Building Web Applications Using BIRT APIs

Contains information proprietary to:
Actuate Corporation, 951 Mariners Island Boulevard, San Mateo, CA 94404

www.opentext.com
www.actuate.com

Document No. 141215-2-431302 June 23, 2015

Contents

About *Building Web Applications Using BIRT APIs*

Building Web Applications Using BIRT APIs provides information about using the Actuate REST API in web browser-based applications.

- *About Building Web Applications Using BIRT APIs.* This chapter provides an overview of this guide.
- *Chapter 1. Introducing BIRT APIs for applications.* This chapter introduces web applications and introduces the BIRT Aviatio web application.
- *Chapter 2. Building web content with BIRT.* This chapter introduces the tools used to build the BIRT Aviatio sample application.
- *Chapter 3. Integrating REST API.* This chapter discusses methods to integrate the REST API in a web application.
- *Chapter 4. Integrating JavaScript API.* This chapter discusses methods to integrate the JavaScript API in a web application.
- *Chapter 5. Extending web functionality.* This chapter discusses possible customization of the BIRT Aviatio application.
- *Chapter 6. Using developer resources.* This chapter lists resources to learn more about REST API.

Chapter

1

Introducing BIRT APIs for applications

This chapter contains the following topics:

- Using BIRT APIs in applications
- Introducing the BIRT Aviatio example application
- Accessing source code and resources
- About application project files
- About resources used by the example application
- About AngularJS frameworks

Using BIRT APIs in applications

This chapter discusses how to incorporate BIRT data objects and reports into your web application. The BIRT files contain your data, report templates, and visualizations. The BIRT iHub server 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 reports in PDF and Excel formats
- Downloading and filtering data in JSON or CSV format

An application can extract data from iHub using the REST API and use the values to navigate content or send the values to a third-party data visualization, such as a chart or map.

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

BIRT visualizations, such as a chart, table or a multi- page BIRT report display in interactive web pages using JSAPI.

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 Aviatio example application

This example illustrates how to integrate BIRT iHub resources into a web application using HTML and JavaScript. This application is designed for a minimum screen size of 1024 x 768px.

The Actuate REST API authenticates the application and retrieves data used to build navigation links and information about files on the demonstration BIRT iHub 3.1 server.

The Actuate JavaScript API (JSAPI) retrieves data and visualizations from a 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
- Searching for the file ID of a BIRT data object
- Downloading a list of location names from the BIRT data object
- Building an interactive list of location names
- Executing a BIRT design to update data and create a BIRT document using the selected state as a parameter value
- Searching for the file ID of the generated BIRT report document
- Retrieving meta-data about the BIRT document
- Loading BIRT visualizations into different DIV elements of the web page

When the web application first loads, it authenticates with the iHub server and retrieves a JSON formatted list of region and state names.

After using REST API to extract location names from a BIRT data object, this application builds navigation links. When a user selects a link, the REST API generates a report for the selected location if a valid one does not already exist. The Actuate JSAPI then displays items from the report in the DIV elements of the HTML page.

The index.html file contains the following HTML DIV elements:

- startPage, containing the landing page for the application
- content, containing a menu and report content
  - leftSidebarDiv, containing the menu of location names
  - mainDiv, containing the loading animation and user selected visualizations
    - overlay, containing loading message while data is downloads
    - container, containing navigation buttons and report visualizations

The container DIV contains the following DIV elements:

- appTitle, containing the name of the location
- nextButton and backButton to navigate the list of locations
- busiest_worst, containing the bookmarked grid named busiest_worst

- percentage, containing the bookmarked chart named percentage
- average, containing the bookmarked chart named average
- causes, containing the bookmarked grid named causes, displaying pie charts
- timestamp, containing the time and date the report was generated

Figure 1-1 shows the application's interface.

