security application 898, 899  
  RSSE applications 889, 890

Int data type 725, 742  
int element 722, 787  
int objects 709  
Integer data type 742, 764  
Integer element 725, 779  
Integer parameter 779  
integers 709, 742, 787  
integrated development environment 493  
Integration element 783  
Integration service 669, 783, 923  
Integration Technology. *See* iHub Integration Technology  
IntegrationLevel element 907  
Interactive Crosstabs  
    accessing 38, 390  
    building user interface for 128  
    changing CSS position attribute for 406  
    closing 405  
    determining status of 401, 402  
    displaying cross tabs and 119, 121, 122, 394, 467  
    displaying data cubes and 485  
    displaying specific object in 400  
    enabling driller for 127, 431  
    enabling filter summary view 485  
    getting content for 397, 467  
    getting CSS position attribute for 399  
    getting current instance 401  
    getting feature map for 487  
    getting ID for 467  
    getting margins for 398, 399  
    getting parameters for 469  
    getting size 398, 400  
    getting UI options for 399  
    handling errors for 435  
    handling events for 122, 434  
    hiding features of 129  
    initializing 391  
    instantiating 390, 394  
    integrating with Interactive Viewer 401, 405, 409  
    loading 390  
    resizing 403, 404, 409  
    restarting 396  
    restoring initial state 403  
    rolling back changes to 404  
    saving documents 486, 487  
sessions timing out and 434  
setting display options for 413, 460  
setting margins for 405, 408  
setting UI options for 408, 484  
submitting requests for 391, 410  
Interactive Crosstabs API 390  
Interactive Crosstabs API class reference 393  
Interactive Crosstabs API classes 391  
Interactive Crosstabs API error constants 435  
Interactive Crosstabs API event constants 434  
Interactive Crosstabs objects 394  
Interactive Crosstabs toolbars  
    enabling or disabling 485  
    help feature for 486  
    save design feature for 486  
    save document feature for 487  
    save feature for 486  
Interactive Crosstabs viewer 390, 394, 400  
Interactive Crosstabs viewer ID 467  
interactive features 103, 124, 338  
Interactive mode (Data Analytics) 405  
Interactive mode (Interactive Crosstabs) 402  
Interactive Viewer  
    accessing 868  
    disabling 336  
    enabling 338  
    integrating crosstab analyzer with 401, 405, 409  
    linking to 868  
    navigating through reports and 869  
Interactive Viewer servlet 869  
interactive viewing status 348  
interactivity editor 105  
interfaces  
    *See also* user interfaces  
Internet Explorer 106  
IntervalInSeconds element 773  
Invoke Script action 106  
invokeSubmit parameter 843, 859  
IOCacheDBIndexConstraint data type 742  
IOCacheDefinition data type 742  
IOCacheState data type 742  
iportal URL 40, 138, 143  
iportal web service connection 179  
iPortalID parameter 827  
iPortalLogin action 831

iPortalSecurityAdapter class 877, 878, 879  
    *See also* IPSE applications  
iPortalURL parameter 137, 139  
iportalURL variable 137, 140  
IPSE applications 5, 874, 877  
IPSE Java classes 877  
IS\_MULTISHEET constant 364  
isActive function 401  
IsAdHoc element 765  
isAdHoc function 207, 217  
isAscending function 177, 473  
IsAutoArchiveRunning element 814  
isAutoSaveEnabled function 148  
IsBundled element 634, 689, 745  
isCascadingParameter function 207  
isChartWithAxes function 259, 263  
IsColor element 751, 772  
IsCompoundDoc element 740  
isConnected function 140  
isDashboard function 402  
IsDefaultPrinter element 772  
isDynamicFilter function 207  
IsDynamicSelectionList element 766  
isEnabled function 481  
isEnterprise function 881  
iServer releases 657, 679, 680  
iServer System. *See* iHub System  
IServerContext interface 11  
iserverURL parameter 332  
iserverURL variable 137, 140  
isExceptionType function 182, 437  
IsExecutable element 739  
IsExecutable property 720  
isFileTypeAllowed function 882, 883, 885  
IsHidden element 733, 765  
isHidden function 217  
IsInherited element 703  
isInitialized function 141  
isInteractive function 348, 402  
IsLoginDisabled element 804, 806  
isMultiList function 208  
IsNative element 738  
IsNative property 720  
isnull parameter 843  
IsPassword element 765  
isPassword function 218  
IsPrintable element 739  
IsPrintable property 720  
IsProgressive element 777  
IsReportCompleted element 655  
IsRequired element 733, 739, 765  
isRequired function 208, 218  
isSavingNeeded function 149  
IsSSOEnabled operations 666  
IsSyncFactory element 778  
IsSyncJob element 777  
IsTransient element 768, 777  
isUsingPersonalDashboard function 149  
IsViewParameter element 766, 767  
isViewParameter function 218, 230  
isViewParameter value 223  
ItemList element 649, 676  
iterators 77, 590  
IUploadSecurityAdapter interface 883, 885  
iv action 828  
ivMode parameter 405  
IVServlet page 869

## J

Jakarta Commons code libraries 518  
Jakarta Struts action mapping 825, 828  
.jar files  
    archiving schemas package and 517  
    building RSSE sample application and 890  
    generating data objects and 22, 25  
    installing log consolidator and 927  
    proxy objects and 502  
JAR files  
    Java event handlers and 11  
Java Architecture for XML Binding framework 927  
Java classes  
    accessing 877, 883  
    building 525  
    event handlers and 10  
    generating data objects and 22  
    generating schemas for 516  
Java Components 41  
Java event handlers 10, 12, 13  
Java namespace 927  
Java objects 927  
Java RSSE framework 888  
    *See also* RSSE applications

Java stubs 528  
Java types 526  
JavaBeans 516, 525, 526  
JavaBeans Activation Framework  
    libraries 518  
javac compiler 517  
Javadocs  
    Apache Axis 517  
JavaMail code libraries 518  
JavaScript API  
    accessing 38  
    closing sessions for 43  
    designing cross tabs and 390, 392  
    designing reports and 90  
    developing with 38, 132  
    initializing sessions for 40, 41  
    loading resources for 42  
    providing security with 41–43  
JavaScript API class libraries 38, 132, 141  
JavaScript API class reference 132, 135, 393  
JavaScript API classes 38, 91, 132, 391  
JavaScript API function reference 135  
JavaScript API functions  
    custom web pages 38, 39  
    Interactive Crosstabs 390  
    report designs 90  
JavaScript API usage error constants 195, 304  
JavaScript API usage errors 181, 436  
JavaScript event handlers 10, 12  
JavaScript framework 38  
JAXB JAR files 927  
JDBC drivers  
    running log consolidator and 927, 928  
job attributes 700  
job condition objects 710  
job events 730, 732, 743, 754  
job IDs 751  
job names 752  
job owners 768  
job purging errors 925  
job state attributes 744, 748, 752, 756  
job status attributes 635, 660  
job type attributes 743, 752, 756, 774  
job type codes (log consolidator) 933, 937  
job type descriptions (log consolidator) 937  
JobAttributes element 652, 654  
JobAttributes property 651  
JobCondition data type 742  
JobCondition objects 710  
JobEvent data type 743  
JobEvent element 730, 732  
JobField data type 743  
JobId element  
    CancelJob operations 628  
    GetJobDetails operations 651  
    GetNoticeJobDetails operations 653  
    GetReportParameters operations 657  
    JobEvent type 743  
    JobNotice type 747  
    JobNoticeField type 749  
    JobProperties type 751  
    PrintReport operations 673  
    RunningJob type 777  
    SubmitJob operations 690  
jobID parameter  
    delete job page 837  
    delete status page 838  
    request detail page 840  
    request drop page 841  
    requests index page 848  
JobInputDetail data type 744  
JobName element  
    ExecuteReport operations 633  
    JobEvent type 743  
    JobField type 743  
    JobNotice type 747  
    JobNoticeField type 749  
    JobProperties type 752  
    JobScheduleField type 755  
    PrintReport operations 672  
    SubmitJob operations 687  
jobName parameter  
    delete job page 837  
    delete status page 838  
    execute report page 843  
    request drop page 841  
    submit job page 859  
JobNotice data type 747, 816  
JobNotice objects 710  
JobNoticeCondition data type 748  
JobNoticeCondition objects 710  
JobNoticeField data type 749  
JobNotices element 679  
JobNoticeSearch data type 750

- JobPrinterOptions data type 750, 816  
 JobProperties data type 751, 816  
 JobProperties objects 710  
 JobRetryInterval element 813  
 JobRetryInterval property 803  
 jobs 15, 18
  - assigning resource groups to 687, 752
  - cancelling 841
  - canceling 628, 715
  - deleting 727, 837, 841
  - displaying 848, 856
  - failing. *See* failed jobs
  - filtering 848
  - getting information about 643, 660, 679, 839
  - getting parameters for 652, 657
  - getting properties for 651
  - getting state of 679
  - listing pending 853, 857
  - pending. *See* pending jobs
  - preserving status information for 689
  - print operations and. *See* print jobs
  - prioritizing 687, 752
  - removing notifications for 837
  - removing resource groups and 632
  - repeating 773
  - retrying 690, 776
  - running 713, 777
  - running immediately 843, 859
  - scheduling 754, 860
  - searching 679, 742, 743, 757
  - selecting 679, 680
  - sending notifications for 848, 859
  - sending notifications for. *See* notifications
  - setting file attributes for 744
  - setting options for 700
  - setting priorities for 560, 844, 860
  - setting properties for 751
  - setting state 748, 752
  - setting status of 635, 660, 732
  - submitting 686, 847, 858
  - updating schedules for 793, 794
  - viewing logging information for 937
- Jobs element 679, 680  
 jobs pages 848  
 JobSchedule data type 754, 816  
 JobScheduleCondition data type 754  
 JobScheduleCondition objects 710  
 JobScheduleDetail data type 754  
 JobScheduleDetail objects 710  
 JobScheduleField data type 755  
 JobScheduleSearch data type 756  
 JobSearch data type 757  
 JobState element 748, 749  
 jobState parameter 837, 838, 841  
 JobStatus element 743  
 JobType element 743, 752, 756  
 jrem.jar 11  
 JSPs
  - mapping actions to 825, 829
  - naming 825
- ## K
- KeepOutputFile element 690, 747  
 KeepWorkspace element 745
- ## L
- Label class 281  
 Label element 717  
 label elements 281, 343, 360  
 Label objects 281  
 labels (report explorer) 302  
 LagTime element 731  
 language-independent applications 492, 502  
 large reports 45  
 LargeImageURL element 739  
 LatestVersionOnly element
  - CopyFile operations 719
  - DeleteFile operations 726
  - GetFolderItems operations 649
  - MoveFile operations 762
  - SelectFiles operations 675
  - UpdateFile operations 789
- launch viewer feature 379  
 layout type constants (parameters) 186, 192  
 LAYOUT\_COLLAPSIBLE event 195  
 LAYOUT\_GROUP event 195  
 LAYOUT\_NONE constant 195  
 layouts (reports) 39  
 LDAP authentication application 888, 890  
 LDAP environments 42  
 LDAP external registration application 888, 890, 896

LDAP server configurations 891, 892, 897  
LDAP servers 888  
legends (chart) 103  
LESS\_THAN operator 163, 439  
LESS\_THAN\_OR\_EQUAL operator 163, 439,  
  845  
level attribute names 451  
level attribute objects 451  
level attributes 448  
Level class 122, 448  
level index values 449  
level names  
  getting 441, 449, 458, 477  
  setting 443, 450, 459, 478  
level objects 448  
LevelAttribute class 451  
libraries  
  accessing third-party code 517  
  accessing web services and 502, 516, 600  
  determining if initialized 141  
  generating code 493, 517, 600  
  loading IPSE Java class 877  
  loading JavaScript API class 38, 132  
  migrating RSSE applications and 899  
  running logging extensions and 916  
  running Open Security applications  
    and 628, 905  
LicenseOption data type 759  
LicenseOption objects 711  
LicenseOptions element 663, 665, 911  
licensing options (iHub) 663, 759, 810  
Lightweight Directory Access Protocol. *See*  
  LDAP servers  
LIKE operator 163, 439, 845  
limit parameter 843  
limitNumber parameter 844  
line charts 263  
linear gauge gadgets 253  
Link To This Page command 868  
linking to  
  Actuate BIRT Viewers 868  
linking to folders 851  
links 851, 868  
links (Information Console)  
  *See also* hyperlinks  
Linux servers  
  *See also* UNIX systems

list boxes 220  
list controls 220, 734, 765  
list pages 835  
listening port 601  
lists 849  
  copying files or folders in 718  
  counting parameters in 656  
  displaying parameters and 734  
  filtering file 313  
  getting parameter names in 656  
  returning user preferences in 663  
  searching 313  
  selecting repository contents and 297  
  setting items for folder 325  
literal attribute 498  
literal strings 350  
load function  
  actuate class 39, 40, 141  
  Dashboard class 50  
  DataService class 76  
  Parameter class 66  
  ReportExplorer class 56  
loading  
  class libraries 38, 132  
  cross tabs 390  
  dashboards 50  
  data cubes 390  
  dialog boxes 90, 141  
  Interactive Crosstabs 390  
  JavaScript API library 132  
  parameter components 66  
  report components 136, 141  
  report content 48  
  report elements 40  
  report viewers 44, 50, 90  
  resources 42  
Locale element  
  Attachment type 714  
  SOAP headers and 507, 509, 741  
locale parameter 492, 827  
Locale property 650  
Locale variable 535, 608  
locales  
  converting parameters for 197  
  formatting data for 163, 509  
  generating data for 39, 492  
  getting current 331

locales (*continued*)  
  sending attachments and 714  
  setting 332  
  showing data for 194  
  specifying 509  
  specifying current 827  
LocalExtension element 739  
localhost parameter 499, 601  
localhost value 822  
LocalServer element 781  
Location element 769  
location settings (distribution) 794  
locks 780  
log consolidator application 926–941  
log consolidator command line utility 930  
log consolidator database 926, 928, 931–942  
log consolidator schema 932, 941  
log consolidator services 931  
log files  
  naming 919  
  setting paths for 944, 945  
  tracking error information and. *See* error log files  
  tracking usage information and. *See* usage log files  
  vacant fields in 921, 924  
log4j API 889  
log4j utility 517, 889  
LogFile element 907  
logging  
  error information 916, 926, 945  
  RSSE objects 889  
  usage information 916, 926, 944  
logging counters 637, 639  
logging extension file names 916  
logging extensions 916, 918, 919, 943  
logging in to  
  BIRT iHub System 536  
  BIRT iHub system 532, 606  
  iHub System 667, 690, 881  
  Information Console 822, 850, 876  
  log consolidator database 928  
logging in to BIRT Studio 865  
logging levels  
  RSSE applications 889  
Logging Services Project (Apache) 889  
logging tool (open source) 889  
login action 828, 831  
login applications 532, 533, 536, 606  
Login class 526  
login function 528, 536, 605  
login information 42, 866, 875  
Login operations  
  *See also* SystemLogin operations  
  authenticating users and 537  
  binding to requests 498, 527, 605  
  defining 532, 606, 667  
  developing 532, 536, 606  
  generating authentication IDs for 507, 532, 606  
  handling errors for 608  
  specifying target volumes for 509  
login page 822, 826, 835, 850  
login page (BIRT Studio) 865  
login parameters 608  
Login requests  
  authenticating users and 875  
  binding to operations 498, 527, 605  
  disabling 804, 806  
  sending 532, 606, 608  
  type definitions for 495  
Login responses 532, 537, 606, 609  
login servlet 42  
Login type 495  
loginPostback parameter 851  
logins  
  customizing 5, 874  
  forcing 42, 827, 875  
  getting user information for 880, 881  
  redirecting 851  
LoginVolume element 668  
logout action 828, 831  
logout function 43, 142  
logout page 826, 835, 851  
Long data type 759  
Long objects 710  
LongDescription element 739

## M

machine addresses 527, 605  
machine names 822  
mail. *See* e-mail  
main menu 379

Management Console  
    creating users and 564  
    enabling features for 813  
    text string limitations for 815  
    uploading report files and 586

