



ACTUATE.  
The BIRT Company™



BIRT iHub

## Building Mobile Applications Using BIRT APIs

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 - 2015 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.opentext.com](http://www.opentext.com)  
[www.actuate.com](http://www.actuate.com)

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

Actuate Corporation trademarks and registered trademarks include:

Actuate, ActuateOne, the Actuate logo, Archived Data Analytics, BIRT, BIRT 360, BIRT Analytics, The BIRT Company, BIRT 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](http://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](http://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](http://www.entypo.com)): Entypo Pictogram Suite. Castor ([www.castor.org](http://www.castor.org)), ExoLab Project ([www.exolab.org](http://www.exolab.org)), and Intalio, Inc. ([www.intalio.org](http://www.intalio.org)): Castor. Alessandro Colantonio: CONCISE Bitmap Library. d3-cloud. Day Management AG: Content Repository for Java. Dygraphs Gallery. Eclipse Foundation, Inc. ([www.eclipse.org](http://www.eclipse.org)): Babel, Data Tools Platform (DTP) ODA, Eclipse SDK, Graphics Editor Framework (GEF), Eclipse Modeling Framework (EMF), Jetty, and Eclipse Web Tools Platform (WTP). Bits Per Second, Ltd. and Graphics Server Technologies, L.P.: Graphics Server. Dave Gandy: Font Awesome. Gargoyle Software Inc.: HtmlUnit. GNU Project: GNU Regular Expression. Google Charts. Groovy project ([groovy.codehaus.org](http://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](http://code.google.com)): Protocol Buffer. H2 Database: H2 database. IDAutomation.com, Inc.: IDAutomation. IDRolutions Ltd.: JPedal JBIG2. InfoSoft Global (P) Ltd.: FusionCharts, FusionMaps, FusionWidgets, PowerCharts. InfoVis Toolkit. Matt Inger ([sourceforge.net](http://sourceforge.net)): Ant-Contrib. Matt Ingenthron, Eric D. Lambert, and Dustin Sallings ([code.google.com](http://code.google.com)): Spymemcached. International Components for Unicode (ICU): ICU library. JCraft, Inc.: JSch. jQuery: jQuery, jQuery Sparklines. Yuri Kanivets ([code.google.com](http://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. Membrane SOA Model. MetaStuff: dom4j. Microsoft Corporation (Microsoft Developer Network): CompoundDocument Library. Mozilla: Mozilla XML Parser. MySQL Americas, Inc.: MySQL Connector/J. Netscape Communications Corporation, Inc.: Rhino. NodeJS. nullsoft project: Nullsoft Scriptable Install System. OOPS Consultancy: XMLTask. OpenSSL Project: OpenSSL. Oracle Corporation: Berkeley DB, Java Advanced Imaging, JAXB, Java SE Development Kit (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. Raphael. RequireJS. Rogue Wave Software, Inc.: Rogue Wave Library SourcePro Core, tools.h++. Sencha Inc.: Extjs, Sencha Touch. Shibboleth Consortium: OpenSAML, Shibboleth Identity Provider. Matteo Spinelli: iscroll. StAX Project ([stax.codehaus.org](http://stax.codehaus.org)): Streaming API for XML (StAX). Sam Stephenson ([prototype.conio.net](http://prototype.conio.net)): prototype.js. SWFObject ([code.google.com](http://code.google.com)): SWFObject. ThimbleWare, Inc.: JMemcached. Twitter: Twitter Bootstrap. VMware: Hyperic SIGAR. Woodstox Project ([woodstox.codehaus.org](http://woodstox.codehaus.org)): Woodstox Fast XML processor (wstx-asl). World Wide Web Consortium (W3C) (MIT, ERCIM, Keio), Flute, JTIty, Simple API for CSS. XFree86 Project, Inc.: ([www.xfree86.org](http://www.xfree86.org)): xvfb. ZXing Project ([code.google.com](http://code.google.com)): ZXing.

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

Document No. 141215-2-431301 June 23, 2015

# Contents

<b>About Building Mobile Applications Using BIRT APIs</b> .....	<b>iv</b>
Chapter 1	
<b>Introducing BIRT APIs for applications</b> .....	<b>2</b>
Using BIRT REST API and JavaScript API in applications .....	3
About mobile applications .....	4
About mobile application development .....	4
Introducing the BIRT Gazetteer example application .....	5
Accessing source code and resources .....	10
About Xcode project files .....	13
About resources used by the example application .....	15
Chapter 2	
<b>Understanding BIRT and mobile tools</b> .....	<b>18</b>
Overview of BIRT iHub Visualization Platform .....	19
Considering which Actuate API to use .....	19
About the representational state transfer API .....	20
About the JavaScript API .....	20
About the Information Delivery API .....	21
Using BIRT Designer Professional for mobile results .....	22
IBuilding the mobile application source code .....	23
Introducing GitHub .....	23
Chapter 3	
<b>Integrating REST API</b> .....	<b>26</b>
Reviewing REST API integration .....	27
Authenticating with REST API .....	28
Displaying a list with REST API .....	30
Displaying data visualizations with REST API .....	31
Chapter 4	
<b>Integrating JavaScript API</b> .....	<b>36</b>
Reviewing JSAPI integration .....	37
Updating JavaScript in a web view .....	37
Displaying BIRT designs in a web view .....	39
Chapter 5	
<b>Extending mobile functionality</b> .....	<b>42</b>
Optimizing BIRT content for mobile viewing .....	43
Accessing mobile device features and applications .....	43

Using external authentication .....	44
Changing application default values .....	44
Customizing web view options .....	44
Additional optimizations .....	45
 Chapter 6	
<b>Using developer resources .....</b>	<b>48</b>
Using Actuate documentation .....	49
Visiting the Actuate developer site .....	51
Using the developer site for the mobile platform .....	52
About additional REST API resources .....	52
 <b>Index .....</b>	<b>53</b>

# About Building Mobile Applications Using BIRT APIs

---

*Building Mobile Applications Using BIRT APIs* provides information about using the Actuate REST API in software applications for mobile devices. The sections in this guide are:

- *About Building Mobile Applications Using BIRT APIs.* This chapter provides an overview of this guide.
- *Chapter 1. Introducing BIRT APIs for applications.* This chapter introduces the different types of mobile applications and introduces the BIRT Gazetteer application for iOS devices.
- *Chapter 2. Understanding BIRT and mobile tools.* This chapter introduces the tools used to build the BIRT Gazetteer sample application.
- *Chapter 3. Integrating REST API.* This chapter discusses methods to integrate the REST API in an application.
- *Chapter 4. Integrating JavaScript API.* This chapter discusses methods to integrate the JavaScript API in an application.
- *Chapter 5. Extending mobile functionality.* This chapter discusses possible customization of the BIRT Gazetteer application.
- *Chapter 6. Using developer resources.* This chapter lists resources to learn more about REST API.



# 1

## Introducing BIRT APIs for applications

This chapter contains the following topics:

- Using BIRT REST API and JavaScript API in applications
- Introducing the BIRT Gazetteer example application
- Accessing source code and resources
- About Xcode project files
- About resources used by the example application

---

## Using BIRT REST API and JavaScript API in applications

This chapter discusses how to incorporate BIRT data objects and reports into your mobile application. The BIRT iHub supports application development using the REST API and JavaScript API (JSAPI). You can use one or both of these APIs to integrate BIRT visualizations and access data files stored in BIRT iHub servers.

The REST API supports:

- Authenticating users
- Searching for BIRT files
- Running jobs from BIRT designs with selected parameters and locales
- Downloading files
- Downloading reports in PDF and Excel formats
- Downloading and filtering data in JSON or CSV format

The JSAPI supports:

- Embedding interactive BIRT visualizations in web pages
- Handling scripted events within BIRT reports or BIRT report elements
- Accessing table of contents and parameters in BIRT reports
- Operating the BIRT Interactive Viewer and Crosstabs

The BIRT files contain your data and report templates. Use these APIs to access and generate BIRT content, enabling your application to display secure, interactive data visualizations in any programming language that supports REST and JavaScript.