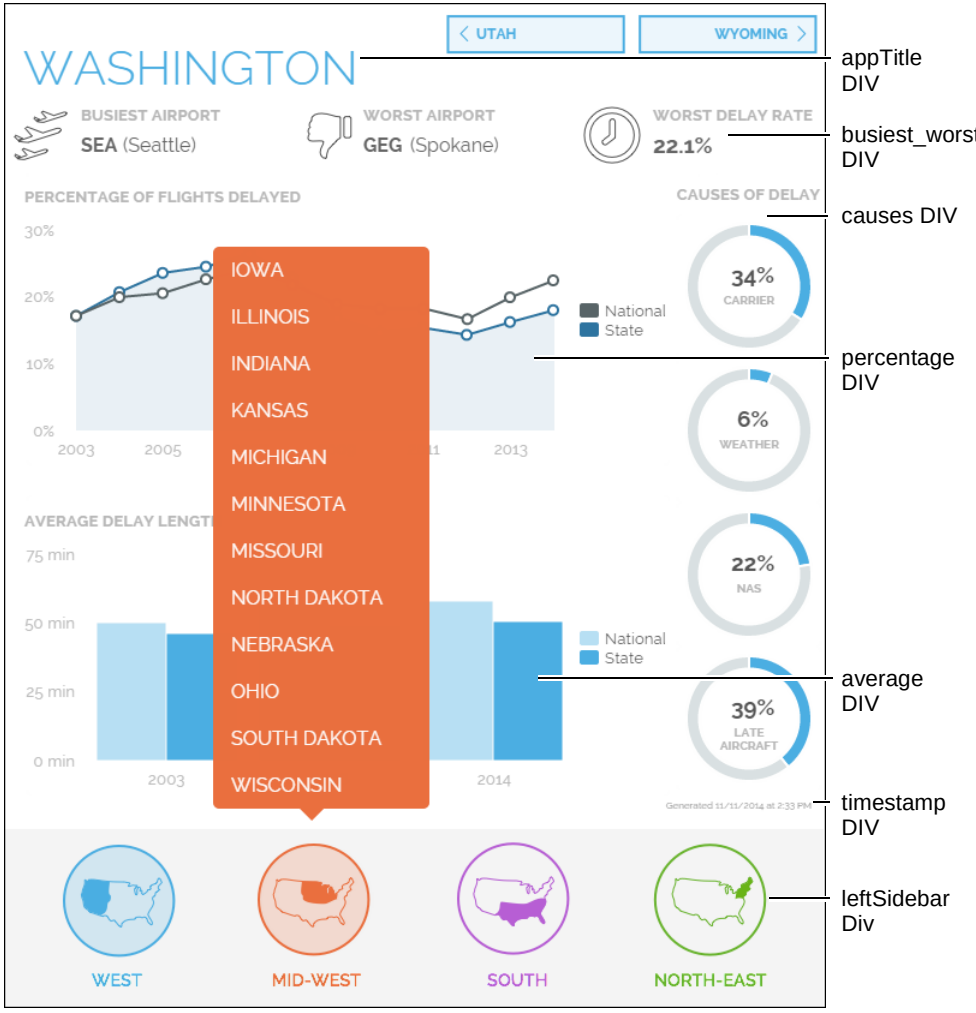

Figure 1-1 User interface displaying a list, charts, and navigation buttons

This example writes data to the JavaScript console of the web browser for debugging purposes. When the application authenticates with an iHub server the word success appears in the JavaScript console.

When the user selects a location, the region name, state abbreviation, and generated report are displayed in the JavaScript console. The following console output shows the result of a user selecting the state of Washington:

```
region = West
state = WA
filename = /Home/West/WA.rptdocument
Cleared div mask
```

While a new report is generated, a temporary DIV appears in the application notifying the user that data is loading. After the report generation finishes, the report items download to the web application and the loading DIV is cleared.

Accessing source code and resources

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

`https://github.com/ActuateBIRT/AviatioExample`

BIRT Aviatio uses the following third-party chart libraries:

- HTTPS://angularjs.org/
- HTTP://fonts.googleapis.com
- HTTP://jquery.com

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

About application project files

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

- index.html, containing the HTML to render the application in a web browser
- /css folder containing the style.css file for the web application
- /images folder containing the application image files and icons
- /js folder containing the JavaScript files used in the web application
  - angular-loader.min.js, AngularJS loader for Angular modules
  - angular-resource.min.js, AngularJS interaction support with RESTful services via the $resource service
  - angular.min.js, minimized version of AngularJS frameworks