Manufacturer element 769

margins  
    enabling or disabling 375  
    getting Interactive Crosstabs 398, 399  
    getting viewer 340  
    setting Interactive Crosstabs 405, 408  
    setting viewer 349

marker lines (charts) 103

mashup page 91

MasterPage property 593

Match element  
    FileCondition type 736  
    JobCondition type 743  
    JobNoticeCondition type 749  
    JobScheduleCondition type 754  
    UserCondition type 806  
    UserGroupCondition type 808

MATCH operator 164, 439, 845

MaxFactory element 775, 781

MaxFactory property 684, 691

MaxFactoryProcesses element 642

MaxJobPriority element 805, 806, 911

MaxJobRetryCount element 813

MaxJobRetryCount property 803

MaxNotices element 805

MaxPriority element 774

MaxPriority property 691, 908

MaxRetryCount element 745, 776

MaxSyncJobRuntime element 642

MaxVersions element  
    CopyFile operations 719  
    JobInputDetail type 745  
    MoveFile operations 762  
    NewFile type 763

MDS. *See* Message Distribution service

MDSInfo data type 759

MDSInfo objects 711

MDSInfoList element 660

MDSIPAddress element 759

MDSPortNumber element 759

MDSSSLPortNumber element 759

Measure class 122, 125, 452

measure index values  
    adding totals and 126  
    changing axis type and 125  
    getting 453, 454, 481  
    setting 455, 456, 482

measure names 124, 454, 456

measure objects 452

measureDirection parameter 413, 460

measureExpression variable 124

measureName variable 124

measures  
    adding to cross tabs 123, 413, 452  
    changing direction of 125, 414  
    changing order of 125, 421  
    deleting 420  
    editing 124, 416  
    entering expressions for 455  
    filtering data in 443  
    generating summary values for 125  
    getting aggregate functions for 452  
    getting data type of 453  
    getting direction for 463  
    getting expressions for 453  
    getting index for 453, 454, 481  
    getting level values in 459  
    getting names 454  
    naming 456  
    retrieving 124  
    setting aggregate function for 454  
    setting data type of 455  
    setting direction of 413, 465  
    setting index values for 455, 456, 482  
    sorting values 472  
    viewing data cubes and 120

media types 504

member value objects 457, 458

MemberOfGroupId element 810

MemberOfGroupName element 810

members (cross tabs)  
    defining values for 457  
    drilling through 431, 433  
    getting level names for 458  
    sorting values 473, 474

MemberValue class 457

memory 210

menus 847  
    enabling or disabling 379, 384

Message Distribution Service 854  
Message Distribution service  
enabling 783  
getting names and properties for 660  
pinging 668  
setting attributes of 759  
message element (SOAP definitions) 494, 496  
message transfer protocols 492  
MessageContext class 590  
messages 847  
adding output 560  
creating users and 560  
printing file names and 581  
messages. *See* e-mail  
meta tag 39  
metadata  
creating dashboards and 150  
mapping XML to Java types and 526  
retrieving 593, 640, 653  
metadata components 640  
metadata schemas. *See* schemas  
meter gadgets 253  
methods 11, 15  
*See also* functions  
converting to SOAP calls 528  
Microsoft Active Directory servers 888  
Microsoft C# development environments 600  
Microsoft .NET environments. *See* .NET environments  
Microsoft Windows. *See* Windows systems  
MIME attachments 573, 577, 617, 633  
MIME content IDs 574  
MIME encoding 504  
MimeType element 723, 730  
MimeType property 650  
MinFactory element 775, 781  
minOccurs attribute 496  
MinPriority element 774  
MinPriority property 691  
mirror column starting level  
getting 461  
setting 414, 460, 464  
mirror row starting level  
getting 463  
setting 414, 460, 465  
missing files or folders 789  
mode attributes (Ping) 669

Mode element 669  
mode parameter 855  
Model element 769  
MonitoredFilePath element 736  
monitoring counters 637, 639  
monitoring tools  
*See also* performance monitoring  
Monthly data type 759  
Monthly element 755  
mouse scrolling 366, 367  
move columns feature 379  
MoveFile element 702  
MoveFile operations 761  
multi-byte characters 824  
multi-clue parameters 196  
multidimensional data structures 120  
multilevel dimensions 125  
multilingual reports. *See* locales  
multi-list UI elements 208  
multipart/related media type 504  
multi-select parameters 222  
multisheet render option 364  
multi-volume environments 18  
MutexClass element 739  
My Documents folder 847

## N

Name element  
ColumnSchema type 717  
CopyFile operations 719  
DeleteFile operations 726  
DeleteFileType operations 727  
DeleteResourceGroup operations 632  
DeleteUser operations 729  
DeleteUserGroup operations 730  
FieldDefinition type 733  
File type 734, 736  
FileType type 738  
GetResourceGroupInfo operations 658  
InstallApp operations 666  
LicenseOption type 759  
MoveFile operations 761  
NameValuePair type 762  
NewFile type 762  
ObjectIdentifier type 763  
ParameterDefinition type 764

ParameterValue type 767  
Printer type 769  
PropertyValue type 772, 910  
ResourceGroup type 774  
SelectFiles operations 676  
SelectFileTypes operations 677  
SelectUserGroups operations 682  
SelectUsers operations 683  
Stream type 784  
UpdateFileType operations 792  
UpdateUser operations 797  
UpdateUserGroup operations 798  
User type 804, 911  
UserField type 806  
UserGroup type 807  
UserGroupField type 808  
Volume type 813  
name element 717  
name parameter  
  file or folder detail page 839  
  file or folder drop page 841  
  privileges page 856  
Name property 720  
NameList element  
  CopyFile operations 718  
  DeleteFile operations 726  
  DeleteFileType operations 727  
  DeleteUser operations 729  
  DeleteUserGroup operations 729  
  MoveFile operations 761  
  SelectFiles operations 676  
  SelectFileTypes operations 677  
  SelectUserGroups operations 682  
  SelectUsers operations 683  
  UpdateFileType operations 792  
  UpdateUser operations 797  
  UpdateUserGroup operations 798  
names  
  *See also* user names  
  defining file types and driver 739  
  duplicating 506, 721, 761  
  getting administrator 911  
  getting parameter 656, 766  
  getting translated role 665, 904  
  portType definitions and 497  
  servlets 864  
  setting volume 813  
text string limits for 815  
namespace 132, 495, 506, 927  
namespace attribute types 493, 603  
namespace declarations 493, 496, 505, 506  
name-value pairs 762, 772, 910  
nameValueArray variable 196  
NameValuePair class 200, 593  
NameValuePair data type 762  
NameValuePair objects 711  
naming  
  configuration templates 781  
  data types 495, 787  
  Encyclopedia volumes 813  
  files 734, 762  
  folders 300, 720  
  iHub servers 780  
  JSPs 825  
  log files 919  
  output files 844, 853, 860  
  parameters 764, 767  
  printers 771, 813  
  report designs 350, 351, 407  
  report documents 193, 350, 351, 407  
  report files 311, 323  
  reports 19  
  resource groups 774  
  user groups 807  
  users 804  
naming collisions 506  
naming restrictions 825, 858  
naming restrictions. *See* case sensitivity  
native file types 738  
NAV\_FIRST constant 195, 304  
NAV\_LAST constant 195, 304  
NAV\_NEXT constant 195, 304  
NAV\_PREV constant 195, 304  
navigate function 188  
navigation bars 187  
navigation buttons 188  
navigation constants 195, 304  
navigation feature (page) 380  
navigation links 195, 304  
navigation options 868  
.NET client application 600, 606, 608  
.NET environments 493, 496  
networks 874  
NeverExpire element 703, 745

New Java Class dialog 28  
new request index page 835, 847  
newDataMartCube function 23  
newDataMartDataCube function 24  
newDataMartDataSet function 23, 24  
newDataMartSource function 23  
NewFile data type 762  
NewFile element 675, 692  
NewFile variable 573, 617  
newUser function 554  
newUser method 559  
next function 77, 175  
NextStartTime element 753, 756  
nll element 723, 787  
node start or stop errors 925  
NodeLockViolation element 780  
NodeLockViolationExpirationDate element 780  
nodes. *See* cluster nodes  
NoEvent element 732  
NON\_DASHBOARD constant 353  
non-native reports. *See* third-party reports  
Normal mode (Ping) 669  
NOT\_BETWEEN operator 164, 439, 846  
NOT\_EQ operator 164, 439  
NOT\_IN operator 164, 439, 846  
NOT\_LIKE operator 164, 439, 846  
NOT\_MATCH operator 164, 439, 846  
NOT\_NULL operator 164, 439, 846  
notification groups  
  deleting 727  
notification headlines 748  
notification options 559  
notifications 18  
  data transfer protocols for 492  
  deleting 728, 837  
  directing to external users 907  
  expiring 814  
  getting list of users for 905  
  getting parameters for 655  
  getting properties for 653, 678  
  overriding preferences for 673  
  print operations and 673  
  searching 678, 748, 749, 750  
  sending 559, 688, 848, 859  
  sending attachments with. *See* attachments  
  setting attributes of 747  
    setting number sent 753  
    specifying user names for 844, 848  
  notificationSupported parameter 859  
NotifiedChannelId element  
  GetNoticeJobDetails operations 654  
  JobNotice type 748  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
NotifiedChannelName element  
  GetNoticeJobDetails operations 654  
  JobNotice type 748  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
NotifiedUserId element  
  JobNotice type 748  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
NotifiedUserName element  
  JobNotice type 748  
  JobNoticeSearch type 750  
  JobScheduleSearch type 757  
  JobSearch type 758  
notify parameter 860  
NotifyChannels element 652, 654  
NotifyChannels property 651, 654  
NotifyChannelsById element 673, 688  
NotifyChannelsByName element 673, 688  
NotifyCount element 744, 753  
NotifyGroups element 652, 654  
NotifyGroups property 651, 653  
NotifyGroupsById element 673, 688  
NotifyGroupsByName element 673, 688  
NotifyUsers element 652, 654  
NotifyUsers property 651, 653  
NotifyUsersById element 673, 688  
NotifyUsersByName element 673, 688  
NULL operator 164, 439, 846  
null parameter 42, 43  
null value data type 701, 787  
null value flag 766, 767  
null values 14, 843  
  assigning to parameters 221, 233, 363, 471  
  authentication and 137  
  getting 215, 229, 363

reserved parameters and 40  
subtotals and 422  
testing for 164, 439, 470, 846  
**NumberOfCopies** element 751, 770, 771  
**NumBytes** element 670  
numeric data types 724, 787  
numeric values  
    defining as double 730  
    defining as integer 742  
    defining as long 759  
    defining as short 783  
    filtering 163  
    specifying precision for 725

**O**

**Object** element  
    CubeExtraction operations 630  
    DataExtraction operations 631  
    GetBookmarks operations 637  
    GetCustomFormatData operations 640  
    GetJavaReportEmbeddedComponent operations 649  
    GetJavaReportTOC operations 650  
    GetMetaData operations 653  
    GetPageCount operations 655  
    GetPageNumber operations 655  
    GetParameterPicklist operations 656  
    SaveTransientReport operations 674  
    SelectJavaReportPage operations 677  
object IDs 763  
    *See also* **ObjectId** element  
object renderer package 889  
object types 369, 386  
**ObjectId** element  
    CancelReport operations 628  
    ExecuteReport operations 635  
    GetSyncJobInfo operations 660  
    PendingSyncJob type 768  
    RunningJob type 777  
    WaitForExecuteReport operations 693  
objectID parameter 839  
**ObjectIdentifier** data type 763  
objects  
    as input parameters 196  
    expiring 703  
    getting bookmarks for 637  
getting custom formats for 640  
getting data from 631  
getting properties for 568  
initializing actuate 40  
mapping Java 927  
searching 738  
selecting viewer content and 368, 400  
setting privileges for 908  
viewing logging information about 938  
**ODBC** databases 671  
OFF logging level 889  
**OFFLINE** element 782  
**ON\_CHANGE\_COMPLETED** constant 199  
**ON\_CHANGED** constant 199  
**ON\_CONTENT\_CHANGED** constant 357, 434  
**ON\_CONTENT\_SELECTED** constant 357, 434  
**ON\_CONTENT\_SELECTED** event 368  
**ON\_DIALOG\_OK** constant 357  
**ON\_EXCEPTION** constant  
    Dashboard class 157  
    Parameter class 199  
    ReportExplorer class 305  
    Viewer class 357  
    XTabAnalyzer class 434  
**ON\_EXCEPTION** event 386  
**ON\_SELECTION\_CHANGED** constant 199, 305  
**ON\_SESSION\_TIMEOUT** constant  
    Dashboard class 157  
    Parameter class 199  
    ReportExplorer class 305  
    Viewer class 357  
    XTabAnalyzer class 434  
**OnceADay** element 701, 722, 760, 814  
onceDate parameter 860  
onceTime parameter 860  
**OnDay** element 760  
on-demand report generation. *See*  
    synchronous jobs  
online analytical processing servers. *See*  
    OLAP servers  
online documentation ix  
**ONLINE** element 782  
online help  
    *See also* help

OnlineBackupSchedule property 664  
OnlineOnly element 660  
onlyLatest parameter 850  
onPrepare events 12, 13  
onRender events 17  
onUnload function  
    Dashboard class 149  
    Parameter class 188  
    ReportExplorer class 299  
OnWeekDay element 760  
Open Security cache 796  
Open Security library 628, 905  
Open Security web service 683, 813, 896  
open server drivers 669  
open server options 634  
open server reports 690  
open server technology 492  
open source applications 868  
open source logging tool 889  
OpenInfoObject operations 668  
opening  
    BIRT Studio 864  
    connections 38, 40, 139  
    dashboards 50  
    login page 827  
    report designs 866  
    reports 44, 59, 350, 351  
    web applications 846  
OpenSecuritySelectGroupsOfUser  
element 813  
OpenSecuritySelectUsersOfRole element 813  
OpenSecuritySelectUsersOfUserGroup  
element 813  
OpenServerOptions data type 764  
OpenServerOptions element 634, 690  
Operand1 element 723  
Operand2 element 723  
Operand3 element 723  
operating systems 502, 783  
Operation element 687, 723  
operation name element 497  
operational information (usage logs) 922  
operations  
    administration. *See* Administrate  
    operations  
    archiving files and. *See* archiving  
        operations  
binding to SOAP messages 497, 498  
defining web service 493, 498, 527, 605  
login. *See* Login operations  
omitting responses for 497  
processing 106  
running multiple 502  
running RSSE applications and 902  
searching. *See* search operations  
structuring 497  
text string limitations for 815  
updating files and. *See* update operations  
viewing logging information for 932, 938  
Operator element 733, 910  
operator lists 216  
Operator user groups 733  
OperatorList element 766  
operators (ad hoc parameters) 766  
operators (filter expressions) 163, 165, 166,  
    438  
options action 832  
Options class 460  
Options element 730  
options save action 832  
Oracle databases 927, 928, 931  
Orientation element  
    JobPrinterOptions type 751  
    Printer type 769  
    PrinterOptions type 771  
OrientationOptions element 770  
OSVersion element 783  
output 844, 852, 853, 859  
    attaching to e-mail 689  
    changing report 66  
    converting 641, 686  
    formatting. *See* output formats  
    linking to 689  
    saving 591, 634  
    viewing PDF 812  
Output element 905  
output element 498  
output file access types 652  
output file names 591, 634  
output file types 635, 693, 739  
output files  
    creating job notifications and 689  
    deleting 842, 843, 859  
    executing jobs and 687, 744, 753, 795

formatting content 634  
getting access rights to 652  
getting names 591  
limiting number of 843  
naming 844, 853, 860  
saving 690, 747, 844, 852  
setting size of 753  
output format codes 933, 938  
output format descriptions 938  
output formats  
    converting report documents and 641, 686  
    displaying reports and 677, 811  
    exporting reports and 337  
    sending e-mail attachments and 689  
    viewing logging information for 938  
output message element 497  
output messages  
    *See also* responses  
    create user transactions 572, 617  
    creating users and 560  
    defining 497, 498  
    file search operations 568, 614  
    printing file names and 581  
    specifying 527, 605  
    upload file operations 618  
output page 826, 835, 852  
output parameters 794, 905  
OutputFileType element 652, 655  
OutputFileName element 748, 749  
OutputFileSize element 744, 753  
OutputFileType element 635, 693  
OutputFileVersion element 748  
OutputFileVersionName element  
    JobInputDetail type 744  
    JobNotice type 748  
    JobProperties type 753  
outputFolderType parameter 844, 853  
OutputFormat element  
    DataExtractionFormat type 723  
    DocumentConversionOptions type 730  
    ExecuteReport operations 634  
    GetDocumentConversionOptions  
        operations 641  
    SelectJavaReportPage operations 677  
outputFormat parameter 860  
OutputMaxVersion element 744  
\_ \_outputName parameter 860  
outputName parameter  
    execute report page 844  
    output page 853  
    submit job page 860  
OutputParameter element 628  
OutputProperties element 678  
OutputType element 739  
OutputType property 720  
OverrideRecipientPref element 673, 689, 746  
overwrite parameter 844  
Owner element  
    File type 735, 737  
    FileSearch type 737  
    JobField type 743  
    JobProperties type 752  
    JobScheduleField type 755  
    JobSearch type 758  
    PendingSyncJob type 768  
    RunningJob type 778  
Owner property 648  
OwnsVolume element 780