---

## Introducing the BIRT Gazetteer example application

This example illustrates how to integrate BIRT iHub resources into a native mobile application using iOS Objective-C. Two BIRT APIs—the REST API and the JavaScript API (JSAPI)—retrieve data and visualizations from a demonstration BIRT iHub 3.1 server. The iHub server resources used by this example are included with the source code.

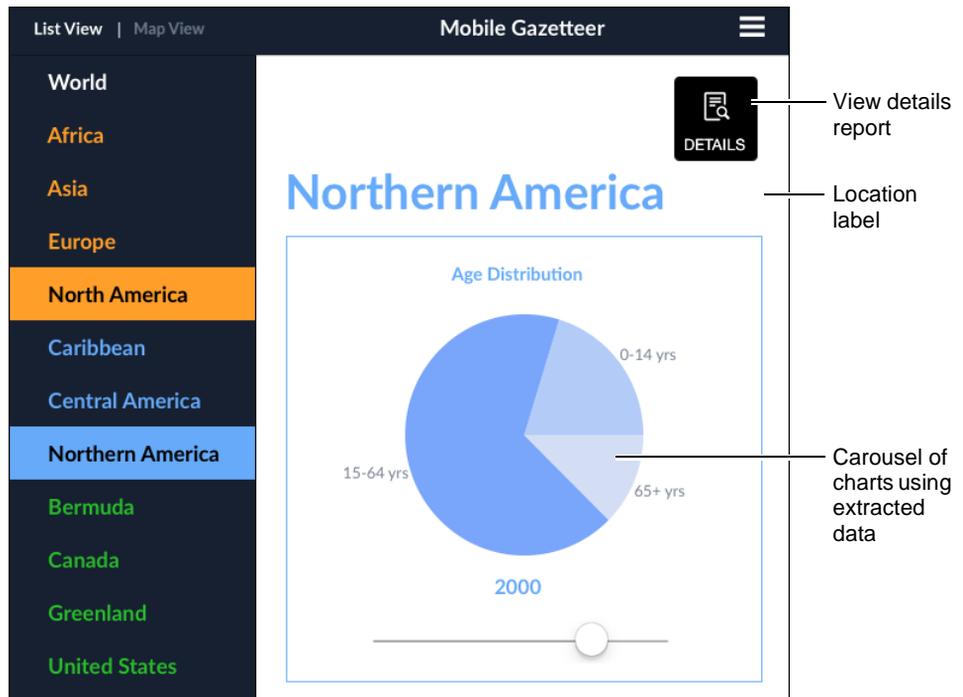
This example application demonstrates the following functionality:

- User authentication to a user account residing on a BIRT iHub server
- Requesting and setting values of parameters in BIRT reports

- Building the following interactive content:
  - Hierarchical list of parameter values
  - Global map containing location data and links to additional reports
- Extracting data for display as text
- Extracting data for display in third-party visualizations
- Sending values to an Objective-C UIWebView
- Displaying a report item from a BIRT design file in an Objective-C UIWebView
- Displaying a full BIRT report in an Objective-C UIWebView
- Display the appropriate report design for current device orientation

After using REST API to extract location names from a BIRT file, this application builds touch-enabled navigation links. When a user selects a link, Objective-C code passes the information to the embedded web view for display using JSAPI. The appropriate BIRT content is displayed for the current device orientation, portrait or landscape.

Figure 1-1 shows the application’s list interface.



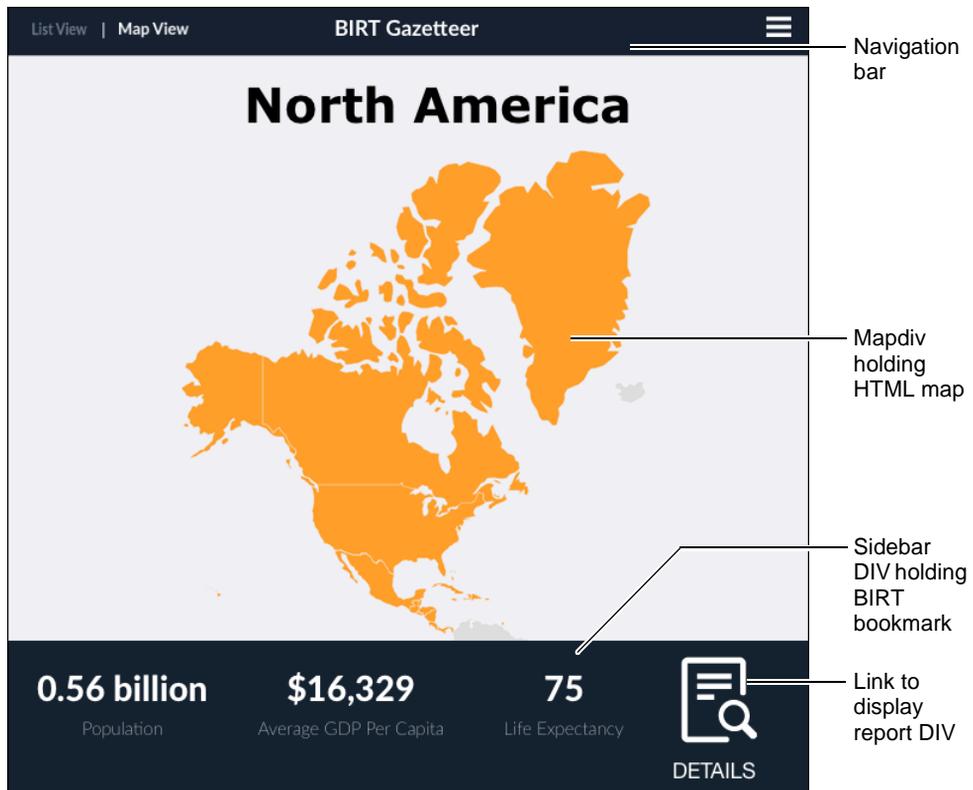
**Figure 1-1** User interface displaying a list, chart, and link to BIRT reports

An application can extract data from iHub using the REST API and send the values to third-party data visualizations, such as a chart or map. BIRT visualizations, such as a chart, table or a full page BIRT report display in interactive web views using JSAPI.

This example enables the JavaScript visualization to act as a tool to select a BIRT report for display. As a result, both the reports and third-party JavaScript visualizations use the same web view. When a full page BIRT report is displayed, the other JavaScript content is hidden using CSS.

Figure 1-2 contains the following HTML DIV elements:

- Mapdiv, which displays the JavaScript map
- Sidebar, which displays a BIRT bookmark
- Report, which displays a selected BIRT report



**Figure 1-2** Map and BIRT bookmark in web view

This web view also contains an image button that runs the loadReport JavaScript function. This function displays a report in the report DIV and hides the container div element that contains the JavaScript map and BIRT bookmark.

---

## Accessing source code and resources

The source code and BIRT resources to build the example iOS Objective-C application are available from the Actuate GitHub web site at the following URL:

<https://github.com/ActuateBIRT/GazetteerExample>

BIRT Gazetteer uses the following third-party chart libraries:

- [HTTPS://github.com/AlexandrGraschenkov/MagicPie](https://github.com/AlexandrGraschenkov/MagicPie)
- [HTTPS://github.com/zhuhuihuihui/Echart](https://github.com/zhuhuihuihui/Echart)
- [HTTPS://www.amcharts.com](https://www.amcharts.com)

If you want to distribute or use any of these libraries, check the library's web site for licensing information.