- controller.js, contains JavaScript functions such as login, initViewer, loadBookmark, executeReport, nextState, previousState, selectState
- factory.js, generates requests to each RESTful service the application uses
- main.js, generates REST URIs and sets default values for the application
- services.js, template for RESTful HTTP requests

About resources used by the example application

The example, BIRT Aviatio, retrieves data from a BIRT data store an visualizations from BIRT report document files installed at the iHub server. This application uses the following server-side files to generate all BIRT content displayed in the web page:

- flight delay.data
- Flight Performance.RPTDESIGN

These resource files are included with the BIRT Aviatio Reports folder in the source code, stored on GitHub. 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 flight delay.data file in the \Resources\Data Objects folder of the iHub volume.

The report design is stored in the home folder of the user account that the application uses to log in. The user name is set in the main.js JavaScript file, and uses a default value of flightdemo. Install the Flight Performance.rptdesign file into the \Home\<username> folder of the iHub volume, where <username> is the account used to log in to the iHub server.

The report design uses parameters to filter data displayed in the report. The reports items are then shown in DIV tags of the web application when a user selects a state name or navigates to the next or previous state report.

When a State name is selected, REST API checks if the report exists and if not generates an the report for the selected state.

The generated report document file includes bookmarked charts that are displayed in the application using Actuate JSAPI.

About AngularJS frameworks

This web application uses the AngularJS JavaScript framework to build navigation links in the web page, organize JavaScript functions and maintain a

responsive web application. AngularJS is a client-side JavaScript framework for building interactive web applications.

Chapter

2

Building web content with BIRT

This chapter contains the following topics:

- Overview of BIRT iHub Information Console
- Considering which Actuate API to use
- Using BIRT Designer Professional for web content
- Introducing GitHub

Overview of BIRT iHub Information Console

BIRT iHub Information Console is a browser-based solution for document delivery, data analysis and building reports. Information Console 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.

Information Console 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 Information Console 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 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 API libraries extend functionality in applications that provide API integration points. The APIs used for client-side application development include:

- 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 visualizations and reports 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 using 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; providing access to essential functions and raw data. Actuate offers many APIs that provide additional functionality but they are implemented using specific tools or access resources using different formats and interfaces.

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

Web 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.

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. The iHub server receives the JSAPI requests and sends HTML content for display in a selected HTML DIV element.

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.

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

Using BIRT Designer Professional for web content

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 build 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 design 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 group data and 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.

BIRT data objects also offer the following features:

- Securing data using Actuate Page and Row Level Security, and Java code execution security
- Supporting data sources such as Amazon, Cassandra, Cloudera, Hbase, Hive, JDBC, MongoDB, POJO, and web services
- Easy scripting of data queries
- Fast analytic queries using columnar data stores
- Improving data retrieval performance using caching and in-memory data models
- Optimizing direct data access using data trimming and push down

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 Aviatio is available at GitHub. You do not need an account with GitHub to download the source code for BIRT Aviatio, 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/`

Chapter

3

Integrating REST API

This chapter contains the following topics:

- Reviewing REST API integration
- Building URIs to access REST API
- Authenticating with REST API
- Displaying a list with REST API
- Searching for files with REST API
- Running reports 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 JavaScript 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 that are used to build BIRT reports.
- Download visualizations for display in HTML DIV tags.

The application sends the RESTful URI request to the iHub server using services.js. The metadata collected using the REST API includes the timestamp when the BIRT document was created. JavaScript uses this value to check if the data is older than the report_refresh_Time value, located in main.js. If the data is older the report is executed to refresh the data.

The following REST API operations are used in BIRT Aviatio:

- /Login to return an authId for an authenticated user.
- /Files to retrieve a file id by searching for the file name.
- /Visuals to execute a BIRT design file.
- /Dataobject to extract values from a data set in a selected data store file.

The main JavaScript functions used in the application are organized into the following files:

- controller.js contains JavaScript functions to log in, load visualizations, and run reports.
- factory.js generates requests to each RESTful service the application uses.
- main.js library builds RESTful URIs to access resources on the iHub server.
- services.js is a template for RESTful HTTP requests.