## P

packages 888  
page break editing feature 380  
page break intervals 462, 463, 464  
page break status 462  
page breaks 462, 465  
page components  
    accessing report elements in 358  
    adding Flash objects to 248, 252  
    adding tables to 287, 291  
    adding text elements to 282, 294  
    creating cross tabs and 397, 418, 467  
    displaying charts and 237, 239, 270  
    displaying data and 244  
    getting content from 48, 341, 397  
    getting current number for 341  
    getting Flash objects in 359, 360  
    linking to 379  
    navigating to specific 195, 304, 347, 380  
page content objects 358, 467  
page counts  
    files 308, 311  
    reports 345, 399, 655  
Page element 677

page layouts 39  
page-level security 888, 899, 901, 904  
*See also* page security application  
Page Level Security feature (BIRT) 668  
Page Level Security licensing option 888  
page names 825  
page navigation constants 195, 304  
page navigation feature 380  
page not found messages 847  
page numbers 341, 655, 764  
page position (viewer) 347  
page ranges 337, 340, 751, 764  
page security application 889, 890, 898, 899  
PageContent class 358, 467  
PageCount element 655, 734, 736, 753  
PageCount property 649  
PageIdentifier data type 764  
PageNum element 764  
PageNumber element 656  
PageRange element 751  
PageRef element 678  
pages  
    counting file 308  
    counting report 345, 399, 655  
    numbering 764  
    retrieving 592, 677  
    setting range of 751, 764  
    setting size 751, 770, 771  
PageSecureViewing feature 668  
PageSize element 751, 770, 771  
PageSizeOptions element 770  
paper orientation 751, 769, 771  
paper size 661, 751  
paper trays 751, 770, 771  
PaperLength element 751  
PaperTray element 751, 770, 771  
PaperTrayOptions element 770  
PaperWidth element 751  
param1 parameter 66  
Parameter class 66, 184, 195  
parameter components 66, 141, 184  
parameter control type UI values 204  
parameter control types 214, 220  
parameter convert utility class 196  
parameter definition data type 764  
parameter definition names 216  
parameter definition objects 185, 202, 210, 711  
parameter definitions  
    creating 210, 764  
    displaying parameters and 200, 226  
    entering passwords and 218, 223  
    exporting 636  
    getting auto suggest threshold for 213  
    getting column names for 213  
    getting control type for 214  
    getting data type for 214  
    getting default values for 215  
    getting display names for 214, 215  
    getting help text for 216  
    getting name-value pair for 217  
    getting operator list for 216  
    getting required parameters for 218  
    getting values for 217  
    naming 224  
    retrieving 636  
    selecting report parameters and 217, 224  
    setting auto suggest threshold for 219  
    setting column names for 219  
    setting column type for 219  
    setting control type for 220  
    setting data type for 220  
    setting display names for 220, 221  
    setting help text for 222  
    setting multiple values for 222  
    setting name-value pairs for 224  
    setting required parameters for 223  
    specifying data type returned by 213  
    specifying default values for 221  
    storing position of 216, 224  
parameter events 189  
parameter files  
    *See also* data object values files; report object value files  
parameter global constants 195  
parameter group definitions 764  
parameter group names 186, 192  
parameter groups  
    creating 210, 221, 231  
    displaying 192  
    expanding 191  
    hiding parameters in 187  
    returning 215, 228

parameter index values 224  
parameter layout type constants 186  
parameter lists  
  changing values in 199  
  defining name-value pairs for 200, 224  
  getting auto suggest threshold for 213  
  getting name-value pair for 217  
  getting parameter names in 656  
  getting parameter position in 229  
  returning 657  
  selecting values in 734, 765  
  setting auto suggest delays for 190  
  setting auto suggest threshold for 219  
  setting column names for 230, 363  
  setting column type for 230  
  setting fetch size of 191  
  setting length of 191  
  setting parameter position in 232  
parameter maps 197  
parameter names  
  getting cascading 213  
  getting for specific value 228, 362  
  getting from cross tabs 469  
  getting from data objects 205  
  getting from lists 656  
  setting 219, 232, 470  
  specifying display names for 765, 767  
parameter objects 184, 188, 193  
parameter page components 141  
parameter pages  
  changing 188  
  displaying parameter definitions and 221  
  displaying parameters and 184, 195, 210  
  enabling 380  
  getting group names for 186  
  getting layout type for 186  
  hiding navigation bar for 187  
  loading 141  
  navigating through 67, 188, 195  
  rendering content for 184, 190  
  retrieving parameters for 67  
  setting fonts for 192  
  setting layout of 192  
parameter panels 354  
parameter types 773  
parameter value objects 226, 362, 469

parameter values files. *See* data object values files; report object value files  
ParameterData class 202  
ParameterDefinition class 210  
ParameterDefinition data type 764  
ParameterDefinition objects 185, 202, 210, 711  
ParameterFile element 630  
ParameterFileDialog element 752, 756  
ParameterFileName element 753  
ParameterList element  
  ExportParameterDefinitionsToFile operations 636  
  ExtractParameterDefinitionsFromFile operations 636  
  GetFileTypeParameterDefinitions operations 648  
  GetReportParameters operations 657  
ParameterName element 656  
ParameterPickList element 656  
parameters  
  *See also* report parameters  
  accessing result sets and 174  
  adding to HTML containers 184  
  adding to URIs 823, 824, 826  
  adding to viewer component 68, 196  
  authenticating web services and 41, 137  
  changing output and 66  
  converting values 196  
  customizing 42, 137  
  declaring input 38  
  defining a dynamic filter. *See* dynamic filter parameters  
  defining scalar 733, 778  
  determining type 207, 217, 218  
  displaying 860  
  displaying dashboards and 51  
  displaying reports and 44, 50, 56, 59  
  displaying tables and 106  
  downloading files and 589  
  enabling user interface options and 48  
  filtering data and 163  
  generating locale-specific data and 492  
  generating query output and 219, 230  
  getting cross tab 398  
  getting custom URL 332  
  getting file type 648

parameters (*continued*)  
getting job specific 652  
getting viewer 362, 363  
handling events for 189, 199  
initializing HTTP sessions and 40, 139  
linking to web services 193  
loading web pages and 822  
localizing 194  
logging error information and 945  
logging in to BIRT Studio and 866  
logging usage information and 944  
performing garbage collection for 188  
prompting for input and 210, 229, 232  
referencing report 858  
retrieving data and 76, 202  
retrieving from third-party  
  executables 492  
returning Interactive Crosstabs 469  
running BIRT Studio servlet and 864  
running jobs and 687, 795  
running reports and 187, 350, 842, 846  
search operations and 566, 568, 611  
setting Interactive Viewer servlet 869  
submitting requests for 194, 226  
unloading authentication information  
  and 43  
  uploading files and 575  
Parameters class 90  
parameters page 826, 835  
ParameterValue class 226, 362, 469  
ParameterValue data type 767  
ParameterValue objects 198, 711  
ParameterValueFileDialog element 687  
ParameterValueFileName element 687  
ParameterValueList element 630  
ParameterValues element 634, 687  
ParameterValuesFile element 630  
ParameterValuesFileDialog value 634  
ParameterValuesFileName value 634  
paramValues variable 198  
parent parameters 213  
parentname variable 213  
ParentUserId element 809  
ParentUserGroupName element 808  
parser tools 507, 518  
PartitionName element 670  
partitionName parameter 855  
partitions 635, 670, 855  
passback variable 38  
PassThrough message 628  
PassThrough operations 905  
pass-through security 903  
Password element 667, 804, 902  
password parameter 137, 827, 866  
Password property 671  
password variable 43, 533, 607  
passwords  
  adding to URIs 827  
  assigning to users 559  
  changing 804  
  connecting to Deployment Kit and 41  
  connecting to web services and 42  
  creating system 690  
  creating user 554, 667, 804  
  encrypting 223  
  external security systems and 888  
  getting 218, 880  
  logging in to BIRT iHub System and 536  
  requiring 765  
  running BIRT Studio and 866  
  sending in URLs 866  
  starting logging consolidator and 931  
  updating 876  
path commands (SVG) 279  
path function 278  
PathInformation element 812  
pathName parameter 59  
paths  
  BIRT\_HOME variable for 22  
  dashboard templates 148, 154  
  error logs 945  
  Java event handlers and 11  
  output files 591  
  Report Explorer 302  
  setting folder names in 56  
  temporary files 16  
  updating file 59  
  usage logs 944  
pattern operators 163, 164, 845, 846  
Payload element 672  
payloadSize parameter 855  
PDF documents  
  displaying 593  
  setting viewing quality of 812

PDF formats 337, 689, 811  
PdfQuality element 812  
Pending element 660, 732  
pending job attributes 643  
pending job status 732  
pending jobs 643, 767, 853, 857  
pending page 826, 835, 853  
PendingSyncJob data type 767  
PendingSyncJob objects 711  
PendingSyncJobs element 642, 645  
PendingSyncJobsResultDef element 643  
PendingTime element 768  
PercentTransientReportCacheInUse element 642  
performance 210  
performance monitoring extension 943  
period (.) character 865  
Permission data type 768, 909  
Permission objects 711  
permissions. *See* privileges  
Permissions property 574, 618, 692  
persistent connections 508, 777  
persistent reports 508  
personal dashboards 149, 155  
phased downloading and viewing 508  
pick lists. *See* selection lists  
pie chart sectors 263  
pie charts 263  
ping action 832  
ping modes 855  
Ping operations 668  
ping page 826, 835, 853  
Ping requests 668, 669  
Ping responses 672  
pivot function 125, 419  
pivotChart function 263  
pixel values (viewer) 340  
platform-independent applications 492, 502  
plug-in extensions. *See* extensions  
plug-ins  
    accessing BIRT 504  
PMD. *See* Process Management Daemon  
PmdPortNumber element 781  
PollingDuration element 731  
PollingInterval element  
    Event type 731  
JobInputDetail type 746  
port name attribute 527, 605  
port names 498  
Port Number property 897  
port numbers 671  
ports 529, 759, 822, 897  
portType definitions 497, 498, 527, 605  
portType element 494, 497  
Position element 764, 767  
POST directive 504  
postback parameter 860  
PostScript formats 337  
PowerPoint formats 337, 811  
PPT formats. *See* PowerPoint formats  
Pragma directive 505  
PrecedingParameterValues element 656  
Preference element 664, 686  
PreferenceName element 663  
PreferenceNameList element 664  
preferences 559, 827  
preferences (users). *See* user preferences  
preferences (viewer) 805, 806, 908  
prepareDownloadFileCall function 589  
previewing  
    reports 90  
primary keys (log consolidator) 932  
print dialog 354  
print jobs  
    *See also* printing  
    creating 672  
    preserving status information for 689  
    retrying 673, 690, 776  
    scheduling 672, 686  
    setting printer options for 672, 750  
    setting priority for 672, 687  
    updating 794  
print requests. *See* print jobs  
print to file property 751  
Printer data type 769, 816  
Printer objects 712  
printer option objects 712  
printer options 672, 750, 771  
printer settings. *See* printer options  
PrinterName element 661, 664, 750, 771  
PrinterOptions data type 771  
PrinterOptions element  
    GetJobDetails operations 652  
    GetNoticeJobDetails operations 654

PrinterOptions element (*continued*)  
    GetUserPrinterOptions operations 664  
    GetVolumeProperties operations 665  
    PrintReport operations 672  
    SubmitJob operations 688  
PrinterOptions objects 712  
PrinterOptions property 651, 653, 665  
printers  
    defining attributes for 769  
    getting information about 660  
    getting settings for 664  
    naming 771, 813  
    setting options for 672, 750, 771  
    specifying default 772  
    updating options for 803  
    updating user list for 802  
Printers element 661  
printing  
    reports 354, 380, 672  
    specific page ranges 751  
    to files 751  
printing events 920, 922  
printing requests. *See* print jobs  
PrintReport operations 672  
PrintToFile element 751  
printUsage variable 534, 607  
priorities (jobs) 560  
prioritizing jobs 687, 752, 844  
Priority element  
    JobField type 744  
    JobProperties type 752  
    JobScheduleField type 756  
    PrintReport operations 672  
    SubmitJob operations 687  
priority parameter  
    execute report page 844  
    submit job page 860  
priorityValue parameter  
    execute report page 844  
    submit job page 860  
private access type 652, 735  
Private element 735  
private files 307, 310, 735, 763  
privilege attributes 769, 910  
privilege filter objects 326  
privilege filters  
    creating 326  
    getting 319  
    setting 322  
    setting attributes of 772  
PrivilegeFilter class 326  
PrivilegeFilter data type 772  
PrivilegeFilter element 738  
privileges 856, 858  
    assigning to files or folders 802  
    assigning to users or groups 312, 768, 772, 909  
    changing file or folder properties and 791  
    getting user 309  
    revoking 802  
    searching for 738  
    testing 772, 898  
    updating 791, 795  
privileges page 826, 835, 856  
PrivilegeTemplate element 912  
process IDs 670  
Process Management Daemon 851, 926  
Process Management Daemon port 781  
processed action status page 841  
processError function 77  
ProcessID element 670  
processID parameter 856  
processParameters function 67  
ProductFamilies element 663, 666, 674, 721  
ProductFamily element  
     GetUserExtendedProperties  
        operations 662  
     GetUserGroupExtendedProperties  
        operations 663  
     GetUserPreference operations 664  
     SetUserExtendedProperties  
        operations 685  
     SetUserGroupExtendedProperties  
        operations 685  
        SetUserPreference operations 686  
profiles (volume) 331, 333  
programming environments 493, 502, 600  
programming languages 492  
progressive parameter 844, 860  
progressive viewing 634, 777, 844, 860  
ProgressiveViewing element 634  
project files (Microsoft .NET) 600  
prompt text 206  
PromptParameter element 767

prompts 210, 229, 232, 767  
PROP\_DOMULTIREFS parameter 576  
properties  
  channels 788  
  charting applications and 374  
  configuring RSSE SOAP Service 897  
  connections 638, 683, 903, 908  
  Encyclopedia volumes 803  
  error log files 919  
  file details operations 647  
  file selection operations 566, 612  
  file type 720  
  file update operations 788  
  file upload operations 574, 618, 692  
  folder items 648  
  jobs 651, 751  
  Message Distribution service 660  
  notifications 653, 678  
  output 686  
  report parameters 197  
  resource groups  
    getting 658  
    setting 684, 775  
    updating 684, 691  
  RSSE applications 908, 912  
  search conditions 567, 568, 613  
  search results 678  
  select page operations 593  
  usage log files 919  
  users 797, 800, 903, 905  
  volumes 664

Properties element  
  CubeExtraction operations 630  
  DataExtraction operations 631  
   GetUserExtendedProperties  
    operations 662  
   GetUserGroupExtendedProperties  
    operations 663  
   SetUserExtendedProperties  
    operations 685  
   SetUserGroupExtendedProperties  
    operations 685  
properties files 889, 927  
property name-value definitions 772, 910  
PropertyName element 662  
PropertyNameList element 662  
PropertyValue data type 772, 910

PropertyValue objects 712  
protecting corporate data 874  
Prototype JavaScript Framework 38  
proxy classes 493, 516, 532  
proxy DLLs 600  
proxy namespaces 609  
proxy objects 502  
proxy servers 528, 534, 600, 607, 875  
public directories 56  
publishing  
  reports 10  
PurgeUserInfo element 729  
purging. *See deleting*