---

## About Xcode project files

The following is a general overview of the BIRT Gazetteer source code:

- Framework files to display a menu and table view of the application in iPad
  - BIRTAppDelegate  
Manages basic application tasks occurring during start-up and shutdown
  - BIRTConstants  
Contains default values for iHub URLs and volume paths
  - BIRTDetailViewController  
Contains the overview charts displayed in a split view pane
  - BIRTLoginViewController  
Contains the application login page
  - BIRTMapViewController  
Manages the MapView page and the content displayed in a web view
  - BIRTMasterViewController  
Contains the location list, displayed in a split view pane
  - BIRTPopupViewController  
Opens the pop-up menu for aboutUs/sourceData/logOut
  - BIRTSelectionProtocol.h  
Defines user authentication and user selections in the application

- BIRTSourceDataViewController  
Contains Source Data page font color and other confirmations
- BIRTSplitViewController  
Contains the split view page that displays the location list in one pane and overview charts in another pane
- Files to manage the slider and page control of the three overview charts
  - BIRTPagerView
  - BIRTUISlider
- Files to handle the About us page
  - BIRTExtUrlViewController
  - BIRTAboutUsViewController
- Main.storyboard file to show and handle application interactive work flow
- Images.xcassets containing the application icon and launch image files
- /mapView folder containing the HTML and JavaScript files to display a map
- /jsapi folder for jsapi.html, an HTML file containing Actuate JSAPI code
- /ReportView folder containing an iOS UIView and controller file  
These files display BIRT reports using the jsapi.html.
- /Chart folder containing third-party charts
  - BIRTChartData file to manage REST API data for front page charts
  - /BarChart folder containing a UIView that loads a line chart from a BIRT report using JSAPI
  - /ColumnChart folder containing the EChart library folder and other files to support the column chart
  - /PieChart folder containing the MagicPie library folder and other files to support the pie chart
- /Resources folder containing fonts and images used in the application
  - /Font folder
  - /Images folder
- /Supporting Files folder containing general iOS application files
  - Gazetteer-Info.plist
  - InfoPlist.strings
  - Main.m
  - Gazetteer-Prefix.pch

---

## About resources used by the example application

The example, BIRT Gazetteer, uses the REST API to retrieve data from a BIRT data store installed in the iHub server. These resource files are included with the BIRT Gazetteer source code, stored on the Actuate GitHub site. Choose Download Zip on the GitHub web page to download these resources to your computer.

If you are using your own iHub server, install the world.data file in the \Resources\Data Objects folder of the iHub volume.

Report designs for the application are stored in the administrator's home folder in the volume. Install the following files into the \Home\administrator folder of the iHub volume:

- Continent Report Portrait.rptdesign
- Continent Report.rptdesign
- Country Report Portrait.rptdesign
- Country Report.rptdesign
- GDP per capita.rptdesign
- Map View Content.rptdesign
- Regional Report Portrait.rptdesign
- Regional Report.rptdesign
- World Report Portrait.rptdesign
- World Report.rptdesign

Each report design except the World Report uses parameters to filter data displayed in the report. These reports are then shown in a web view, similar to a web browser, when a user selects the Details button on the application.

The GDP per capita.rptdesign file includes bookmarked charts that are embedded in the chart carousel.

The Map View Content.rptdesign includes bookmarked tables of text values displayed next to the map view.

There is a BIRT design file for a portrait layout and landscape layout. Changing the orientation of the mobile device loads the appropriate design file.



# 2

## Understanding BIRT and mobile tools

This chapter contains the following topics:

- Overview of BIRT iHub Visualization Platform
- Considering which Actuate API to use
- Using BIRT Designer Professional for mobile results
- Introducing GitHub

---

## Overview of BIRT iHub Visualization Platform

BIRT iHub Visualization Platform is a browser-based solution for document delivery, data analysis and building reports. Visualization Platform enables users to securely access data in the following ways:

- View and share interactive reports and dashboards.
- Analyze data in cross tabs and tables.
- Extract data from caches and stores of data.

Visualization Platform includes a user administration console that enables administrators to manage user profiles, user groups, as well as authorizing user and group access to published files. A volume administration console enables administrators to manage volume level operations such as assigning volume license options, managing files, scheduling file creation jobs, and archiving files.

This product is a set of dynamic web pages that installs automatically when you install BIRT iHub. Alternatively, you can install BIRT iHub Visualization Platform as a stand-alone product.

---

## Considering which Actuate API to use

Actuate provides software development tools as a collection of APIs that support designing new Actuate applications or extending or customizing existing applications. Each API offers the developer different methods to access and control data, visualizations and iHub server functionality. The API that you use depends on what you need to do.

Actuate APIs libraries extend functionality in applications that provide API integration points. Actuate provides:

- Representational state transfer API (REST API). The REST API accesses and manages data and files built with Actuate BIRT technology. Use this API to manage and generate new documents, and to request data in the JSON format.
- JavaScript API (JSAPI). The JSAPI provides libraries for web and client-side visualizations using the JavaScript programming language. Use this API to render BIRT content in a web page.

### About the representational state transfer API

The Actuate REST API is an HTTP service that runs on a Node.js platform. This service interacts with BIRT content and files on an iHub server using URI requests such as:

```
http://<web server>:5000/ihub/v1/login
```

This API is installed with iHub and responds to RESTful web requests that uses HTTP methods such as GET, PUT, and DELETE. The REST API is a strategy for developing web and mobile components that are platform and language independent, require very little time to implement, and that use minimal client and server resources.

RESTful requests use a specific command set to access REST API resources, which simplifies implementations by providing access to essential functions and raw data. Actuate offers many APIs that provide broader functionality but they are implemented using specific tools or access resources in a wide array of formats and interfaces. The REST API provides maximum freedom for developers to create their own implementations.

The REST API employs uniform resource identifiers (URIs) references to convey user requests to the iHub system. URIs access iHub functionality, including generating and storing reports, browsing volume contents, extracting data from files and data sources, and managing users and credentials.

Mobile applications request RESTful content by sending URI requests to the REST service. The REST server module interprets REST requests and forwards them as SOAP requests to iHub. For example, iOS applications can use `NSURLConnection` object to request RESTful content, Android applications can use the `ApacheHttpClient` for Java, and JavaScript can use `XMLHttpRequest` or the jQuery AJAX library.

To view interactive visualizations such as filtering, drill down, and dashboards, use the Actuate JSAPI. For more information about using the REST API, see *Integrating Applications into BIRT iHub*.