**Q**

queries  
  building data cubes and 120  
  filtering data cubes with 631  
  running ad hoc 213  
  running dynamic filter 213  
  setting column names for 219  
  setting column types for 219, 230

Query element  
  GetJobDetails operations 652  
Query event type 921  
QueryPattern element 906  
queue 642, 767  
QueuePosition element 768  
QueueTimeout element 768

**R**

radio buttons 220, 765  
Range data type 773  
Range element 764  
RCP Report Designer package  
  *See also* BIRT RCP Report Designer  
read privilege 769  
ReadFile action 669  
read-only parameters 193  
Record data type 773  
RecordDefinition element 766  
RecordFailureStatus element 673, 689, 747  
records 738, 758, 810  
  *See also* rows  
RecordSuccessStatus element 673, 689, 747  
rect function 279

rectangles 277, 279  
recurring jobs 773  
recurringDay parameter  
  execute report page 844  
  submit job page 860  
recurringTime parameter 860  
Recursive element  
  CopyFile operations 718  
  DeleteFile operations 726  
  GetJavaReportTOC operations 650  
  MoveFile operations 761  
  SelectFiles operations 675  
  UpdateFile operations 789  
redirect parameter  
  delete job page 837  
  delete status page 838  
  file or folder drop page 841  
  request drop page 841  
  submit job page 860  
redirection 851, 859  
RedirectPath element 812  
redo feature 385  
redraw function 259, 270  
reference map files 602  
referencing  
  report files 306  
  report parameters 858  
refresh intervals (log consolidator) 929  
registerEventHandler function  
  Dashboard class 149  
  Parameter class 189  
  ReportExplorer class 299  
  XTabAnalyzer class 122, 402  
RelayState element 666  
remote databases 671  
remote exceptions 556, 577  
remote procedure calls 524, 528, 603, 605  
remote services 528  
remove function 267, 271  
RemoveArchiveRules element 791  
RemoveChannelNotificationById  
  element 795  
RemoveChannelNotificationByName  
  element 794  
RemoveChildRolesById element 800  
RemoveChildRolesByName element 799  
RemoveDependentFilesById element 790  
RemoveDependentFilesByName  
  element 790  
removeDimension function 123, 419  
removeEventHandler function  
  Dashboard class 150  
  Parameter class 189  
  ReportExplorer class 299  
  XTabAnalyzer class 122, 402  
RemoveFileCreationPermissions element 802  
RemoveFromGroupsById element 801  
RemoveFromGroupsByName element 801  
removeGroup function 289  
RemoveGroupNotificationById element 795  
RemoveGroupNotificationByName  
  element 794  
RemoveLicenseOptions element 802  
removeMeasure function 420  
RemoveOutputFilePermissions element 795  
RemoveParentRolesById element 800  
RemoveParentRolesByName element 799  
RemoveRequiredFilesById element 791  
RemoveRequiredFilesByName element 790  
removeSeries function 259  
RemoveUserNotificationById element 795  
RemoveUserNotificationByName  
  element 794  
removing. *See* deleting  
render function 271  
render options 364  
render options map 364  
renderContent function 150, 190  
Renderer class 275  
Renderer objects 275, 277, 280  
rendering  
  cross tabs 409  
  dashboard content 150  
  HTML5 charts 271  
  parameter components 184, 190  
  reports 39, 90  
  specific report pages 39  
RenderOptions class 364  
reorderDimension function 125, 420  
reorderMeasure function 125, 421  
Repeat data type 773  
Repeat element 701, 722, 760, 814  
ReplaceExisting element 719, 762  
ReplaceLatestDropDependency element 811

ReplaceLatestIfNoDependents element 811  
ReplaceLatestMigrateDependency element 811  
ReplaceLatestVersion element 744  
Reply element 672  
report classes 48, 134  
report components. *See* components  
report content objects 168  
report context class 12  
report context objects 11, 12  
report design configuration objects 22  
report design engine 23  
report design engine API package 3, 5  
report design engine classes 25  
*See also* Design Engine API  
report design files 869  
    accessing 492  
    getting parameters from 657  
    naming 350, 351, 407  
report design save feature 381  
report designs 868  
    committing changes to 396  
    creating 25, 46, 52, 54, 69, 71, 119  
    opening 866  
    running 66, 509  
    saving 381  
    saving viewer content as 348  
report document files 838, 850, 869  
    getting names 187  
    naming 193, 350, 351, 407  
report document save feature 381  
report documents  
    *See also* reports  
    downloading 587, 620, 632  
    getting embedded components in 649  
    getting parameters for 657  
    getting table of contents for 650  
    running Interactive Crosstabs and 390  
    saving 348, 381  
    viewing 868  
report element IDs 243  
report element types 369  
report elements  
    accessing 23, 48  
    adding 358  
    assigning bookmarks to 48  
    displaying charts and 234, 255, 274  
    displaying data and 244  
    displaying text and 281, 293  
    embedding code in 50  
    embedding in web pages 38  
    getting bookmarks for 341, 345  
    getting from viewer 341  
    getting type 369  
    handling events for 90  
    handling exceptions for 386, 436  
    loading 40  
    retrieving result sets and 169  
    selecting 368  
    setting bookmarks for 171  
    submitting requests for 283, 296  
Report Encyclopedia. *See* volumes  
report engine api package 3, 5  
report engines 11  
report executable files 66, 850  
report execution requests. *See* ExecuteReport operations  
report explorer. *See* ReportExplorer components  
report files  
    *See also* files; specific report file type  
    accessing 849  
    archiving 843, 859  
    assigning privileges to 856  
    bundling 634, 689, 745  
    deleting 726, 841  
    displaying 850, 868  
    downloading 587, 619, 621, 632  
    embedding in SOAP messages 581, 587  
    exporting 739  
    filtering 849  
    getting access rights to 645  
    getting information about 838  
    getting names 187, 308, 344  
    getting parameters from 636, 657  
    getting properties for 566, 612  
    getting security roles for 639  
    moving 698, 761  
    naming. *See* file names  
    overwriting 844, 853, 859  
    replacing 762, 763  
    replicating properties for 574, 618  
    returning information about 675  
    returning list of 675

report files (*continued*)  
    returning location of 648  
    saving 844, 852  
    searching 857  
    searching for 315, 735, 737  
    selecting 297  
    setting access rights for 763  
    setting privileges for 735, 791, 795  
    setting version for 735, 737, 762, 811  
    specifying type 508  
    submitting jobs for 688  
    updating 788  
    uploading 573, 617, 692  
        tutorial for 577  
    viewing logging information for 921, 924

report gadgets 159

report generation events 922

report generation requests  
    assigning to resource groups 509  
    cancelling 715  
    getting status of 628, 635, 693  
    preserving connections for 508  
    retrying 776  
    setting status attributes for 732  
    setting wait intervals for 635  
        submitting jobs for 688

report items  
    displaying 378

report model api package 3

report object value files  
    creating 629, 634  
    exporting parameter definitions in 636  
    generating reports from 687  
    retrieving parameters in 636, 657

report objects. *See* reports

report parameter files  
    *See also* data object values files; report object value files

report parameters 858, 860  
    *See also* parameters  
        assigning data types to 220, 231, 733, 764  
        binding to data sources 724  
        changing 184, 199, 210  
        creating help text for 766  
        defining 764  
        determining if required 208, 218  
        downloading 66, 67, 68, 185, 186

enabling auto collection for 739  
exporting definitions 636  
getting control type for 204  
getting data type of 214, 228  
getting definitions for 636, 657  
getting display names for 228  
getting file names for 187  
getting values for 159, 229, 657  
hiding 187, 188, 217, 222, 733  
naming 470, 764, 767  
overwriting default values 634  
prompting for 229, 767  
restricting values for. *See* cascading parameters  
retrieving 66, 67, 193  
selecting 191, 206, 217, 224  
setting control type attributes for 734, 765  
setting display names for 231  
setting properties for 197  
setting values for 226, 362, 471, 767  
specifying as read-only 193  
specifying default values for 221, 765  
specifying multiple values for 222  
specifying null values for 233, 363, 471  
specifying required 223, 765  
testing for null values in 470  
viewing 67, 184, 210  
writing to files 629, 634, 753  
writing to information objects 724, 764

Report Server Security Extension 888  
    *See also* RSSE applications; RSSE API

report servers. *See* iHub servers

report status attributes 628, 635, 732

report template files 147

report titles 345

report viewer servlet 868

ReportContent class 168

ReportContent objects 168

reportContext objects 11, 12

ReportDeletion event type 921

ReportExplorer class 56, 297

ReportExplorer components  
    getting file descriptions for 307  
    getting files for 307, 308, 309  
    getting folders for 298  
    getting latest items 298  
    getting results definitions for 298

getting timestamps for 309  
getting user information for 326  
handling events for 58, 299, 305  
instantiating 56, 297  
loading 56, 141  
navigating through 58  
retrieving data for 56, 301  
searching for items for 299, 301, 313, 315  
selecting items in 58  
setting container for 300  
setting file descriptions for 310  
setting file names for 311  
setting file types for 57, 310  
setting file version for 312  
setting folder names for 300  
setting folder paths for 302  
setting items displayed in 302  
setting labels for 302  
setting latest version flag for 300  
setting results definitions for 301  
setting target service URL for 301  
setting timestamps for 312  
submitting requests for 57, 302  
viewing report items and 56, 58, 297

ReportExplorer event constants 305  
ReportExplorer global constants 304  
ReportFileDialog element 657  
ReportFileName element 657  
ReportGeneration event type 921, 922  
ReportGeneration feature 668  
reporting system. *See* iHub System  
Reportlet format 812  
Reportlet gadgets 159  
Reportlets  
generating 90  
getting bookmarks for 344  
scaling 812  
setting bookmarks for 351  
setting document mode for 407

ReportParameters element 652, 655  
ReportParameters property 651, 654  
ReportParameterType data type 773  
ReportParameterType element 657  
ReportPrinting event type 921, 922

reports  
accessing 48  
adding data items to 242

adding Flash objects to 246  
assigning to resource groups 509  
building security applications for 888, 899  
cancelling 628  
changing 376  
converting output for 641, 686, 730  
counting pages in 345, 399, 655  
deploying 901–902  
displaying 44, 508, 868  
downloading 168, 337  
embedding code in 50  
embedding in web pages 38  
exporting 337, 353, 376  
filtering 849  
generating 590  
getting embedded components in 649  
getting name of transient 187  
getting parameters for 184, 657  
getting specific pages in 592  
getting status of 693  
getting user information for 326  
getting version of 309  
hiding data in 244  
localizing 492, 509  
managing third-party 492  
naming 19  
navigating through 347, 356, 380, 868  
opening 44, 59, 350, 351  
previewing 90  
printing 354, 380, 672  
publishing 10  
reloading 357  
rendering 39, 90  
rendering specific pages for 39  
retrieving content from bookmarks 339  
retrieving data from 76, 179  
retrieving from web services 38  
retrieving parameters from 66, 67, 187, 193  
retrieving specific pages 337, 340, 341, 347  
retrieving subsets of data in 49  
returning null values 14  
returning unexpected values 14  
running 40, 590, 633, 842  
saving 591, 634, 674, 844, 852  
selecting content in 357, 368  
sending as attachments. *See* attachments  
setting locale for 332

**reports (*continued*)**

- setting page breaks for 380
- setting page numbers for 764
- setting parameters for 350
- setting version information for 312
- setting viewing mode for 593
- submitting requests for 686, 842
- suppressing duplicate values in 382
- viewing hidden elements in 249
- viewing in web browsers 66
- viewing specific parts of 45
- viewing table of contents for 370, 371, 383

**ReportType element** 774

**ReportViewing event type** 921, 922

**repositories** 864

- See also* volumes
- accessing items in 849
- displaying content 56, 297
- getting type 880
- web services and 41

repository access rights 327, 328  
repository file paths 59, 148  
repository types 331, 333  
**REPOSITORY\_ENCYCLOPEDIA**

- constant 333

**REPOSITORY\_STANDALONE** constant 333  
**repositoryType parameter** 864  
**request attributes** 510  
**Request class** 76, 169  
**request detail page** 834, 839  
**request drop page** 834, 841  
**Request element** 783  
**request objects** 76, 169  
**request parameter** 77  
**RequestedHeadline element** 753  
**RequestedOutputFile element** 591, 634, 687  
**RequestedOutputFileName element** 753, 756, 758  
**requestercanceljob action** 828, 832  
**requesterdeletejob action** 828, 832  
**requesterdeletejobschedule action** 828, 832  
**RequestID element** 509, 741  
**RequestName element** 909  
**RequestOptions class** 42, 330  
**RequestOptions objects** 40, 50, 56, 143, 330  
**requestOptions parameter** 137, 139

**requests**

- See also* SOAP messages
- adding filters to 172
- adding Flash objects and 249, 254
- adding sorters to 172
- administering BIRT iHub and 555
- administering volumes and 570, 615
- authenticating 330
- binding to operations 498, 527, 605
- caching 505
- closing HTTP sessions and 43
- defining 496, 497, 509–510
- displaying charts and 240
- displaying cross tabs and 391
- displaying dashboards and 51, 154
- displaying reports and 44, 354
- displaying tables and 291
- downloading files and 589, 620
- downloading result sets and 338
- dropping 841
- failing 143, 511
- getting bookmarks for 170
- getting options for 138
- initiating 504
- instantiating 76
- loading web pages and 822
- locale-specific reports and 492, 509
- preserving connections for 508
- retrieving data and 38, 76, 169, 244
- retrieving parameters and 67, 68, 194, 226
- retrieving printer information and 661
- retrieving report elements and 283, 296
- routing to alternate MDS 660
- running Interactive Crosstabs and 410
- sending 822
  - sending to BIRT iHub 502, 505, 506, 509
  - sending to BIRT iHub clusters 509
  - sending to volumes 505, 509, 555
- specifying default settings for 42
- specifying file type for 508
- submitting 842, 875
- testing volumes and 855
- transaction operations and 570
- validating 507

**requests index page** 835, 847

**required parameters** 208, 218, 223, 765

**required passwords** 765

RequiredFileDialog element 737  
requiredFileDialog value 319, 322  
RequiredFileName element 737  
requiredFileName value 319, 323  
Reserved element 775  
reserved parameters 40  
Reserved property 691  
reset function 403  
resetFilter parameter 848, 850  
resetting driller objects 127  
resizeTo function 403  
resizing  
    columns 375  
    Interactive Crosstabs 403, 404, 409  
    rows 381  
    viewers 154, 339, 343, 352  
Resolution element 751, 770, 771  
ResolutionOptions element 770  
resource group objects 712  
resource groups  
    assigning to jobs 687, 752  
    assigning to reports 509  
    binding to iHub servers 775, 781  
    configuring 684  
    creating 630, 774  
    deleting 632  
    getting information about 657, 659  
    getting list of 658  
    getting properties for 658  
    naming 774  
    running jobs and 774  
    setting priority for 774  
    setting properties for 684, 775  
    specifying as reserved 691, 775  
    updating 684, 691  
    viewing logging information for 933, 939  
ResourceGroup data type 774  
ResourceGroup element  
    CreateResourceGroup operations 630  
    GetResourceGroupInfo operations 658  
    JobProperties type 752  
    RunningJob type 778  
    SubmitJob operations 687  
    UpdateResourceGroup operations 691  
ResourceGroup objects 712  
ResourceGroup property 652, 654  
ResourceGroupName element 781  
ResourceGroupSettings data type 775  
ResourceGroupSettings objects 712  
ResourceGroupSettingsList element  
    CreateResourceGroup operations 630  
    GetResourceGroupInfo operations 658  
    UpdateResourceGroup operations 691  
ResourcePath element 814  
resources  
    freeing 919, 920  
    loading JavaScript API 42  
    monitoring 916  
responses  
    *See also* SOAP messages  
    attaching files to 634  
    binding to operations 498, 527, 605  
    caching 505  
    defining 496, 497, 509–510  
    embedding files in 581, 587  
    executing reports and 591  
    omitting 497  
    preserving connections for 508  
    retrieving printer information and 661, 664  
    setting wait intervals for 635  
    specifying media types for 504  
    transmitting to client applications 502  
result set components 77  
result set names 775  
result set objects 77, 174, 338  
result set schema objects 712  
result set schemas 631, 640, 653, 775  
result sets  
    *See also* queries; search results  
    accessing data in 174  
    creating 174  
    displaying data and 77  
    downloading 76, 179, 338, 353  
    exceeding limits 568  
    getting content in 175  
    incrementing data rows for 77, 175  
    referencing 174  
    retrieving data for 169, 174, 179  
    sorting values in 172, 176  
ResultDef element  
    GetFileDetails operations 647  
    GetFolderItems operations 648  
    GetJobDetails operations 651

ResultDef element (*continued*)  
    GetNoticeJobDetails operations 653  
     GetUserProperties operations 905  
    GetVolumeProperties operations 664  
    SelectFiles operations 675  
    SelectFileTypes operations 676  
    SelectJobNotices operations 678  
    SelectJobs operations 679  
    SelectJobSchedules operations 680  
    SelectUserGroups operations 681  
        SelectUsers operations 682  
ResultDef parameter 568  
ResultDef String array 568  
ResultSet class 77, 174  
ResultSet objects 77, 174, 338  
ResultSetDisplayName element 776  
ResultSetName element 775  
ResultSetSchema data type 775  
ResultSetSchema element 631  
ResultSetSchema objects 712  
RetryInterval element 745, 776  
RetryOption element 745, 776  
RetryOptions data type 776  
RetryOptions element 673, 690  
RetryOptionType data type 776  
ReturnCode element 628, 905  
RevokePermissions element 791  
RevokeUserGroupCapabilities  
    operations 674  
RevokeUserGroupCapabilitiesResponse  
    element 674  
RevokeUserGroupProductAccess  
    operations 674  
RevokeUserGroupProductAccessResponse  
    element 674  
rich text formats 812  
Role data type 776  
role IDs 327, 328, 799  
role names 327, 328, 904  
RoleCondition data type 777  
RoleField data type 777  
RoleId element 638, 684, 769  
RoleList element 796  
RoleName element  
    DoesRoleExist operations 903  
    GetConnectionProperties operations 638  
    Permission type 768, 910  
SelectUsers operations 906, 908  
SetConnectionProperties operations 684  
RoleNames element 639  
roles 13, 16, 17  
    assigning privileges 768, 909  
    deleting 799  
    getting external user 665, 904, 906  
    getting file-specific 639  
    mapping external users to 904, 910  
    reassigning 799  
    searching 681, 906  
    setting privileges for 858  
    testing for external 903  
    translating 904  
Roles element  
    SelectRoles operations 906  
    SelectUserGroups operations 682  
    UserAndProperties type 912  
RoleSearch data type 777  
rollback function 404  
RoutedToNode element 744, 753  
.rov files. *See report object value files*  
row headings 118  
row index values  
    getting 171  
    retrieving table data and 288, 418  
    setting 76, 172, 173  
row mirror starting level  
    getting 463  
    setting 414, 460, 465  
row objects 708  
ROW\_AXIS\_TYPE constant 428  
rowMirrorStartingLevel parameter 414, 460  
rowPageBreakInterval parameter 460  
rows  
    adding to cross tabs 418, 463  
    adding to tables 288  
    getting first 171  
    getting index values for 171  
    getting last 171  
    grouping 91, 288  
    incrementing 77  
    iterating through 175  
    resizing 381  
    retrieving result sets and 77, 172, 175  
    retrieving specific sets of 49, 76, 106, 164  
    returning from information objects 742

setting attributes of 723  
setting index values for 76, 172  
setting page breaks for 465  
setting page breaks on 463, 466  
sorting 49  
viewing summary data in 125  
RPCs. *See* remote procedure calls  
rptdesign format 689  
*See also* report design files  
rptdocument format 689  
*See also* report document files  
RSSE API 888, 889  
RSSE API data types 909  
RSSE API libraries 628, 905  
RSSE API operations 902  
RSSE API reference 889  
RSSE applications 897  
*See also* Report Server Security Extension  
accessing sample 888  
building sample 890, 899  
configuring LDAP servers for 891, 892, 897  
configuring logging levels for 889  
configuring volume for 890, 896  
enabling page-level security with 899  
initializing 907  
installing 889, 890  
logging in to iHub System and 667  
migrating 899  
registering external users for. *See* external user registration  
running as web service 888, 890, 898  
setting user properties for 912  
stopping 908  
translating access control lists and 899  
RSSE interface 888  
RSSE objects 889  
rsse property 889  
RSSE SOAP service settings 897  
RSSEVersion element 908  
RTF formats 812  
rules. *See* archiving rules  
run requests. *See* report generation requests  
runAdminOperation function 555, 556, 572  
RunAsUser element 667  
RunLatestVersion element 687, 752

running  
BIRT Viewer servlet 868  
Interactive Crosstabs 38, 390  
JavaBeans 516  
JavaScript API library 132  
jobs 713, 777  
report designs 66, 509  
report executables 66  
reports 40, 590, 633, 842  
RSSE applications 889  
sample applications 517, 530  
Running element 660  
running page 826, 835, 856  
RunningJob data type 777  
RunningJob objects 713  
RunningJobs element 642, 645  
RunningJobsResultDef element 644  
RunningSyncJobs element 642  
RunningTime element 778  
RunOn element 701, 760, 814  
runReport function 40  
run-time generation requests 509  
run-time parameters 218

## S

SAMLRequest element 666  
SAMLSessionIndex element 668  
SAMLToken element 667  
sample applications  
accessing code libraries for 517  
accessing Java RSSE 888  
building Java RSSE 890, 899  
CreateUser operations and 554, 610  
creating transaction applications and 571, 615  
DownloadFile operations and 588, 620  
executing reports and 591  
installing RSSE 898, 899  
login operations and 532, 536, 606  
running Apache Axis 517, 530  
running log consolidator 926, 928  
SelectFiles operations and 566, 612  
UploadFile operations and 574, 618  
sample .NET projects 600  
sample report document 391  
save function 151

saveOutput parameter 844  
SaveOutputFile element 634  
saveReportDesign function 348  
saveReportDocument function 348  
savesearch action 833  
SaveSearch operations 674  
SaveTransientReport operations 674  
saving  
  dashboards 148, 149, 151  
  output files 690, 747, 844, 852  
  report designs 381  
  report documents 348, 381  
  reports 591, 634, 674  
  temporary files 674, 745  
  viewer content 348  
Scalable Vector Graphics 352, 408  
Scalable Vector Graphics elements 275, 276, 278  
scalar parameters 733, 778  
ScalarDataType data type 778  
Scale element 751, 770, 771  
ScaleOptions element 770  
scaling factor (printer) 751, 770  
scaling options 770  
ScalingFactor element 812  
scatter charts 263  
schedule attributes 700  
schedule type attributes 755  
scheduled job page 826, 835, 857  
scheduled jobs 632, 680  
  *See also* jobs  
ScheduleDetails element 754  
ScheduleEndDate element 755  
schedulePeriod parameter 861  
schedules  
  deleting 728  
  getting information about 679, 680  
  running print jobs and 672, 686  
  running reports and 688  
  searching 680, 754, 755, 756  
  setting archiving 804  
  setting frequency 722, 759, 814  
  updating 793, 794  
Schedules element  
  GetJobDetails operations 652  
  GetNoticeJobDetails operations 654  
  PrintReport operations 672  
  SubmitJob operations 688  
Schedules property 651, 653  
ScheduleStartDate element 755  
ScheduleType element 755  
scheduleType parameter 861  
scheduling jobs 754, 860  
schema objects 707, 712  
schemas  
  *See also* WSDL schemas  
  column values and 724  
  cube metadata and 640  
  information objects and 741  
  log consolidator and 932  
  result sets and 775  
schemas package 516, 517, 525  
schemas proxy classes 516, 532  
script editor 10  
script tag 38, 39, 132  
scriptapi.jar 11  
scripting 90  
scripts  
  BIRT designers and 90  
  encapsulating in HTML code 39  
scroll bars 45, 356  
scroll controls 366  
scroll panels 366  
scrolling 45, 366, 370  
ScrollPane class 366  
SDI (Service Definition Interface) 527  
search conditions  
  entering special characters in 736  
  entering wildcards in 566, 612  
  finding users and 806, 808, 809  
  getting 316  
  matching character patterns and 736, 743, 749, 806  
  moving files and 761  
  retrieving files and 313, 567, 737  
  retrieving job schedules and 754, 755, 756  
  setting 320, 566, 612, 735  
  specifying multiple 320, 567, 613  
  specifying parameters for 566, 568, 611  
search criteria. *See* search conditions  
Search element  
  CopyFile operations 718  
  DeleteFile operations 726  
  DeleteJob operations 727, 728

DeleteJobNotices operations 728  
DeleteUser operations 729  
DeleteUserGroup operations 729  
GetFolderItems operations 649  
MoveFile operations 761  
SelectFiles operations 675  
SelectFileTypes operations 677  
SelectJobNotices operations 679  
SelectJobs operations 679  
SelectJobSchedules operations 680  
SelectUserGroups operations 681  
SelectUsers operations 682  
UpdateJobSchedule operations 793  
UpdateUser operations 797  
UpdateUserGroup operations 798  
Search event type 921, 923  
search expressions 857  
search folders page 826, 835, 857  
search operations  
*See also* searching  
assigning FileSearch objects to 301  
comparing field values and 313  
developing 566, 611  
getting FileSearch objects for 299  
retrieving external user information and 906  
setting conditions for. *See* search conditions  
targeting items for 566, 611  
search results  
counting records for 738, 758, 810  
retrieving number of items in 676, 683  
setting CSV options for 678  
Search setting property 897  
searchByCondition function 566  
searchfiles action 832  
searchFilter parameter 857  
searching  
*See also* search operations  
files 315, 676, 735, 737  
folders 566, 612, 857  
jobs 679, 742, 743, 757  
list items 313  
notifications 678, 748, 749, 750  
schedules 680, 754, 755, 756  
security roles 681, 906  
user groups 807, 808  
users 682, 806, 809, 906  
volumes 566, 611  
SearchReport operations 675  
SearchReportByIdList data type 779  
SearchReportByNameList data type 779  
SearchReportByNameList data type 779  
 SearchResultProperty data type 779  
secure read privilege 769  
security 41–43, 874, 883  
security adapter class 875, 877, 878, 879  
security adapters 43, 875, 876–886  
security applications 888, 899, 901  
security credentials. *See* credentials  
security extension Java classes 883  
security information 866  
security integration options 813  
security manager 879  
security roles 13, 16, 17  
security roles. *See* roles  
SECURITY\_ADAPTER\_CLASS parameter 878  
SecurityIntegrationOption element 813  
select function 267, 271  
SelectChannels operations 675  
SelectedContent class 368  
SelectFiles class 566  
selectFiles function 566, 568, 613  
SelectFiles operations 566, 612, 675  
SelectFileTypes operations 676  
SelectGroups operations 677  
selection list flag 766  
selection lists 206, 656  
SelectJavaReportPage class 592  
selectJavaReportPage function 594  
SelectJavaReportPage operations 593, 677  
selectjobnotices action 832  
SelectJobNotices operations 678  
selectjobs action 832  
SelectJobs operations 679  
SelectJobSchedules operations 680  
SelectNameValueList element 734, 766  
SelectPage operations 681  
SelectRoles operations 681, 906  
SelectRolesOfUser function 906  
SelectUserGroups operations 681  
SelectUsers operations 682, 906  
SelectUsersOfRole element 908

SelectValueList element 734, 765  
SEND\_TYPE\_ATTR parameter 576  
SendEmailForFailure element  
    JobInputDetail type 746  
    PrintReport operations 673  
    SubmitJob operations 688  
    User type 805, 912  
    UserField type 807  
SendEmailForSuccess element  
    JobInputDetail type 746  
    PrintReport operations 673  
    SubmitJob operations 688  
    User type 805, 912  
    UserField type 807  
SendFailureNotice element 673, 688, 746  
sending  
    embedded files 581  
    notifications 559  
sending notifications 848, 859  
sending requests 822  
sendmail utility 492  
SendNoticeForFailure element 805, 807, 911  
SendNoticeForSuccess element 805, 807, 911  
SendSuccessNotice element 673, 688, 746  
sequence element 496  
ser package 526  
Serialization class 604  
serializing  
    C# attributes 604  
    JavaBeans 516  
    remote procedure calls 524, 603  
    SOAP messages 495, 504, 526, 576, 603  
series (charts)  
    adding 256, 262, 269  
    changing 259  
    deleting 259, 270  
    displaying values 103  
    drilling through 235, 236  
    getting number of run-time 257  
    getting values for 257, 258  
    managing 269  
    removing 271  
    replacing 271  
    setting values for 260, 261  
        setting visibility of 259, 264, 272  
series names 106  
series objects 269, 270  
server configuration error messages 926  
Server element 670  
server error constants 195, 304  
server errors 181, 435  
Server Name property 897  
server parameter 856  
server state attributes 782  
server status attributes 780, 782  
server URLs  
    getting 138, 330, 591  
    setting 332, 534  
ServerBuild element 783  
serverContext objects 11  
ServerHome element 907  
ServerInformation data type 780  
ServerInformation objects 713  
ServerIPAddress element 781  
ServerList element 661  
ServerName element  
    GetFactoryServiceInfo operations 642  
    GetServerResourceGroupConfiguration  
        operations 659  
    MDSInfo type 759  
    PendingSyncJob type 768  
    RunningJob type 778  
    ServerInformation type 780  
    SetServerResourceGroupConfiguration  
        operations 684  
ServerResourceGroupSetting data type 781  
ServerResourceGroupSetting objects 713  
ServerResourceGroupSettingList  
    element 659, 684  
servers 865  
    *See also* iHub servers  
accessing 874  
building proxy 534, 600, 607  
deploying Information Console over 874  
pinging 671  
securing web services and 41, 42  
sending SOAP messages over 503  
setting context paths for 593  
    setting up firewalls and 874  
ServerState data type 782  
ServerState element 759, 782  
ServerStateErrorCode element 783  
ServerStateErrorDescription element 783  
ServerStatusInformation data type 782