## About the JavaScript API

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

The Actuate JavaScript API uses the Prototype JavaScript Framework. The following URI to an iHub server contains the Actuate JavaScript API library:

```
http://<web server>:8700/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 and 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.

Mobile applications integrate BIRT visualizations using JSAPI in a web view. A web view is a class or object that displays an HTML content such as a web page within a native application. The iHub server receives the JSAPI requests and sends HTML content for display in a selected HTML DIV element. For example, iOS applications can use UIWebView object to display JSAPI, Android applications can use the WebView class and JSAPI can display in HTML files using most web browsers.

For more information about using the JSAPI, see *Integrating Applications into BIRT iHub*.

---

## Using BIRT Designer Professional for mobile results

Actuate BIRT Designer Professional is a report designer for report developers who want to use the functionality provided by Actuate Corporation that enhances the Eclipse BIRT Report Designer.

You can use BIRT Designer Professional to designing the following content:

- BIRT visualizations that securely display data charts, cross tabs, maps, and tables
- Templates to export HTML, PDF, and Microsoft Excel file formats
- Structured data from databases, web services, XML files, and other data sources
- Custom data and visualization solutions using expressions and scripting

BIRT designs files query data sources and display charts, tables, cross tabs and maps interactively on web pages using the Actuate JSAPI. These designs can also be run and downloaded in formats such as Adobe PDF and Microsoft Excel using the Actuate REST API.

BIRT data object files can query multiple data sources and cache the data in data sets, data models, and data cubes for analysis and visual display in charts and maps. You can filter and retrieve data sets from a data object in the JSON format using REST API. You can also use data objects to provide data to BIRT designs and dashboards.

You can use the REST API to extract aggregated data when that data is grouped in BIRT report items. Each item in a BIRT report such as a chart, cross tab, and table can include a bookmark name to identify the item. The REST API uses the bookmark value to find the report item and then to extract the data displayed in the report item. For example, a bookmark named MapState can identify a cross

tab that summarizes population statistics about each state in a BIRT design file. You can use the REST API to find the bookmark name and extract the data summary in the JSON format for use in your application.

For more information about using BIRT Designer Professional, see *Actuate BIRT Application Developer Guide*.

---

## Introducing GitHub

GitHub is a web site that stores source code repositories for many public and private projects. The source code for BIRT Gazetteer is available at GitHub. You do not need an account with GitHub to download the source code for the BIRT Gazetteer, but you must have a user account to use the GitHub issue tracker or to submit comments or changes about the source code.

For more information about GitHub, visit the following URL:

<https://github.com/>



# Integrating REST API

This chapter contains the following topics:

- Reviewing REST API integration
- Authenticating with REST API
- Displaying a list with REST API
- Displaying data visualizations with REST API

---

## Reviewing REST API integration

The BIRT iHub server offers many RESTful URI endpoints to access stored resources on the server. This example uses Objective-C to make the following REST API requests:

- Authenticate the user to receive an authentication ID to attach to other REST API requests.
- Download a list of locations values that are used to build BIRT reports.
- Download data sets in JSON format for display in third-party visualizations.

Using Objective-C for iOS, a RESTful URI request to a resource is built using the `NSString` class. `NSURLConnection` sends the `NSString` to the iHub server.

An `NSDictionary` object is created from the iHub server's JSON formatted response using the `NSJSONSerialization` class. The Objective-C code uses these `NSDictionary` values to request additional resources, display available location names in a table, and to display data about a location in a text string or an embedded visualization such as a chart.

Using the Android SDK for Android mobile devices, you can select from multiple HTTP client libraries to make a RESTful URI request, such as the `HttpClient` from the Apache HTTPComponents project or Java's `HttpURLConnection` class. After making an HTTP request to a URI endpoint and checking for errors, the response from the iHub server is assigned to a string for additional processing.

The following REST API operations are used in BIRT Gazetteer:

- `/Login`  
Used in `BIRTLoginViewController.m` to return an `authId` for an authenticated user.
- `/Files`  
Used in `BIRTBarChartViewController.m` and `BIRTMasterViewController.m` to retrieve a file id by searching for the file name.
- `/Visuals`  
Used in `BIRTBarChartViewController.m` to search for bookmark names in a BIRT file.
- `/Dataobject`  
Used in `BIRTColumnChartViewController.m` and `BIRTMasterViewController.m` to extract values from a data set in a selected data store file.

---

## Authenticating with REST API

An `authId` is an authentication identifier passed back from iHub after successful authentication and is required for all subsequent REST API requests.

To generate the `authId` token, use a POST request for the `/login` resource with a `username` query parameter. Other parameters for `/login` are optional. An HTTP request does not encrypt the password field, so always use an HTTPS request for `/login`. For instructions to enable HTTPS support for REST API see *Integrating Applications into BIRT iHub*.

When successful, the REST API request returns an authentication identifier, `authId`. A REST API authentication identifier remains valid for 24 hours by default.

This example uses Objective-C code to make an authentication request from the iHub server. After collecting the username and password from the application's user interface, these values are stored in an `NSData` object, with the following code:

```
-(void) login {
    NSString *yourName = self.username.text;
    NSString *password = self.password.text;

    NSString *post = [NSString stringWithFormat:
        @"username=%@&password=%@", yourName, password ];

    NSData *postData =
        [post dataUsingEncoding:NSUTF8StringEncoding
         allowLossyConversion:YES];
    NSString *postLength = [NSString stringWithFormat:@"%lu",
        (unsigned long) [postData length]];
}
```

Then the URI to authenticate a user with the REST API is built. An `NSMutableURLRequest` uses the URI and attaches the authentication values as the body of the HTTP POST request, with the following code:

```
NSMutableURLRequest *request = [NSMutableURLRequest
    requestWithURL:[NSURL URLWithString:[NSString
        stringWithFormat:@"%s",
        REST_API_URL,
        @"login"]]];
[request setHTTPMethod:@"POST"];
[request setHTTPBody:postData];
NSError *error;
NSURLResponse *response;
NSData *urlData = [NSURLConnection
    sendSynchronousRequest:request returningResponse:&response
    error:&error];
```

After checking for errors, the JSON formatted response from the iHub server is then converted into a Foundation object using the `NSJSONSerialization` class. This Foundation object is converted into an `NSDictionary` for use in other REST API requests, as shown in the following code:

```
NSDictionary *loginResponse = [NSJSONSerialization
    JSONObjectWithData:urlData options:NSJSONReadingMutableLeaves
    error:nil];
    _authId = [loginResponse objectForKey:@"AuthId"];
```

See the source code for the complete example.

The source code to send RESTful requests using the Android SDK depends on the HTTP library that you are using. For example, if you are using Java's `URLConnection` class, your RESTful request to the login endpoint can be similar to the following code:

```
JSONObject authIDResponse = WebserviceUtil.requestWebService(
    "http://ihubserver:5000/ihub/v1/login");

public static JSONObject requestWebService(String targetURL) {
    disableConnectionReuseIfNecessary();

    URL restURL = new URL(targetURL);

    String formValues =
        "user=" + URLEncoder.encode("demo", "UTF-8") +
        "&password=" + URLEncoder.encode("demo", "UTF-8");

    HttpURLConnection restConnection = (HttpURLConnection)
        restURL.openConnection();
    restConnection.setRequestMethod("POST");
    restConnection.setRequestProperty("Accept", "application/
        json");
    restConnection.setFixedLengthStreamingMode(formValues.getBytes(
        ).length);
    restConnection.setRequestProperty("Content-Type",
        "application/x-www-form-urlencoded");

    restConnection.setConnectTimeout(15000);
    restConnection.setReadTimeout(10000);
    restConnection.setDoOutput(true);

    //PrintWriter formStream = new
    //PrintWriter(restConnection.getOutputStream());
    //formStream.print(formValues);
    //formStream.close();

    DataOutputStream formStream = new DataOutputStream (
        restConnection.getOutputStream ());
    formStream.writeBytes (formValues);
    formStream.flush ();
```

```

formStream.close ();

if (restConnection.getResponseCode() != 200) {
    throw new RuntimeException("Connection Error code : " +
        restConnection.getResponseCode());
}

InputStream restResponse = new BufferedInputStream(
    restConnection.getInputStream());

responseText =
    EntityUtils.toString(httpresponse.getEntity());

return new JSONObject (getResponseText (restResponse));

restConnection.disconnect ();
}

private static String getResponseText (InputStream inStream) {
    // http://weblogs.java.net/blog/pat/archive/2004/10/
    // stupid_scanner_1.html
    return new Scanner (inStream) .useDelimiter ("\\A" ) .next ();
}

```

---

## Displaying a list with REST API

This example builds a navigation list of locations, BIRT Gazetteer requests values from a data set from the world.data file. The following is a general overview of that process:

- Search for the world.data file by building a URI using the REST API.
- Append the authentication identifier to the end of the URI and send it to an iHub server.
- Extract the file id from the file search request.
- Use the file id to make a second REST API request that retrieves a specific data set from the world.data file.
- Parse the JSON response from the iHub server into a collection of name and value pairs, such as an iOS NSDictionary object or an Java JSONObject for Android.
- Retrieve from the collection of name and value pairs the names of continents, regions, and countries.
- Build the list view navigation of the application using these names.

An example of this process using Objective-C is shown in the following selected source code from BIRTMasterViewController.m:

```

NSString *fileId;
@try {
    NSString *fileName = [NSString
        stringWithFormat:@"%%%@", DATA_OBJECT_FOLDER, @"world.Data"];
    NSString *getUrl = [NSString stringWithFormat:[NSString
        stringWithFormat:@"%%%@", REST_API_URL,
        @"files?search=%@&authId=%@"],
        [fileName stringByAddingPercentEscapesUsingEncoding
            ::NSUTF8StringEncoding], self.authId];

    NSURLRequest *urlRequest = [NSURLRequest
        requestWithURL:[NSURL URLWithString:getUrl]];
    NSError *urlConnectionError;
    NSURLResponse *urlResponse;
    NSData *data = [NSURLConnection
        sendSynchronousRequest:urlRequest
        returningResponse:&urlResponse error:&urlConnectionError];

    NSError *error;
    NSDictionary* response = [NSJSONSerialization
        JSONObjectWithData:data options:NSJSONReadingMutableLeaves
        error:&error];