ServerStatusInformation element 780  
serverURL parameter 864  
  execute report page 844  
  submit job page 861  
  URIs 827  
serverURL variable 533, 607  
ServerVersion element 783  
ServerVersionInformation data type 783  
ServerVersionInformation element 780  
Service classes 529  
Service data type 783  
Service Definition Interface (SDI) 527  
service element 494, 498  
service enable or disable errors 925  
Service objects 713  
service type codes 936, 939  
service type descriptions 939  
ServiceList element 780  
ServiceOrPort property 671  
services 853, 855, 856  
  *See also* iHub services; web services  
serviceurl parameter 42, 43  
servlet names 864  
servlets 868  
  changing 864  
  routing messages to 504  
  running BIRT Studio and 864  
session attributes 849  
session handles  
  starting user sessions and 30  
session IDs 508  
session information 851  
session state 30  
session variables 849  
setAccessRights function 328  
setAccessType function 310, 319  
setActiveTab function 152  
setAdminOperation function 556  
setAggregationFunction function 454, 481  
SetArchiveRules element 791  
setAscending function 177, 474  
SetAttributes element  
  UpdateFileOperation operations 790  
  UpdateFileTypeOperation operations 792  
  UpdateJobScheduleOperation  
    operations 794  
UpdateUserGroupOperation  
  operations 799  
UpdateUserOperation operations 800  
UpdateVolumePropertiesOperation  
  operations 803  
setAuthId function 537  
SetAutoArchiveSchedules element 804  
setAutoSaveDelay function 152  
setAutoSuggestDelay function 190  
setAutoSuggestFetchSize function 191  
setAutoSuggestListSize function 191  
setAutoSuggestThreshold function 219  
setAxisType function 428, 447  
SetBearingUsersById element 800  
SetBearingUsersByName element 799  
setBookmark function 171  
setCascadingParentName function 219  
SetChannelNotificationById element 795  
SetChannelNotificationByName element 794  
SetChannelSubscriptionsById element 802  
SetChannelSubscriptionsByName  
  element 801  
setChartTitle function 238  
setChartType function 263  
setChildData function 208  
SetChildRolesById element 800  
SetChildRolesByName element 799  
setColumnMirrorStartingLevel function 464  
setColumnName function  
  Data.Filter class 166  
  Data.Sorter class 177  
ParameterDefinition class 219  
ParameterValue class 230, 363  
setColumnPageBreakInterval function 464  
setColumns function 171  
setColumnType function 219, 230  
setCondition function 320  
setConditionArray function 320  
SetConnectionProperties operations 683  
setContainer function 152, 300  
setContentMarg function 349  
setContentPanel function 371  
setControlType function 220  
setCountLimit function 320  
setCreateUser function 555  
setCurrentDisplayName function 220  
setCurrentValue function 208

setCustomParameter function 332  
setDashboardName function 51, 152  
setData function 271  
setDataSource function 24  
setDataType function  
    Measure class 455  
    ParameterDefinition class 220  
    ParameterValue class 230  
setDefaultValue function 221  
setDefaultValueIsNull function 221  
setDependentFileId function 320  
setDependentFileName function 321  
SetDependentFilesById element 790  
SetDependentFilesByName element 790  
setDescription function 310  
setDimension function 238, 432  
setDimensionName function 428  
setDisplayName function 221, 231  
setEmailAddress method 559  
setEmptyCellValue function 464  
setEnabled function 482  
setEnablePageBreak function 465  
setExpandedGroups function 191  
setExpression function 124, 455  
setFetchDirection function 321  
setFetchHandle function 321, 325  
setFetchSize function 321  
setField function 314  
SetFileCreationPermissions element 802  
setFileType function 310  
setFilters function  
    Chart class 239  
    Crosstab class 422  
    FlashObject class 248  
    Gadget class 252  
    Request class 171  
    Table class 290  
setFilterType function 442  
setFocus function 349  
setFolderName function 56, 300  
setFont function 192  
setGadgetId function 404  
setGadgetType function 253  
setGrantedRoleId function 328  
setGrantedRoleName function 328  
setGrantedUserId function 329  
setGrantedUserName function 329  
setGroup function 221, 231  
setGroupContainer function 192  
SetGroupMembershipsById element 801  
SetGroupMembershipsByName element 801  
SetGroupNotificationById element 795  
SetGroupNotificationByName element 794  
setHeadline method 18  
setHeight function  
    Dashboard class 153  
    Viewer class 349  
    XTabAnalyzer class 404  
setHelpText function 222  
setId function 310  
setIncludeHiddenObject function 322  
setIndex function  
    Dimension class 428  
    Level class 449  
    Measure class 455  
setIsAdHoc function 222  
setIServerUrl function 332  
setIsHidden function 222  
setIsMultiSelectControl function 222  
setIsPassword function 223  
setIsRequired function 223  
setIsViewParameter function 223, 231  
setItemList function 325  
setIVMode function 405  
setKey function 474  
SetLargeWebIcon element 792  
setLatestVersionOnly function 300  
setLayout function 192  
setLeft function 405  
setLevelAttributeName function 442  
setLevelName function  
    Filter class 443  
    Level class 450  
    MemberValue class 459  
    Sorter class 474  
    SubTotal class 478  
setLevels function 429  
SetLicenseOptions element 802  
setLocale function 332  
setLocation function 478  
setMatch function 314  
setMaxJobPriority method 560  
setMaxRows function 172  
setMeasureDirection function 465

setMeasureIndex function 482  
setMeasureName function 456  
setMember function 474  
setMembers function 433  
setMouseScrollingEnabled function 367  
setName function  
    File class 311  
    LevelAttribute class 451  
    NameValuePair class 201  
    ParameterDefinition class 224  
        ParameterValue class 232, 470  
setNewAxisType function 429  
set newIndex function 430, 456  
setOnClosed function 405  
setOperationName function 577  
setOperationStyle function 576  
setOperator function 166, 443  
setOption function 364  
SetOutput FileAccess element 795  
SetOutput FilePermissions element 795  
setOwner function 311, 322  
setPageCount function 311  
setPageNum function 406  
setPanInOutEnabled function 367  
SetParameterDefinitions element 791, 792  
SetParameters element 794  
setParameters function 349  
SetParameterValues element 795  
setParameterValues function  
    Viewer class 68, 350  
    XTabAnalyzer class 406  
setParentData function 209  
SetParentRolesById element 800  
SetParentRolesByName element 799  
SetPermissions element 791  
setPosition function  
    ParameterDefinition class 224  
    ParameterValue class 232  
    XTabAnalyzer class 406  
SetPrinterOptions element  
    UpdateJobScheduleOperation  
        operations 794  
    UpdateUserOperation operations 802  
    UpdateVolumePropertiesOperation  
        operations 803  
setPrivilegeFilter function 322  
setPromptParameter function 232  
SetQuery element 795  
setReadOnly function 193  
setReportDocument function  
    Viewer class 350  
setReportletBookmark function 45, 351  
setReportletDocumentMode function 407  
setReportName function  
    Parameter class 66, 193  
    Viewer class 44, 51, 81, 351  
    XTabAnalyzer class 390, 407  
setRepositoryType function 333  
setRequiredFileDialog function 322  
setRequiredFileName function 323  
SetRequiredFilesById element 791  
SetRequiredFilesByName element 790  
setResultDef function 301  
SetRolesById element 802  
SetRolesByName element 801  
setRowMirrorStartingLevel function 465  
setRowPageBreakInterval function 466  
SetSchedules element 794  
setSearch function 301  
setSelectNameValueList function 224  
setSelectValueList function 224  
setSendEmailForFailure method 559  
setSendEmailForSuccess method 559  
setSendNoticeForFailure method 559  
setSendNoticeForSuccess method 559  
setSeriesVisible function 259, 264  
SetServerResourceGroupConfiguration  
    operations 684  
SetServerResourceGroupConfiguration  
    Response element 684  
setService function  
    Dashboard class 153  
    Parameter class 193  
    ReportExplorer class 301  
    Viewer class 351  
    XTabAnalyzer class 407  
setShowDisplayType function 194  
setShowToc function 371  
setSize function  
    Chart class 239  
    Dashboard class 153  
    File class 311  
    Gadget class 253  
    HTML5Chart Renderer class 280

setSize function (*continued*)  
    Viewer class 352  
SetSmallWebIcon element 792  
setSorters function  
    Crosstab class 126, 422  
    Request class 172  
    Table class 290  
setStartingFolder function 302  
setStartRow function 172  
setSubType function 240  
setSupportSVG function 352, 408  
SetSystemPrinters element 803  
setTemplate function 154  
setTimeStamp function 312  
settings. *See* properties  
setTitle function 260, 264  
setTop function 408  
setTotalCount function 325  
setTotals function  
    Crosstab class 422  
    GrandTotal class 447  
    SubTotal class 479  
setUIOptions function  
    Viewer class 48, 82, 352  
    XTabAnalyzer class 408  
setUseDescriptionAsLabel function 302  
SetUserExtendedProperties operations 685  
SetUserExtendedPropertiesResponse  
    element 685  
SetUserGroupExtendedProperties  
    operations 685  
SetUserGroupExtendedPropertiesResponse  
    element 685  
SetUserGroupsByID element 802  
SetUserGroupsByName element 801  
SetUserNotificationById element 795  
SetUserNotificationByName element 794  
setUserPermissions function 312  
SetUserPreference operations 685  
SetUserPreferenceResponse element 686  
setValue function  
    NameValuePair class 201  
    ParameterValue class 232  
    viewer.ParameterValue class 363  
    XTabAnalyzer.MemberValue class 459  
    XTabAnalyzer.ParameterValue class 471  
setValueIsNull function  
    ParameterValue class 233, 471  
    viewer class 363  
setValues function  
    ClientChart class 260  
    Data.Filter class 167  
    XTabAnalyzer.Filter class 443  
setVersion function 312  
setVersionName function 312  
setVersionName method 19  
setViewingMode function 353  
setViewPreference method 559  
setVisible function 272  
setVolume function 333  
setVolumeProfile function 333  
SetWaitForEvent element 796  
setWebService function 209  
setWidth function  
    Dashboard class 154  
    Viewer class 353  
    XTabAnalyzer class 409  
SetWindowsIcon element 792  
setXAxisRange function 260  
setXAxisTitle function 264  
setXTabBookmark function 409  
setXTabId function 409  
setYAxisRange function 261  
setYAxisTitle function 264  
severe errors 924  
severity levels (error logs) 924  
shared access type 653, 735  
Shared element 735  
shared files 735, 763  
shared libraries 916  
Short data type 783  
short values 787  
ShortDescription element 739, 759  
show function  
    Chart class 240  
    ClientSeries class 273  
    DataItem class 244  
    FlashObject class 249  
    Gadget class 254  
    Label class 283  
    Table class 290  
    TextItem class 295  
showColumn function 291

showDetail function  
  Crosstab class 129, 423  
  Table class 291

showDocument parameter 850

showDownloadReportDialog function 353

showDownloadResultSetDialog function 353

showExecutables parameter 850

showFacebookCommentPanel function 354

showFolders parameter 850

showFoldersOnly function 302

showParameterPanel function 354

showPrintDialog function 354

showTabNavigation function 154

showToc flag 370, 371

showTocPanel function 354

sht element 722, 787

side menu 847

simple object access protocol. *See* SOAP

single sign-on authentication 43, 666

Size element 734, 737

Size property 649

SkipPermissionError element 798

slash (/) character 865

SLN files 600

SmallImageURL element 739

SMTP messaging protocol 492

SOAP endpoints 502  
  *See also* SOAP ports

SOAP engine error messages 926

SOAP envelopes 502, 503, 505–506

SOAP Fault messages 511

SOAP header extensions 534, 535, 608

SOAP headers  
  binding definitions and 498  
  creating 507–509  
  defining attributes of 741  
  message definitions and 496  
  namespace declarations in 496, 505, 506

SOAP message error descriptions 181, 435

SOAP messages  
  accessing multiple data sources and 506  
  accessing proxy objects for 502  
  adding HTTP headers to 502, 504  
  binding to client applications 524, 603  
  binding to web service operations 209, 497, 498  
  creating 496, 502, 503

defining locale-specific 492, 509

defining requests/responses in 496, 509–510, 527, 605

embedding files in 587

preserving connections for 508

running .NET clients and 600

running JavaBeans and 516

running multiple operations and 502

scoping to WSDL files 495

sending and receiving 502, 507, 524, 603

serializing and deserializing 495, 526, 576, 603

setting length of 505

specifying media type for 504

streaming reports with 573, 590, 617

structuring content 493, 505

SOAP messaging  
  embedding files in 581

SOAP messaging framework 492, 502

SOAP ports 498, 529

SOAP processor 516, 524, 577, 600, 603

SOAP requests. *See* requests

SOAP responses. *See* responses

SOAP VersionMismatch error 506

SOAP with Attachments API libraries 518

SOAPAction directive 505

soapAction parameter 498

SOAPAction URI parameter 576

SOAP-based RSSE API 888

SOAP-based RSSE data types 909

SOAP-based RSSE operations 902

Software Development Kit  
  *See also* SDK package

solution description files 600

sort columns 631, 708, 724

sort conditions 176, 472

sort feature 373, 382

sort fields. *See* sort columns

sort keys 472, 474

sort order  
  setting 176, 177, 474  
  testing for 177, 473

sort order attributes 724

SortColumn data type 784

SortColumnList element 631

SortDirection element 724

Sorter class 126, 176, 472

sorter object arrays 172, 422  
sorter objects 126, 172, 176, 472  
sorters  
    creating 126, 176, 472  
    getting column names for 176  
    sending requests for 171, 172  
    setting column names for 177  
    setting sort order for 177, 474  
    sorting data and 49, 126  
    specifying specific tables for 290  
    testing sort order for 177, 473  
sorting data 49, 126, 172  
sortTable function 50  
source code  
    accessing for auxiliary classes 533  
    accessing JavaScript API 38  
    adding chart interactive features and 105  
    calling remote services and 528  
    compiling 25  
    constructing requests and 76  
    creating run configuration for 31  
    displaying cross tabs and 123, 127  
    displaying dashboards and 51  
    displaying parameters and 67, 68  
    displaying reports and 45, 48  
    embedding 50  
    enabling user interface options and 48  
    generating C# 603  
    generating data objects and 22  
    hiding user interface options and 129  
    initializing HTTP sessions and 40  
    initializing HTTPS sessions and 42  
    log consolidator application and 928  
    logging extensions and 919  
    logging information and 889  
    login classes and 609  
    registering event handlers and 122  
    RSSE applications and 889, 890  
    running 91  
    SOAP messages and 502, 525  
    writing event handlers and 12, 13  
source files 38  
SourceForge.net code libraries 518  
space character 824  
space characters 865  
spark gadgets 253  
special characters 736, 823, 865  
Specified element 776  
spreadsheets. *See* Excel spreadsheets  
SQL statements. *See* queries  
SSL port number 759  
Standalone element 784  
stand-alone servers 661, 784  
standards compliance mode 39  
Start operations (RSSE) 907  
start parameter 76  
StartArchive command 635  
StartArguments element  
    ResourceGroup type 775  
    ResourceGroupSettings type 775  
    ServerResourceGroupSetting type 782  
StartExpanded element 766  
StartIndex element 656  
starting  
    archiving operations 635  
    BIRT Studio 864  
    log consolidator application 928, 930, 931  
    logging operations 944, 945  
    RSSE applications 907  
STARTING element 782  
StartPartitionPhaseOut command 635  
StartRowNumber element 631  
StartTime element  
    JobField type 744  
    JobProperties type 753  
    Repeat type 773  
    RunningJob type 778  
startup messages 847  
startUpMessage parameter 847  
StartUpParameters property 671  
State element 744, 752, 756  
status codes (events) 936  
Status element  
    CancelJob operations 628  
    CancelReport operations 628  
    ExecuteReport operations 635  
    ExecuteVolumeCommand operations 636  
    GetJobDetails operations 652  
    GetNoticeJobDetails operations 655  
    GetSyncJobInfo operations 660  
    Printer type 771  
    WaitForExecuteReport operations 693  
status information 689, 732, 940  
status messages

cancelled jobs and 628  
canceling reports and 628  
transaction operations and 572, 617  
Status property 651, 654  
Stop operations 908  
StopArchive command 635  
stopping  
  archiving operations 635  
  log consolidator application 928, 930, 931  
  logging operations 919, 920, 945, 946  
  report execution 628  
  RSSE applications 908  
STOPPING element 782  
str element 723, 787  
Stream data type 784  
stream objects 618  
streaming content 573, 590, 617, 784  
strict.dtd value 39  
String data type 725, 764, 784  
String element 725, 779  
String objects 713  
String parameter 779  
string pattern operators 163, 164, 845, 846  
strings 826, 845  
  access control lists and 900  
  creating debugging messages and 14  
  filtering files and 313  
  job schedules and 755  
  limitations for characters in 815  
  localizing 39  
  logging extensions and 944, 945  
  naming report files and 734  
  passwords and 804  
  running reports and 350  
  user names and 804  
Structure data type 765  
Structure element 725  
Struts action mapping 825, 828  
subclasses 38  
subdirectories. *See* subfolders  
subfolders 858  
  deleting 726  
  getting files in 648  
  searching 566, 612  
SubmissionTime element 753, 768, 778  
submit function  
  actions and 106  
Chart class 240  
Crosstab class 423  
Dashboard class 51, 154  
DataItem class 244  
FlashObject class 249  
Gadget class 254  
Label class 283  
Parameter class 66, 194  
ReportExplorer class 302  
Table class 291  
TextItem class 295  
Viewer class 44, 48, 81, 354  
XTabAnalyzer class 410  
submit job events 924  
submit job page 826, 835, 858  
submitCallback function 51  
submitJob action 833  
SubmitJob operations  
  defining 686  
  printing requests and 672, 687  
  viewing error log information for 924  
submitting jobs 847, 858  
subpage parameter 847, 848  
SubscribedToChannelId element 810  
SubscribedToChannelName element 810  
SubscribeToChannelsById element 802  
SubscribeToChannelsByName element 801  
SubTotal class 125, 476  
subtotals 125, 422, 476, 479  
Succeeded element 715  
success notices 559, 673, 688, 814  
SuccessNoticeExpiration element 805, 806, 911  
SuccessNoticeExpiration property 908  
summary data  
  adding 118, 120, 125  
  deleting 126  
  generating grand totals and 125, 445  
  generating subtotals and 125, 422, 476  
  getting aggregate function for 480  
  getting level names for 477  
  getting location name for 477  
  getting type names for 478  
  setting aggregation function for 481  
  setting level names for 478  
  setting location of 478  
SupportCollation element 770

SupportColorMode element 771  
SupportDuplex element 771  
SupportedQueryFeatures data type 784  
SupportedQueryFeatures element 652, 654  
SupportGetTranslatedUserNames  
    element 908  
SupportNumberOfCopies element 770  
SupportOrientation element 769  
SupportPageSize element 770  
SupportPaperTray element 770  
SupportResolution element 770  
SupportScale element 770  
SVG charts 593  
SVG elements 275, 276, 278  
SVG flag 352, 408, 593  
SVG path commands 279  
SVGFlag property 593  
swapColumns function 292  
switch view feature 382  
SyncFactoryProcesses element 642  
synchronous job status attributes 635, 660  
synchronous jobs  
    assigning to resource groups 509  
    cancelling 715  
    getting information about 643, 660  
    pending 767  
    running 774, 777  
synchronous reports 509, 590, 628, 633  
synchronous resource groups 658, 775  
SyncJobQueueSize element 642  
SyncJobQueueWait element 642  
SyncResourceGroupList element 658  
system administrators 509  
    *See also* administrators  
system component ID 934, 936  
system component log records 940  
system errors 925  
system passwords 690  
system printers 660, 803  
system resources 916  
system schemas. *See* schemas  
system-generated tokens 667  
SystemLogin operations 690  
SystemPassword element 690  
SystemPasswordEncryptLevel element 690  
SystemType data type 784  
SystemType element 782

## T

Tab class 161  
tab objects 161  
tab type constants 161  
tabbed property sheets 847  
table bookmarks 106, 286  
Table class 48, 285  
Table data type 765  
Table element 725  
table elements 285  
table headers 91  
table names 286, 287  
table objects 285  
table of contents (reports)  
    displaying 370, 371  
    enabling or disabling 383  
    getting 650  
table of contents panel 354  
tableList action 833  
tables  
    adding page breaks to 462, 465  
    changing 48  
    displaying 45, 106, 285  
    filtering data in 105, 286, 290  
    getting columns in 286  
    getting instance ID of 287  
    getting rows for 288  
    grouping rows in 91, 288  
    hiding 289  
    hiding columns in 48, 289  
    hiding data in 289  
    logging operations and 928, 932  
    removing groups from 289  
    reordering columns in 292, 379, 381  
    resizing columns in 375  
    resizing rows in 381  
    retrieving 106, 344, 360  
    showing columns in 291  
    showing groups in 291  
    sorting data in 49  
    submitting requests for 291  
    suppressing duplicate values in 382  
TableValue element 767  
tabs  
    selecting 847  
tab-separated values files. *See* TSV files

Target element 718, 761  
target service URL 301, 351  
target volumes 533, 607  
targetNamespace attribute 495  
targetPage parameter 822, 851  
TargetResourceGroup element 509, 741  
TargetServer element 509, 741  
TargetVolume element 509, 741  
TargetVolume variable 535, 608  
tasks 716  
tcpmon utility. *See* TCPMonitor  
TCPMonitor utility 531  
template files 147  
template paths 154  
TemplateName element 775, 781  
templates  
    access control lists 646  
    uploading files and 578  
temporary files 16, 855  
temporary files. *See* transient files  
temporary reports. *See* transient reports  
testing  
    applications 564  
    connections 140  
    iHub components 668  
    page-level security 901  
    report generation 655  
    reporting services 856  
    RSSE application installations 898  
    scripts 90  
    upload operations 586  
text  
    displaying 344  
    getting from downloaded content 168  
    getting from text elements 295  
    getting label 282  
    limitations for 815  
    SVG flag for 593  
text box controls 734, 765  
Text class 48  
text editing feature 383  
text elements 48, 293, 295, 361  
text files 678  
text function 280  
text graphic elements 280  
text item objects 293  
text items 293, 294, 295  
text messages 847  
text objects 344  
text strings. *See* strings  
TextItem class 293  
thermometer gadgets 253  
third-party code libraries 516, 517  
third-party files 657  
third-party reports 492, 573, 617  
this keyword 50  
threads 919, 920, 944, 945  
3D charts 238  
Time data type 725, 784  
time data types 725, 784  
time dimensions (cubes) 121  
Time element 725, 779  
time formats 784  
Time parameters 779  
time stamps 844, 855  
    file creation operations and 735, 737  
    getting 309  
    setting 312  
time values 163  
time zone indicator 784  
time zones 754, 827  
TimeInstant data type 725  
TimeStamp element 735, 737  
TimeStamp property 649  
timeToDelete parameter  
    execute report page 844  
    submit job page 861  
timezone parameter 827  
TimeZoneName element 754  
titles  
    charts 235, 236, 238, 239  
    reports 345  
TocRef element 651  
tokens 667  
Tomcat servers 42  
toolbar help feature 384  
toolbar save feature 486  
toolbars 868  
    disabling 81, 485  
    Interactive Crosstabs 485  
    viewers 48, 81, 383  
tooltips 103, 382  
top N filter feature 384  
TOP\_N operator 164, 439

TOP\_PERCENT operator 164, 439  
Total class 125, 480  
total objects 126, 478, 480  
TotalCount element  
  GetFileACL operations 646  
  GetFileCreationACL operations 647  
  GetFolderItems operations 649  
  GetParameterPicklist operations 657  
  SelectFiles operations 676  
  SelectJobs operations 679, 680  
  SelectJobSchedules operations 680  
  SelectRoles operations 906  
  SelectUserGroups operations 682  
  SelectUsers operations 683, 907  
totals  
  adding grand totals and 445, 447  
  adding to subtotals 422, 476, 479  
  enabling or disabling 481, 482  
  getting 446, 478  
  returning axis type for 446  
  returning index values for 481  
  returning type 447  
  setting axis type for 447  
  setting index values for 482  
  viewing in charts 103  
    viewing in cross tabs 125, 480  
Trace mode (Ping) 670  
transaction applications 570, 571, 614, 615  
Transaction element 570, 615, 701  
transaction logs 919, 944  
Transaction operations 571, 615, 785  
Transaction requests 570, 615  
TransactionOperation arrays 571, 615  
TransactionOperation operations 785  
TransactionOperation requests 570, 615  
transactions 785, 929  
  *See also* Transaction operations  
transfer protocols 492, 502  
transient files 16, 855  
  getting names 187  
  saving 745  
transient reports 508, 642, 674, 777  
TransientReportCacheSize element 642  
TransientReportTimeout element 642  
translated user names 908  
TranslatedRoleNames data type 910  
TranslatedRoleNames element 665, 796, 904  
TranslatedRoleNames property 665  
TranslatedUserGroupNames element 797  
TranslatedUserNames data type 911  
TranslatedUserNames element 796  
TRUE operator 164, 439  
truncated strings 826  
trusted names 822  
type definitions (WSDL) 525, 603, 695  
type descriptors (Java) 526  
Type element  
  ObjectIdentifier type 763  
  ResourceGroup type 774  
  ServerResourceGroupSetting type 781  
type element 717  
TypeName data type 787  
TypeName element 717  
typens namespace prefix 495  
types. *See* data types  
types element 494, 495–496

## U

UI configuration objects 345, 370  
UI elements 208  
  *See also* user interfaces  
UIConfig class 370  
UIOptions class  
  Viewer 47, 81, 372  
  XTabAnalyzer 128, 484  
ULocale methods 11  
unauthorized users 874  
uncategorized exceptions 181  
undefined parameters 629  
UndeleteUser element 701, 785  
UndeleteUser operations 788  
undo feature 385  
Unicode character sets 504  
Uniform Resource Locators. *See* URLs  
uniform resource names (URNs) 504  
universal hyperlinks. *See* hyperlinks  
Universal Resource Identifiers. *See* URIs  
UNIX systems  
  running log consolidator for 927, 928, 930  
  running logging extensions for 916, 918, 919  
  running sample applications for 531  
  sending notifications over 492

UnsubscribeFromChannelsById element 802  
UnsubscribeFromChannelsByName element 801  
update function 268  
update operations, saving 508  
UpdateChannel element 786  
UpdateChannel operations 788  
UpdateChannelOperation operations 788  
UpdateChannelOperationGroup operations 788  
UpdateDatabaseConnection operations 691  
UpdateFile element 702, 786  
UpdateFile operations 788, 790, 791  
UpdateFileOperation element 791, 792  
UpdateFileOperation operations 790, 791  
UpdateFileOperationGroup operations 791  
UpdateFileType element 702, 786  
UpdateFileType operations 791  
UpdateFileTypeOperation operations 792  
UpdateFileTypeOperationGroup operations 792  
UpdateGroup element 785  
UpdateGroup operations 793  
UpdateGroupOperation operations 793  
UpdateGroupOperationGroup operations 793  
UpdateJobSchedule element 702, 787  
UpdateJobSchedule operations 793, 794  
UpdateJobScheduleOperation element 796  
UpdateJobScheduleOperation operations 794, 796  
UpdateJobScheduleOperationGroup operations 796  
UpdateOpenSecurityCache element 702, 787  
UpdateOpenSecurityCache operations 796  
UpdateResourceGroup operations 691  
UpdateRole operations 797, 800  
UpdateUser element 701, 785  
UpdateUser operations 797, 800  
UpdateUserGroup element 702, 786  
UpdateUserGroup operations 798  
UpdateUserGroupOperation element 800  
UpdateUserGroupOperation operations 798  
UpdateUserGroupOperationGroup operations 800  
UpdateUserOperation element 803  
UpdateUserOperation operations 800, 803  
UpdateUserOperationGroup operations 803  
UpdateVolumeProperties element 702, 787  
UpdateVolumeProperties operations 803  
UpdateVolumePropertiesOperation element 804  
UpdateVolumePropertiesOperation operations 803, 804  
UpdateVolumePropertiesOperationGroup operations 804  
updating  
channels 788  
cross tabs 127  
data 208  
external user information 796, 797  
file types 791, 792  
folders 788  
licensing options 802  
Open Security cache 796  
passwords 876  
print jobs 794  
privileges 791, 795  
report files 788  
resource groups 684, 691  
schedules 793, 794  
user groups 798, 801  
user properties 797, 800  
WSDL files 601  
upload applications 574, 618  
upload operations  
building applications for 577  
verifying 586  
upload security adapter class 883, 884  
upload security adapter interface 883, 885  
upload security adapters 882–885  
UPLOAD\_FILE\_TYPE\_LIST parameter 882  
UPLOAD\_SECURITY\_ADAPTER parameter 884  
UploadFile application 578, 585  
UploadFile class 573, 617  
uploadFile function 574, 575  
UploadFile objects 580  
UploadFile operations  
defining 573, 617, 692  
handling errors for 577, 618  
setting parameters for 575  
UploadFile requests 618  
UploadFile responses 576

uploadimage action 833  
uploading  
  report files 573, 577, 617, 692  
  third-party reports 573, 617  
uploading binary files 864, 868  
uploadlicense action 833  
URI parameters 576, 864, 869  
URIs 868  
  adding parameters to 823, 824, 826  
  creating 822, 823  
  encoding characters and 823, 824  
  obtaining list values and 849  
  overview 623, 695, 822, 825  
  Process Management Daemon and 851  
  processing SOAP messages and 504, 505  
  redirecting logins and 851  
  returning diagnostic information and 854  
  running reports and 842, 846  
  submitting requests and 822  
  uploading files and 576  
URIs reference 834  
URL parameters  
  authentication requests and 137, 193  
  HTTP sessions and 42  
  returning custom parameters in 332  
URL property 601  
URLs 868  
  accessing JavaScript library and 132  
  accessing web services and 43, 407, 498  
  accessing WSDL documents and 499, 517  
  activating security manager and 876  
  BIRT Studio 864, 865  
  BIRT Studio design environment 866  
  BIRT Studio login page 865  
  BIRT Studio servlet 864  
  character codes in 865  
  configuring log consolidator 928  
  connecting to multiple web services 42  
  displaying reports and 812  
  getting 138, 145, 330, 591  
  iHub 864  
  initializing HTTP sessions and 40  
  logging in to volumes and 534  
  retrieving parameters and 193  
  retrieving SOAP port 529  
  returning default web service 138  
  returning failed requests for 143  
  returning from exception objects 143  
  security information in 866  
  setting server 332, 534  
  setting target service 301, 351  
  specifying media types and 504  
  testing connections for 140  
  text string limits for 815  
  unloading authentication information and 43  
URNs (uniform resource names) 504  
usage error constants 195, 304  
usage errors 181, 436  
usage event IDs 932  
Usage function 607  
usage log consolidator 926  
usage log file names 916, 919  
usage log file types 916  
usage log files 920, 921, 940, 944  
usage log settings 929  
usage logging extension 916, 918, 944  
usage statements 534, 607, 608  
USAGELOG\_FILE\_EXT property 919  
USAGELOG\_FILE\_NAME property 919  
usePersonalDashboard function 155  
UseQuoteDelimiter property 678  
user activity errors 925  
user attributes 668, 804, 806, 911  
user credentials 42, 137, 667  
User data type 804, 817, 911  
User element  
  Authenticate operations 902  
  CreateUser operations 721  
   GetUserProperties operations 905  
  Login requests 667  
  Login responses 668  
  UserAndProperties type 912  
user error messages 926  
user group names 638  
user groups. *See* groups  
user IDs 827  
  *See also* UserId element  
  authentication and 42, 137, 144  
  privilege filters and 327, 329  
  removing users and 729  
user information 326  
user interface configuration objects 345, 370  
user interface elements 208

user interface options  
  changing 48  
  enabling 48, 372  
  getting 345, 399  
  hiding 129  
  setting Interactive Crosstabs 408, 484  
  setting viewer 352

user interfaces  
  browsing repository contents and 297  
  controlling viewer features and 47  
  disabling features in web pages 81  
  enabling or disabling Interactive Crosstabs  
    viewer 128

user logins, forcing 42

user names 866  
  character limitations for 815  
  connecting to Deployment Kit and 41  
  duplicating 721  
  filtering privileges and 328, 329  
  getting translated 908  
  logging in to BIRT iHub System and 536  
  logging in to iHub and 881  
  mapping external 904  
  setting 554, 804

User objects 714

user parameter 851

user preferences  
  adding users and 554, 806  
  getting list of 663  
  setting 685  
  viewing reports and 610

UserACLExternal element 908

User-Agent directive 504

UserAgent element 812

UserAndProperties data type 912

UserAndProperties element 903

UserCondition data type 806

UserCondition objects 714

UserField data type 806

UserGroup data type 807, 817

UserGroup element 721

UserGroup objects 714

UserGroupCondition data type 807

UserGroupCondition objects 714

UserGroupField data type 808

UserGroupId element  
  GetCapabilities operations 637

GetConnectionProperties operations 639

GetUserGroupExtendedProperties  
  operations 662

GetUserGroupProductAccess  
  operations 666

GrantUserGroupCapabilities  
  operations 665

GrantUserGroupProductAccess  
  operations 663

Permission type 769

RevokeUserGroupCapabilities  
  operations 674

RevokeUserGroupProductAccess  
  operations 674

SetConnectionProperties operations 684

SetUserGroupExtendedProperties  
  operations 685

UserGroupList element 796

UserGroupName element  
  GetCapabilities operations 637  
   GetUserGroupExtendedProperties  
  operations 662

GetUserGroupProductAccess  
  operations 666

GrantUserGroupCapabilities  
  operations 665

GrantUserGroupProductAccess  
  operations 663

Permission type 768

RevokeUserGroupCapabilities  
  operations 674

RevokeUserGroupProductAccess  
  operations 674

SetConnectionProperties operations 684

SetUserGroupExtendedProperties  
  operations 685

UserGroupNames element 638, 639

UserGroupSearch data type 808

UserId element  
  GetCapabilities operations 637  
  GetConnectionProperties operations 638  
   GetUserExtendedProperties  
  operations 662