    NSArray * responseArr = response[@"ItemList"][@"File"];
    if (responseArr == nil || [responseArr count] == 0) {
        [self showAlert:@"Unable to get the file"];
    } else {
        self.regionData = response[@"data"];
        self.arrayOriginal=responseArr;
    }...
}

```

The region data is then used to make a hierarchical list that is used to populate the UITableView in the MasterView.

See the source code for the complete example.

---

## Displaying data visualizations with REST API

This example collects data for display in third-party visualization code using an chart data object of strings, arrays, and dictionaries. When a user makes a selection from a list of locations, the chart data object and user interface update with data about the selected location.

The following overview describes this functionality in the iOS example application:

- An object of strings, arrays, and dictionaries is defined in BIRTChartData.h. This object contains values associated with the user selection.
- User makes a selection from the list of locations and triggers an update to the chart data object. The user selection calls updateView.
- The updateView function creates a new BIRTChartData object and loads it with data about the selected location.
- The BIRTChartData object is passed to each of the chart view controllers for display, as shown in the following code:

The following Objective-C code shows the BIRTChartData object used in a chart:

```

-(void) updateView {
    _titleLabel.text = self.userData[@"name"];

    BIRTChartData *chartData = [[BIRTChartData alloc] init];
    [chartData setWorldData:self.worldData];
    [chartData setSelectedYear:@"2000"];
    [chartData setUserName:self.userName];
    [chartData setPassword:self.password];
    [chartData setDataObjectId:_dataObjectId];
    [chartData setContAbb:self.userData[@"Cont_Abb"]];
    [chartData setRegAbb:self.userData[@"Reg_Abb"]];
    [chartData setCountAbb:self.userData[@"Count_Abb"]];
    [chartData setAuthId:self.authId];
    [chartData setUserData:self.userData];
    [chartData setContAbbs:self.contAbbs];
    for (UIViewController *child in self.childViewControllers) {
        if ([child isKindOfClass:[BIRTColumnChartViewController
class]]) {
            BIRTColumnChartViewController *chart =
            (BIRTColumnChartViewController *)child;
            [chart setChartData:chartData];
            [chart setRefreshView:TRUE];
            [chart viewWillAppear:YES];
        }

        if ([child isKindOfClass:[BIRTBarChartViewController
class]]) {
            BIRTBarChartViewController *chart =
            (BIRTBarChartViewController *)child;
            [chart setChartData:chartData];
            [chart setRefreshView:TRUE];
            [chart viewWillAppear:YES];
        }

        if ([child isKindOfClass:[BIRTPieChartViewController
class]]) {

```

```
        BIRTPieChartViewController *chart =  
        (BIRTPieChartViewController *)child;  
        [chart deleteValues];  
        [chart setChartData:chartData];  
        [chart setRefreshView:YES];  
        [chart viewWillAppear:YES];  
    }  
}  
}
```

See the source code for the complete example.

# Integrating JavaScript API

This chapter contains the following topics:

- Reviewing JSAPI integration
- Updating JavaScript in a web view
- Displaying BIRT designs in a web view

---

## Reviewing JSAPI integration

This application uses JSAPI to display BIRT visualizations in embedded web views. A web view can render CSS, HTML, and JavaScript in the native code of the mobile application. JSAPI communicates with the iHub server using the authentication ID from the REST API login request. Native code injects values into JSAPI requests before they are sent. The JSAPI then downloads and displays interactive BIRT content about the selected location using a bookmarked chart and a full page report. Bookmarks are a method to identify content in a BIRT report.

This example uses the following Objective-C methods to communicate with the HTML content in a `UIWebView`:

- Replace string values in the embedded HTML file before loading the HTML file into the web view. This replacement uses the `NSString`'s `stringByReplacingOccurrencesOfString` function to find string values in the `jsapi.html` file and replace those values with the current values, such as the server URL, and the username and password.
- Call the `init JavaScript` function embedded in the web view and pass the values required to display the BIRT content. This call uses the `UIWebView`'s `stringByEvaluatingJavaScriptFromString` function to call the JavaScript function.

The `ReportView` folder of the Xcode project contains the Objective-C files that define the `UIWebView`. The `BIRTWebViewController.m` file organizes the values necessary to identify and find a report file, the data to be shown in that file, and how to load the JSAPI library from the current iHub server. These values are put into an `NSString` in the JSON format. The `BIRTWebViewController.m` file then loads the `jsapi.html` file into a `UIWebView` and sends the JSON values to the `init` function in the `jsapi.html` file.

Communication to the `UIWebView` occurs in one direction. The Objective-C code can send parameters and values to the content displayed in a `UIWebView`. The content in the `UIWebView` does not communicate back to Objective-C code in this application.

For more information about embedding BIRT visualizations in HTML see *Integrating Applications into BIRT iHub*.

---

## Updating JavaScript in a web view

Values required to load the Actuate JavaScript API from the server are written to the embedded file `jsapi.html` before that HTML file is loaded in a `UIWebView`.

The UIWebView then loads the embedded jsapi.html file, as shown in the following code:

```
[self loadHTML:@"jsapi.html"];
...
- (void) loadHTML:(NSString*) pageName
{
    NSRange range = [pageName rangeOfString:@"."];
    if (range.length > 0)
    {
        NSString *fileExt = [pageName
            substringFromIndex:range.location+1];
        NSString *fileName = [pageName
            substringToIndex:range.location];
        NSURL *url = [NSURL URLWithString:[NSBundle mainBundle]
            pathForResource:fileName ofType:fileExt inDirectory:@""];

        if (url != nil)
        {
            NSError *error;
            NSString *pageContent = [NSString
                stringWithContentsOfURL:url
                encoding:NSUTF8StringEncoding
                error:&error];
            NSString *finalContent = [pageContent
                stringByReplacingOccurrencesOfString:@"jsapiUrl"
                withString:[NSString stringWithFormat:@"%s%",
                    IHUB_SERVER_URL, @"iportal/jsapi"]];
            NSString *finalContent = [finalContent
                stringByReplacingOccurrencesOfString:@"{uName}"
                withString:[NSString stringWithFormat:@"%s%", "",
                    self.chartData.userName, ""]];
            if (self.chartData.password == nil) {
                finalContent = [finalContent
                    stringByReplacingOccurrencesOfString:@"{pwd}"
                    withString:self.chartData.password];
            } else {
                finalContent = [finalContent
                    stringByReplacingOccurrencesOfString:@"{pwd}"
                    withString:[NSString stringWithFormat:@"%s%", "",
                        self.chartData.password, ""]];
            }
            [self.webView loadHTMLString:finalContent baseURL:url];
        }
    }
}
```

Next, an NSString is built with the values required to display the BIRT content, such as the BIRT report name, the name of the selected location, and the viewing size of the BIRT content, as shown in the following code:

```
NSString *file = [NSString stringWithFormat:@"%@@%", REPORT_FOLDER,
    fileName];

jsData[@"report"] = file;
if (islandscape) {
    jsData[@"width"] = [NSString stringWithFormat:@"%f", 1024.0];
    jsData[@"height"] = [NSString stringWithFormat:@"%f", 768.0];
} else {
    jsData[@"width"] = [NSString stringWithFormat:@"%f", 768.0];
    jsData[@"height"] = [NSString stringWithFormat:@"%f", 960.0];
}

NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsData
    optionerror:&jsonError];

NSString *jsonStr = [[NSString alloc] initWithData:jsonData
    encoding:NSUTF8StringEncoding];
```

See the source code for the complete example.

---

## Displaying BIRT designs in a web view

Displaying the visualizations and layout of a BIRT design in HTML requires JavaScript and the Actuate JSAPI. Using JSAPI enables you to embed a BIRT design or BIRT document into an HTML web page. BIRT Gazetteer uses an iOS UIView class to embed a single HTML file, jsapi.html. This HTML file is included in the /jsapi folder of the Xcode project and contains the Actuate JSAPI necessary to display a BIRT design or document file. The JavaScript in jsapi.html enables you to select the BIRT design file to display and to pass parameter and bookmark values to the report before it is displayed.

Objective-C calls the init() JavaScript function in the HTML file and passes the prepared string, as shown in the following code:

```
[webView stringByEvaluatingJavaScriptFromString:[NSString
    stringWithFormat:@"%@@(%@);", @"init", jsonStr] ];
```

The JavaScript API then assigns all of the required values to download and display BIRT content and submits the request to the iHub server for display in the HTML DIV entity with the id name container. The following JavaScript code summarizes this request:

```
function initView( )
{
    try
```

```

    {
    var viewer = new actuate.Viewer( "container");
    viewer.setReportDesign( report );
    viewer.setWidth(data.width);
    viewer.setHeight(data.height);
    var options = new actuate.viewer.UIOptions( );
    options.enableToolBar(false);
    var parameterValues=[];

    if(data.continent != null) {
        var param=new actuate.viewer.impl.ParameterValue();
        param.setName("continent");
        param.setValue(data.continent);
        parameterValues.push(param);
    }
    ...
    if (parameterValues.length > 0 ) {
        viewer.setParameterValues(parameterValues);
    }
    if (data.bookmark != null) {
        viewer.setReportletBookmark(data.bookmark);
    }
    viewer.setUIOptions( options );
    viewer.submit();
    }
    ...
</script>
<body onload="">
<div id ="container">
</div>
</body>

```

See the source code for the complete example.



# Extending mobile functionality

This chapter contains the following topics:

- Optimizing BIRT content for mobile viewing
- Accessing mobile device features and applications
- Using external authentication
- Changing application default values
- Customizing web view options
- Additional optimizations

---

## Optimizing BIRT content for mobile viewing

The report designs displayed in BIRT Gazetteer use two files for each report. One file uses a master page width of 1024 pixels and a height of 768 pixels, used for devices in the landscape orientation. Another file uses a master page size with a width of 768 pixels and a height of 1024 pixels. This report is used for devices in the portrait orientation.

Content in both the landscape and portrait versions of a report are organized to make the best use of available space on the mobile screen.

You can limit the quantity of data transferred using the REST API by adding data groups in the SQL statements or using REST API to filter the data. Data sets downloaded using the REST API are not aggregated.

---

## Accessing mobile device features and applications

Each mobile platform includes features that you can integrate into your application, such as receiving push notifications, saving a date to a calendar application, saving data to a cloud server, or encrypting sensitive data.

For more information about adding capabilities to your iOS application, see the *iOS App Distribution Guide*, available from the following URL:

<https://developer.apple.com/library/>

Many applications in mobile devices use URI schemes to enable access from other applications. The URI scheme can launch an application and send parameter values to the application. You can use URI schemes in your Objective-C code or in your BIRT reports when building HTML hyperlinks. For example, an HTML link to call a phone number can look like the following code:

```
<a href="tel:1-888-422-8828">1-888-422-8828</a>
```

The same link in Objective-C looks like the following code:

```
tel:1-888-422-8828
```

Common URI schemes for web views include:

- Mail links to enable a hyperlink to send e-mails
- Phone links to enable a hyperlink to make a phone call
- SMS links to enable a hyperlink to send SMS messages
- Map links to enable a hyperlink to open the map application

---

## Using external authentication

The BIRT Gazetteer application uses the iHub server to authenticate users. The iHub server can use its own authentication database, connect to your LDAP or Active Directory user data, or use a single sign-on (SSO) service. See *Using BIRT iHub System Console* for more information about supported authentication services.

---

## Changing application default values

Default URLs and folder paths in an iHub server are set in the `BIRTConstants.m` file of the Xcode project. You can build the application for use with your own iHub server by changing the URL values in this file. You can also change the file path location of BIRT resources used in the application in this file.

The following NSStrings are contained in this file:

- `REST_API_URL`, you can change this value using the following URL format:  
`http://<iHub server name>:5000/ihub/v1/`
- `IHUB_SERVER_URL`, you can change this value using the following URL format:  
`http://<iHub server name>:8700/`
- `REPORT_FOLDER`, the file path where BIRT reports are located:  
`/Home/administrator`
- `DATA_OBJECT_FOLDER`, the file path where BIRT data objects are located:  
`/Resources/Data Objects`

---

## Customizing web view options

When you display interactive BIRT visualizations in a mobile application, you use a web view class to display Actuate JSAPI. The iOS `UIWebView` can also display files such as an Adobe PDF file, a Microsoft Excel file, or hyperlinks to other web pages, similar to the mobile version of the Safari web browser. The following examples are just a few of the ways you can customize the `UIWebView` using Objective-C to change what this web view can display:

- Only display selected file types.
- Only display web content from your network domain.

- Allow users to zoom content.
- Enable paginated web views.
- Restore web content after relaunching the application.
- Customize communication to the web view using the `stringByEvaluatingJavaScriptFromString` method for iOS Objective-C.
- Enable callbacks from the web view to your native operating system classes using your own URL scheme and a `UIWebViewDelegate` for iOS Objective-C or the `addJavascriptInterface` method for the Android `WebView`.
- Disable user selection and callouts of web content.
- Enable printing of the web view content.

---

## Additional optimizations

This example application demonstrates common integration techniques. Optimize your own code to make use of your software platform features and your enterprise requirements. For example, depending on the devices you expect to use and your application specifications, you might:

- Use SSL connections to secure user authentications and data.
- Aggregate data in SQL queries.
- Aggregate data in BIRT report items and identify the data with bookmarks.
- Use the latest software development kit supported by your devices, for example:
  - iOS7 introduces the new `NSURLSession` to replace `NSURLConnection`.
  - iOS8 introduces the new `WKWebView` class to replace `UIWebView`.
  - Android supports third-party HTTP clients such as `OkHttp`, `Retrofit`, and `Volley`.
- Store common resources, such as image files and JavaScript libraries, in your application.
- Store data and content offline in sqllite databases using iOS Core Data, iOS CloudKit, Android Google Cloud, or another storage service.

# 6

## Using developer resources

This chapter contains the following topics:

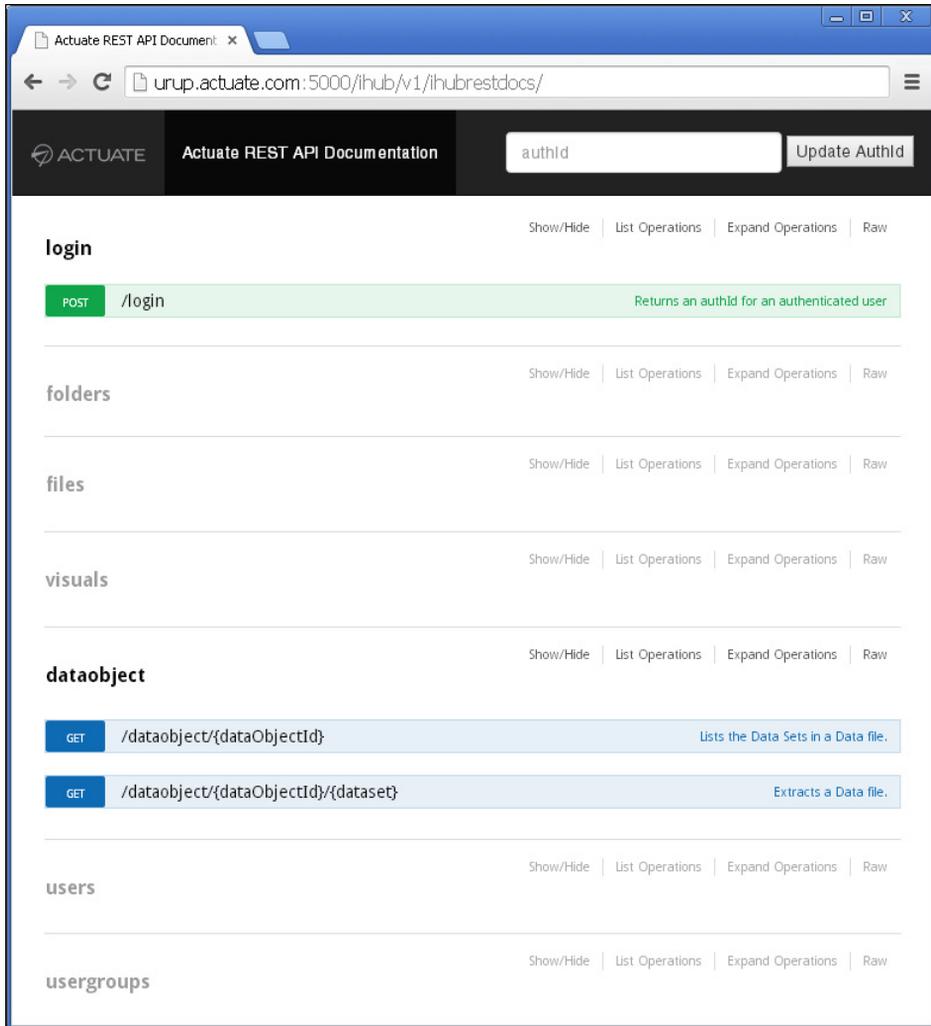
- Using Actuate documentation
- Visiting the Actuate developer site
- About additional REST API resources

# Using Actuate documentation

Interactive documentation for Actuate REST API operations is also installed with an iHub server. This documentation is accessible using a web browser at the following URL:

`http://<iHub server>:5000/ihub/v1/ihubrestdocs/`

Figure 6-1 shows the documentation included with an installation of iHub.



**Figure 6-1** Reviewing the Actuate REST API documentation

This documentation enables you to test the different URIs available in the Actuate REST API. To test a REST API operation, select one of the available operations, type parameter values, and then choose Try it out. Figure 6-2 shows the options to test the /login URI.

**login** Show/Hide List Operations Expand Operations Raw

**POST** /login Returns an authId for an authenticated user

**Implementation Notes**  
Authenticates specified user.

**Response Class**  
Model | Model Schema  
Auth

**Response Content Type** application/json ▼

**Parameters**

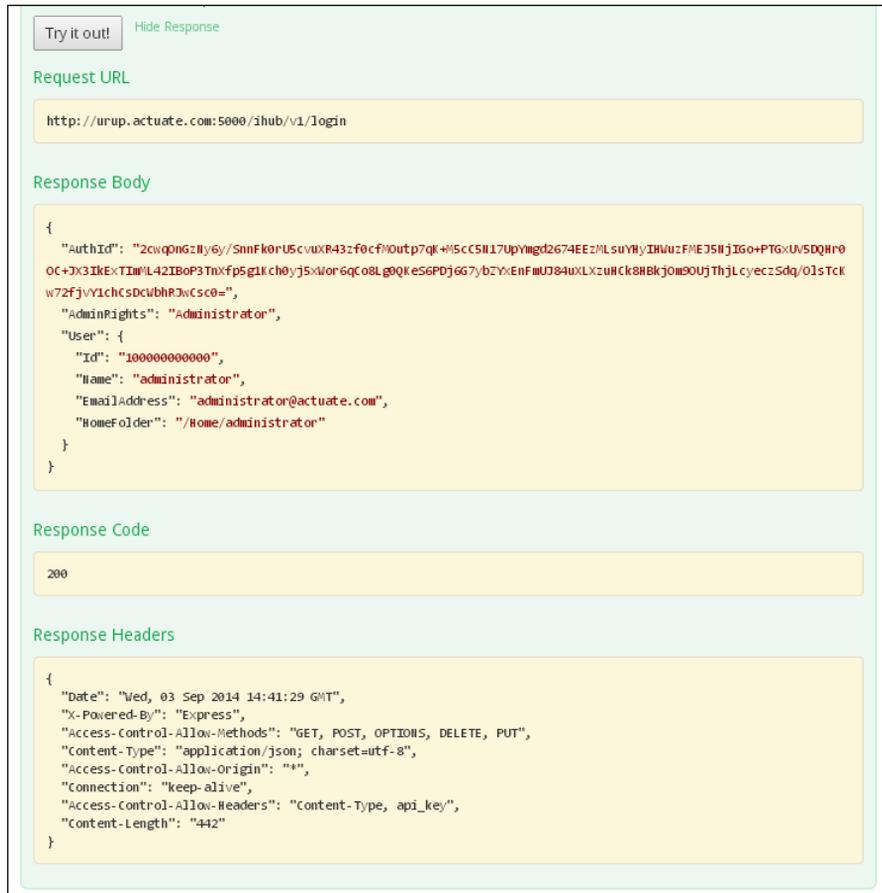
Parameter	Value	Description	Parameter Type	Data Type
username	<input type="text" value="[(required)]"/>	Required. A current iHub user name.	form	string
password	<input type="text"/>	The password corresponding to the user name.	form	string

**Error Status Codes**

HTTP Status Code	Reason
400	invalid username
400	invalid password

**Figure 6-2** Testing the login operation

After sending your test values to the selected operation, the documentation displays the response from iHub. Figure 6-3 shows the results when the username Administrator was sent to the /login URI.



**Figure 6-3** Reviewing results from the REST API

## Visiting the Actuate developer site

Additional information about integrating BIRT technology into applications is available at the following URL:

<http://developer.actuate.com/deployment-center/integrating-birt-into-applications/>

Forums for discussing BIRT technologies are available at the following URL:

<http://developer.actuate.com/community/forum/>

For more information about using the REST API and other Actuate APIs, see *Integrating Applications into BIRT iHub*, and the Actuate developer web site at the following URL:

<http://developer.actuate.com/>

---

## About additional REST API resources

There are many resources available on the internet discussing the use of RESTful web services. The following URLs are samples of some of those web sites:

<http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>

<https://www.ibm.com/developerworks/webservices/library/ws-restful/>



# Index

## A

- access rights
  - See also* privileges
- accessing
  - Information Console functionality 20
- ACL files
  - See also* access control lists
- ACLs. *See* access control lists
- ACS. *See* Caching service
- adding
  - display names. *See* display names
- Administrative operations
  - See also* administration operations
- administrators
  - See also* administration operations
- aggregate data. *See* aggregation
- aggregate functions. *See* aggregation functions
- aging rules. *See* archiving rules
- AIS. *See* Integration service
- application programming interfaces (APIs)
  - See also* Information Delivery API
- applications
  - building user interfaces for. *See* user interfaces
  - developing IDAPI. *See* IDAPI applications
  - developing RSSE. *See* RSSE applications
  - developing web. *See* web applications
- archive files
  - See also* jar files; war files
- archive rules. *See* archiving rules
- ArchiveRule objects
  - See also* archiving rules
- arguments. *See* command line arguments; parameters
- attributes
  - See also* properties
- autoarchiving. *See* archiving operations
- Axis servers. *See* Apache Axis environments

## B

- beans. *See* JavaBeans

- BIRT design files
  - See also* design files
- BIRT iHub. *See* iHub System
- BIRT Interactive Crosstabs. *See* Interactive Crosstabs
- BIRT report files
  - See also* report files
- BIRT reports
  - See also* reports
- BIRT Viewer
  - See also* report viewer
- browsers. *See* web browsers
- Business Intelligence and Reporting Tools.
  - See* BIRT

## C

- cache database. *See* Caching service database
- calculated columns
  - See also* computed columns
- character data. *See* strings
- character encoding. *See* encoding
- character encryption. *See* encryption
- character strings. *See* strings
- chart objects
  - See also* charts
- chart wizard launcher
  - See also* chart builder
- charts
  - See also* Flash charts; HTML5 charts
  - developing. *See* charting APIs
- client applications. *See* applications
- column headers
  - See also* column names
- column headings
  - See also* column names
- comma-separated values files. *See* CSV files
- completion notices
  - See also* notifications
- computed columns
  - See also* calculated columns
- conditions. *See* filter conditions; search conditions

- connection definition files. *See* database connection definition files
- connection handles. *See* ConnectionHandle element
- connections
  - setting properties for. *See* connection properties
- consolidator application. *See* log consolidator application
- creating
  - display names. *See* display names
  - IDAPI applications. *See* applications
- CSS files
  - See also* cascading style sheets

## D

- data
  - aggregating. *See* aggregation
  - extracting. *See* data extraction operations
  - localizing. *See* locales
- data analyzer component
  - See also* Interactive Crosstabs
- data charts viewer