   GetUserLicenseOptions operations 663

   GetUserPreference operations 663

   GetUserPrinterOptions operations 664

  Permission type 769

**UserId** element (*continued*)  
    SetConnectionProperties operations 684  
    SetUserExtendedProperties  
        operations 685  
    SetUserPreference operations 685  
**userID** parameter 42, 137, 827  
**userid** parameter 866  
**userid** variable 137, 140  
**userName** attribute 559  
**UserName** element  
    DoesUserExist operations 903  
    GetCapabilities operations 637  
    GetConnectionProperties operations 903  
     GetUserACL operations 904  
     GetUserExtendedProperties  
        operations 662  
     GetUserLicenseOptions operations 663  
     GetUserPreference operations 663  
     GetUserPrinterOptions operations 664  
    Permission type 768, 910  
    SelectRoles operations 906  
    SetConnectionProperties operations 684  
    SetUserExtendedProperties  
        operations 685  
    SetUserPreference operations 685  
**userName** parameter  
    delete status page 838  
    request detail page 840  
    requests index page 848  
**UserName** property 671  
**username** variable 43, 533, 607  
**UserNameList** element 796  
**UserNames** element 638, 639  
**userpassword** parameter 42  
**UserPermissions** element 735  
**UserPermissions** property 649  
**users**  
    adding multiple 571, 615  
    adding to groups 721  
    assigning privileges 768, 802, 909  
    assigning to groups 801  
    authenticating 532, 606, 902  
    building access control lists for 900  
    checking for existence of 903  
    creating 554, 610, 721  
        tutorial for 557  
    defining available features for 668  
    deleting 729, 788  
    developing RSSE applications for 888  
    displaying preferences for 827  
    getting access control lists for 899, 904  
    getting ACL templates for 646  
    getting assigned privileges for 772  
    getting authentication IDs for 15  
    getting home folders for 881  
    getting licensing options for 663, 810  
    getting passwords for 880  
    getting printer settings for 664  
    getting properties for 903, 905  
    getting security credentials for 880  
    getting security roles for 13, 16, 17, 906  
    managing volume data and 509  
    reassigning roles to 799  
    removing from groups 729  
    running log consolidator and 931  
    searching 682, 806, 809, 906  
    sending notifications to 688, 844, 848, 859,  
        905  
    setting home folders for 554, 805, 806  
    setting licensing options for 759  
    setting passwords for 554, 559, 667, 804  
    setting preferences for 554, 610, 806  
    setting viewing preferences for 559  
    updating licenses 802  
    updating passwords for 876  
    updating properties for 797, 800  
    updating roles for 799  
    validating credentials for 879  
    verifying 564  
    viewing error messages for 926  
**Users** element 683, 905, 907  
**Users** page (Management Console) 564  
**users** parameter 844  
**UserSearch** data type 809  
**UserSetting** element 667, 903  
**UseSOAPAction** parameter 576  
UTF-8 character encoding 39  
UTF-8 character set 504  
UTF-8 encoding 824

## V

**ValidateUserGroups** element 667  
**ValidUserGroups** element 668

Value element  
  LicenseOption type 759  
  NameValuePair type 762  
  ParameterValue type 767  
  PropertyValue type 773, 910  
value parameter 106  
value series (charts)  
  getting maximum values 258  
  getting minimum values 258  
  interactive features and 103  
  setting value range for 261  
  three-dimensional charts and 238  
valueData variable 106  
ValueFileType element 745  
ValueFileVersionName element 745  
valueIsNull value 229, 363  
ValueIsNullValue element 767  
values  
  *See also* data  
  assigning to data components 762  
  assigning to data types 496  
  changing parameter 199, 210  
  converting 196  
  downloading parameters and 186  
  filtering cross tabs and 438  
  filtering locale-specific data and 163  
  filtering top or bottom 384  
  generating summary 125  
  getting default 215  
  getting empty cell 462  
  getting level 459  
  getting list of 217  
  getting parameter 229, 362, 363, 398  
  getting series 257, 258  
  matching set of 163, 164, 845, 846  
  matching top or bottom 163  
  overwriting parameter 634  
  prompting for 210, 229, 232, 767  
  removing duplicate 270  
  retrieving counter 637, 639  
  returning duplicate 506  
  returning specific 175, 243  
  selecting 222  
  setting default 221, 765  
  setting display names for 200  
  setting empty cell 414, 464  
  setting level 459  
  setting parameter 226, 362, 471, 767  
  setting property 910  
  setting series 260  
  specifying null 233, 363, 471  
  suppressing duplicate 382  
  testing for null 164, 439, 470, 846  
valueSeriesName variable 106  
variables 849  
  callback functions and 38  
  unloading JavaScript 188, 299  
  UNIX environment 927  
  Windows environment 530  
vector graphics elements 275, 276, 278  
vector graphics shapes 593  
Vector Markup Language graphics  
  elements 275, 278, 279  
verifyFile function 882, 883, 886  
version attributes 762, 780, 811  
Version element  
  File type 735, 737  
  ObjectIdentifier type 764  
  RSSE Start operations 907  
version information 309, 312, 783  
version names 19, 312  
  *See also* VersionName element  
version numbers 735, 737  
version parameter  
  file or folder detail page 839  
Version property 649  
Versioning element 762  
VersioningOption data type 811  
VersionName element 735, 737, 762  
versionName parameter  
  execute report page 844  
  output page 853  
  submit job page 861  
VersionName property 649  
View element 773  
view parameter definitions 657  
view parameters  
  *See also* ViewParameter element  
  defining parameter types and 773  
  setting 223, 766, 767  
  setting attributes for 811  
view properties 677  
View service 508, 670, 783, 856  
Viewer class 44, 66, 90, 334

viewer classes 48, 135  
viewer components 44, 141, 334  
viewer event constants 357  
viewer getsavedsearch action 833  
viewer IDs 361, 467  
viewer objects 334, 400  
viewer page 835  
viewer preferences 805, 806, 908  
viewer savesearch action 833  
viewer servlet 868  
viewer variable 392  
viewer1 parameter 44  
ViewerException class 386  
viewers  
    accessing report content for 48, 338  
    adding interactive features to 103  
    building user interface for 47, 81, 297  
    determining status of 348  
    disabling 336  
    displaying charts in 106, 338, 358  
    displaying parameters in 67, 68  
    displaying reports in 44, 59, 66, 334  
    enabling interactive features for 338, 370, 372  
    getting browser size for 339  
    getting content for 49, 338, 339, 340, 341  
    getting file names for 344  
    getting margins for 340  
    getting size 343, 347  
    getting specific instance 139, 345  
    getting UI options for 345  
    handling events for 357  
    handling exceptions for 386  
    instantiating 44  
    launching 379  
    loading 44, 50, 90  
    reloading 354  
    resizing 154, 339, 343, 352  
    saving contents 348  
    scrolling in 356, 366, 370  
    selecting content in 368  
    sessions timing out and 434  
    setting focus for 349  
    setting margins for 349  
    setting size 349, 352, 353  
    setting UI options for 352  
    showing main menu in 379  
        showing toolbar help in 384  
        showing toolbars in 383  
        submitting requests for 354  
        switching views 382  
viewframeset action 828  
viewing  
    aggregate values 118  
    columns 291  
    cross tabs 118, 122, 390, 467  
    current jobs 856  
    dashboards 50  
    data 77, 460  
    data cubes 120, 485  
    data items 244  
    data series 259, 264, 272  
    dates 725  
    debugging messages 14  
    error messages 842  
    failed jobs 848  
    file properties 568  
    Flash objects 246, 249, 254  
    folders 302, 850  
    HTML output 593  
    HTML5 charts 273  
    Interactive Crosstabs features 128  
    label elements 283, 343  
    login page 822, 850  
    parameter groups 192  
    PDF documents 593, 812  
    pending jobs 857  
    report elements 48  
    report executables 850  
    report items 378  
    report parameters 67, 184, 210, 860  
    Reportlets 344  
    reports 44, 508, 868  
    standard charts 240  
    summary data 125, 445, 476, 480  
    table elements 291  
    table of contents 370, 371, 383  
    tables 45, 106, 285  
    text 294, 295, 344  
    toolbars 383, 485  
    tooltips 382  
Viewing element 783  
viewing events 920, 922  
viewing mode (dashboards) 353

viewing mode constants 353  
viewing parameters. *See* view parameters  
viewing preferences 559  
viewing preferences. *See* viewer preferences  
viewing service. *See* View service  
ViewMode element 764  
viewMode property 593  
ViewOperation element 812  
viewpage action 829  
ViewParameter data type 811  
ViewParameterList element 657  
ViewParameterValues element 677  
ViewPreference element 805, 806, 911  
ViewPreference property 908  
ViewProperties element 677  
ViewResourceGroupList element 659  
views, switching 382  
view-time parameters 218, 230, 231, 740  
Visibility element 717  
visible privilege 769  
Vista computers. *See* Windows systems  
Visual Basic development environments 600  
VML graphics elements 275, 278, 279  
volume administrators. *See* administrators  
Volume data type 813  
Volume element  
  PendingSyncJob type 768  
  ResourceGroup type 774  
  RunningJob type 778  
  Start operations 907  
volume failover errors 925  
volume file paths 59, 148  
volume health monitoring errors 925  
volume job purging errors 925  
volume names 12, 18, 864  
volume online or offline errors 925  
volume parameter 864  
  execute report page 845  
  submit job page 861  
  URIs 827  
volume partitions 635, 670  
volume profiles 331, 333, 881  
volume schemas. *See* schemas  
volume user activity errors 925  
volume variable 137, 140, 533  
VolumeDefault element 776  
VolumeName element 635  
volumename variable 607  
VolumeProfile parameter 827, 865  
VolumeProfile.xml 865  
VolumeProperties element 665  
VolumeProperties property 665  
volumes  
  accessing items in 508  
  changing 626  
  connecting to 827  
  creating folders for 836  
  creating users for 554, 557, 571, 610, 615  
  deleting objects in 841  
  deploying reports to 901–902  
  developing administration applications for 492, 610  
  developing administration operations for 701  
  downloading files from 587, 619  
  executing commands for 635  
  getting current 881  
  getting information about 509, 838, 855  
  getting properties for 664  
  getting specific 331  
  integrating third-party reports with 492  
  integrating with external security sources 888, 891, 896  
  integrating with RSSE applications 890  
  pinging 670  
  retrieving ACLs from 899, 901  
  searching 566, 611  
  sending requests to 505, 509, 534, 555  
  setting profiles for 333  
  specifying 333, 509, 864  
  testing connections to 855, 881  
  uploading files to 573, 617, 692  
  tutorial for 577  
  viewing error log entries for 925  
  writing reports to 845, 861  
Volumes element 668  
—vp parameter 865

## W

wait intervals 635  
wait parameter 845  
wait values 845  
WaitForEvent element 653, 655, 690

WaitForExecuteReport operations 693  
waitforreportexecution action 834  
WaitTime element 635  
WARN logging level 889  
warnings 924  
web applications 10  
*See also* applications  
  customizing content for 132  
  deploying 874  
  protecting corporate data and 874  
  web services and 493  
web browser content panels 356  
web browser exceptions 181  
web browser windows 339  
web browsers 17, 865  
  displaying file types in 792  
  displaying reports and 66  
  encoding and 823, 824  
  getting exception descriptions for 181  
  initializing HTTP sessions for 40  
  integrating with reporting services 41  
  issuing URLs and 826  
  loading web pages for 822  
  redirecting 851, 859  
  rendering output for 39, 275  
  specifying 812  
  viewer events and 106  
web icons 739, 792  
web pages  
  accessing class libraries for 38, 132  
  adding interactive features to 124  
  adding JavaScript functions to 39  
  adding report components to 136  
  customizing 38  
  developing 38  
  displaying cross tabs in 390  
  displaying dashboards in 50, 51  
  displaying reports as 141, 334  
  embedding report parameters in 66, 68  
  embedding reports in 38, 39  
  enabling SVG support for 352, 408  
  loading 822  
  retrieving data for 38, 76, 168  
web service applications 493, 897  
  *See also* IDAPI applications  
web service attributes 493, 494  
web service connections 179  
web service definitions 498–499  
Web Service Description Language. *See*  
  WSDL  
web service interfaces 605  
web service names 527, 605  
web service namespace 495  
web service ports 527, 605  
web services  
  accessing 492, 493, 498, 502  
  authenticating users for 41  
  building interfaces for 528, 605  
  closing connections to 142  
  defining operations for 493, 498  
  displaying dashboards and 153  
  enabling Open Security 896  
  encapsulating data for 38  
  getting default URL for 138  
  initializing connections for 40  
  integrating web pages with 41  
  integrating with 492, 493, 516, 600  
  linking parameters to 193  
  opening connections for 38, 139  
  providing secure sessions for 42  
  retrieving data from 38  
  running remote 528  
  running RSSE applications as 888, 890, 898  
  sending SOAP messages over 209  
  setting content types for 504  
  setting URLs for 301, 351, 407  
  specifying 498  
web services messaging framework. *See*  
  SOAP  
web services system 603  
web.xml 864, 868  
Weekly data type 814  
Weekly element 755  
well-formed messages 502  
whitespace characters 865  
wildcard characters 566, 612  
wildcards 857  
Windows systems  
  running log consolidator for 927, 928, 930  
  running logging extensions for 916, 918,  
    919  
  running sample applications for 530  
  sending notifications over 492  
WithLicenseOption element 810

WithoutDynamicPickList element 657  
WithRightsToChannelId element 809  
WithRightsToChannelName element 808  
WithRoleId element 810  
WithRoleName element 810  
WithUserGroupId element 810  
WithUserName element 810  
Word document output formats 337  
work unit licenses 775  
workgroup repository type 333  
working folders 836, 849  
workingFolder parameter 836  
WorkingFolderId element  
  CopyFile operations 718  
  CreateFolder operations 720  
  DeleteFile operations 726  
  MoveFile operations 761  
  SelectFiles operations 675  
  UpdateFile operations 789  
workingFolderID parameter 836  
WorkingFolderName element  
  CopyFile operations 718  
  CreateFolder operations 720  
  DeleteFile operations 726  
  MoveFile operations 761  
  SelectFiles operations 675  
  UpdateFile operations 789  
workingFolderName parameter 836  
worksheets 364  
workspace directories 745  
WorkUnitType element 775  
WorkUnitTypes element 659  
wr action 829  
write privilege 769  
WriteFile action 669  
WSDL (defined) 493  
WSDL documents  
  *See also* WSDL files  
  accessing 517  
  Apache Axis clients and 516, 524, 527  
  displaying 499  
  generating C# classes from 603  
  generating JavaBeans and 525  
  Microsoft .NET clients and 600, 605  
  subclassed IDAPI classes and 502  
WSDL elements 525, 604  
  *See also* XML elements

WSDL files 493, 495, 499, 601  
WSDL interface 601  
WSDL schemas  
  case sensitivity for 493  
  creating 493–499  
  defining data type arrays and 704  
  developing IDAPI applications and 496  
  developing web service applications  
    and 493, 516, 600  
  development environments for 499  
  omitting responses and 497  
  structuring operations in 497  
  XML namespaces and 496, 505, 506  
WSDL type definitions 525, 603, 695  
WSDL2Java package 516  
  *See also* code emitter  
wsdlns namespace prefix 495

## X

x-axis labels (charts) 264  
x-axis values (charts)  
  adding interactive features to 103  
  returning maximum value for 257  
  returning minimum value for 258  
  setting data points for 272  
  setting range for 260  
Xerces XML Parser 518  
XML attributes. *See* attributes  
XML code 22, 502, 927  
  *See also* code  
XML documents 505, 603  
XML element (information objects) 742  
XML elements  
  binding definitions and 498  
  case sensitivity for 493  
  defining data types and 495  
  mapping to Java types 526  
  multiple data sources and 506  
  qualifying 495  
  SOAP headers and 507  
  SOAP messages and 493, 496, 502  
XML formats 742, 812  
XML namespace 496, 505, 506  
XML Parser 518  
XML parsing error messages 926  
XML reports. *See* XML documents

XML schema standard 496  
XML schemas  
*See also* WSDL schemas  
namespace prefixes in 495, 506  
web services and 493  
XML type descriptors 526  
XP computers. *See* Windows systems  
xsd namespace prefix 495, 496, 506  
xsi namespace prefix 506, 603  
xsi type attributes 576  
XTabAnalyzer class 122, 390, 394  
xtabAnalyzer components 141  
XTabAnalyzer event constants 434  
XTabAnalyzer exception objects 435

## Y

*y*-axis labels (charts) 264  
*y*-axis values (charts)  
adding interactive features to 103  
converting chart dimensions and 238  
returning maximum value for 258  
returning minimum value for 258  
setting data points for 272  
setting value range for 261

## Z

ZipFilePath element 666  
zooming 367