  - See also* charts
- data cubes. *See* cubes
- data elements
  - See also* data items
- data fields
  - See also* columns
- data filters. *See* filters
- data items
  - See also* data
- data repositories
  - See also* Encyclopedia volumes
- data rows. *See* rows
- data set fields. *See* fields
- data sorters. *See* sorters
- database connection properties. *See* connection properties
- database drivers. *See* drivers
- database schemas. *See* schemas
- databases
  - See also* data sources
- DCD. *See* database connection definitions
- dependent files. *See* file dependencies
- developing

- charts. *See* charting APIs
- IDAPI applications. *See* IDAPI applications
- RSSE applications. *See* RSSE applications
- diagnostic information
  - See also* Ping operations
- directory paths. *See* paths
- display formats. *See* formats
- distributed iHub System. *See* clusters
- documentation
  - See also* help collections
- documents
  - See also* reports
- .dov files. *See* data object values files
- download operations
  - See also* downloading
- downloading
  - See also* download operations
- duplicating. *See* copying

## E

- elements. *See* report elements; XML elements
- e-mail
  - sending attachments with. *See* attachments
  - setting notification options for. *See* notifications
- events
  - handling. *See* event handlers
- execution requests. *See* ExecuteReport operations
- Extensible Markup Language. *See* XML

## F

- fields
  - See also* columns
- file attributes
  - See also* file properties
- file IDs
  - See also* FileId element
- file paths. *See* paths
- files
  - See also* report files
  - naming. *See* file names
  - setting properties for. *See* file properties
- finding data. *See* search operations
- folder paths. *See* paths
- formats

*See also* output formats  
functions  
*See also* callback functions; methods

## G

graphical user interfaces. *See* user interfaces  
graphics elements  
*See also* images  
graphs. *See* charts  
groups  
*See also* notification groups; resource groups  
GUI components  
*See also* user interfaces

## H

header elements (SOAP messages)  
*See also* SOAP headers  
hyperlinks  
*See also* URLs  
hypertext markup language. *See* HTML code  
HyperText Transfer Protocol. *See* HTTP

## I

IDAPI applications  
*See also* Information Delivery API  
iHub  
    sending requests over 20  
iHub clusters. *See* clusters  
iHub repository. *See* Encyclopedia volumes  
iHub services  
*See also* specific iHub service  
Information Console  
    accessing functionality 20  
Information Console Security Extension  
*See also* IPSE applications  
Information Delivery API  
*See also* IDAPI applications  
input file IDs. *See* InputFileId element  
input file names. *See* InputFileName element  
input messages  
*See also* requests  
Integration Technology. *See* iHub Integration Technology  
Interactive Crosstabs

adding toolbars. *See* Interactive Crosstabs toolbars

interfaces  
*See also* user interfaces  
iPortalSecurityAdapter class  
*See also* IPSE applications  
iServer System. *See* iHub System

## J

Java RSSE framework  
*See also* RSSE applications  
jobs  
    failing. *See* failed jobs  
    pending. *See* pending jobs  
    print operations and. *See* print jobs  
    sending notifications for. *See* notifications

## L

Lightweight Directory Access Protocol. *See* LDAP servers  
links (Information Console)  
*See also* hyperlinks  
Linux servers  
*See also* UNIX systems  
log files  
    tracking error information and. *See* error log files  
    tracking usage information and. *See* usage log files  
Login operations  
*See also* SystemLogin operations

## M

mail. *See* e-mail  
MDS. *See* Message Distribution service  
messages. *See* e-mail  
metadata schemas. *See* schemas  
methods  
*See also* functions  
Microsoft NET environments. *See* .NET environments  
Microsoft Windows. *See* Windows systems  
monitoring tools  
*See also* performance monitoring  
multilingual reports. *See* locales

## N

names

*See also* user names

naming restrictions. *See* case sensitivity

nodes. *See* cluster nodes

non-native reports. *See* third-party reports

notifications

sending attachments with. *See* attachments

## O

object IDs

*See also* ObjectId element

on-demand report generation. *See*

synchronous jobs

online analytical processing servers. *See*

OLAP servers

online help

*See also* help

operations

administration. *See* Administrate

operations

archiving files and. *See* archiving

operations

login. *See* Login operations

searching. *See* search operations

updating files and. *See* update operations

output

formatting. *See* output formats

output messages

*See also* responses

## P

page-level security

*See also* page security application

parameter files

*See also* data object values files; report

object value files

parameter values files. *See* data object values

files; report object value files

parameters

*See also* report parameters

defining dynamic filters. *See* dynamic filter

parameters

permissions. *See* privileges

pick lists. *See* selection lists

plug-in extensions. *See* extensions

PMD. *See* Process Management Daemon

PPT formats. *See* PowerPoint formats

preferences (users). *See* user preferences

print jobs

*See also* printing

print requests. *See* print jobs

printer settings. *See* printer options

printing requests. *See* print jobs

purging. *See* deleting

## R

RCP Report Designer package

*See also* BIRT RCP Report Designer

records

*See also* rows

removing. *See* deleting

report components. *See* components

report design engine classes

*See also* Design Engine API

report documents

*See also* reports

Report Encyclopedia. *See* Encyclopedia

volumes

report execution requests. *See* ExecuteReport

operations

report explorer. *See* ReportExplorer

components

report files

*See also* files; specific report file type

naming. *See* file names

report objects. *See* reports

report parameter files

*See also* data object values files; report

object value files

report parameters

*See also* parameters

restricting values for. *See* cascading

parameters

Report Server Security Extension

*See also* RSSE applications; RSSE API

report servers. *See* iHub servers

reporting system. *See* iHub System

reports

sending as attachments. *See* attachments

repositories

*See also* Encyclopedia volumes  
requests  
*See also* SOAP messages  
sending 20  
responses  
*See also* SOAP messages  
result sets  
*See also* queries; search results  
.rov files. *See* report object value files  
RPCs. *See* remote procedure calls  
rptdesign format  
*See also* report design files  
rptdocument format  
*See also* report document files  
RSSE applications  
*See also* Report Server Security Extension  
registering external users for. *See* external  
user registration  
rules. *See* archiving rules  
run requests. *See* report generation requests

## S

scheduled jobs  
*See also* jobs  
schemas  
*See also* WSDL schemas  
search criteria. *See* search conditions  
search operations  
*See also* searching  
setting conditions for. *See* search  
conditions  
searching  
*See also* search operations  
security credentials. *See* credentials  
security roles. *See* roles  
sending requests 20  
servers  
*See also* iHub servers  
services  
*See also* iHub services; web services  
settings. *See* properties  
Simple Object Access Protocol. *See* SOAP  
SOAP endpoints  
*See also* SOAP ports  
SOAP requests. *See* requests  
SOAP responses. *See* responses

Software Development Kit  
*See also* SDK package  
sort fields. *See* sort columns  
spreadsheets. *See* Excel spreadsheets  
SQL statements. *See* queries  
subdirectories. *See* subfolders  
system administrators  
*See also* administrators  
system schemas. *See* schemas

## T

tab-separated values files. *See* TSV files  
tcpmon utility. *See* TCPMonitor  
temporary files. *See* transient files  
temporary reports. *See* transient reports  
text strings. *See* strings  
transactions  
*See also* Transaction operations  
types. *See* data types

## U

UI elements  
*See also* user interfaces  
Uniform Resource Locators. *See* URLs  
universal hyperlinks. *See* hyperlinks  
Universal Resource Identifiers. *See* URIs  
URIs  
submitting requests and 20  
user groups. *See* groups  
user IDs  
*See also* UserId element

## V

values  
*See also* data  
version names  
*See also* VersionName element  
view parameters  
*See also* ViewParameter element  
viewing parameters. *See* view parameters  
viewing preferences. *See* viewer preferences  
viewing service. *See* View service  
Vista computers. *See* Windows systems  
volume administrators. *See* administrators  
volume schemas. *See* schemas  
volumes. *See* Encyclopedia volumes

## W

web applications

*See also* applications

web service applications

*See also* IDAPI applications

Web Service Description Language. *See*  
WSDL

web services messaging framework. *See*  
SOAP

WSDL documents

*See also* WSDL files

WSDL elements

*See also* XML elements

WSDL2Java package

*See also* code emitter

## X

XML attributes. *See* attributes

XML code

*See also* code

XML reports. *See* XML documents

XML schemas

*See also* WSDL schemas

XP computers. *See* Windows systems