Building URIs to access REST API

The BIRT Aviatio application builds the URIs to access the iHub server's REST API in the main.js file. The following code show the different URI's used by the application:

```
/* Controllers */
```

```
var mainCtrl = angular.module('mainApp', []);
var restHost = "aviatioexample.actuate.com";
var restPort = "\\:5000";
var visuals = "visuals";
var ihubHost = "aviatioexample.actuate.com";
var ihubPort = ":8700";
mainCtrl.constant('mainAppCtrl', {
  login: 'http://' + restHost + restPort + '/ihub/v1/login',
  folders: 'http://' + restHost + restPort + '/ihub/v1/folders',
  files: 'http://' + restHost + restPort + '/ihub/v1/files/
   :fileId',
  downloadReport: 'http://' + restHost + restPort + '/ihub/v1/' +
   'files',
  reports: 'http://' + restHost + restPort + '/ihub/v1/' + visuals
   + '/:reportsId/:outputFormat',
  reportData: 'http://' + restHost + restPort + '/ihub/v1/' +
   visuals + '/:reportsId/data/:datasetname/?format=:format',
  reportBookmarks: 'http://' + restHost +restPort + '/ihub/v1/' +
   visuals + '/:reportsId/bookmarks',
  reportMetadata: 'http://' + restHost + restPort + '/ihub/v1/' +
   visuals + '/:reportsId/datasets',
  reportBookmarksData: 'http://' + restHost + restPort + '/ihub/
   v1/' + visuals + '/:reportsId/bookmarks/:bookmarkName',
  reportMetadataData: 'http://' + restHost + restPort + '/ihub/v1/
   ' + visuals + '/:reportsId/datasets/:datasetname',
  downloadFile: 'http://' + restHost + restPort + '/ihub/v1/' +
   visuals + '/:reportsId/download',
  dataObject: 'http://' + restHost + restPort + '/ihub/v1/
   dataobject/:dataObjectId',
  dataObjectElement: 'http://' + restHost + restPort + '/ihub/v1/
   dataobject/:dataObjectId/:dataobjectElement',
  executeReport : 'http://' + restHost + restPort + '/ihub/v1/' +
   visuals + '/:reportsId/execute',
  jsapiUrl: 'http://' + ihubHost  + ihubPort +  '/iportal/jsapi',
  iportalUrl : 'http://' + ihubHost  + ihubPort +  '/iportal',
  report_refresh_Time : '15',
  idle_time : '20',
  username : "flightdemo",
  password: "Demo1234"
});
```

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 as a 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 with information about the user account. A REST API authentication identifier remains valid for 24 hours by default. The URI used to login is created in the /js/main.js file.

The index.html file starts the login function when the page loads using the following code in index.html:

```
<html lang="en"  ng-app="flightApp" ng-controller="loginCtrl" ng-
   init="login()">
```

This starts the login function in /js/controller.js that uses a URI to the Actuate REST /login resource. The login function runs the following code:

```
$scope.login = function() {
var params= {
    'username': mainAppCtrl.username,
    'password': mainAppCtrl.password
  };
  API.Login.post(params, function(dataResponse) {
    $scope.response = dataResponse;
```

A successful authentication returns a response body similar to the following:

```
{
  "AuthId": "jIWy49iySIytHlaDNHSyStu6/
   KSKB2NJeGNo2RKryzupeK23GPyF9wBqBRH2+JDaKtThHpworWqsuMQTPZXR5Zam
   27DckdXTLxdDKfdJxxh6cYr75qMKUmyNJ+FKP4j3iEI2Zn04f61r0luc7tKFnZH
   oPPa9nTXxhrQ+lRiNs8t3NOcglCbGWHc+g64RaLQ0rflDawq/
   6FBfsh87w0D3Qs+raaJrTrbdiuJkrDXYq/
   GZthzB8lmRlhhlRi0MhTfSSDj2kwXqQpX5hepvtWVBDv24a+nhPjRYOQZOlRdSS
   vvhOLPgoNpZM0WmzdRj2+eqFTdDvj+lIcZtkO4Fh7KWaXOVRErKapokfIb/
   77X9h6de0YY63tm0OMzMkq/o5c6+",
  "User": {
    "Id": "200100000100",
    "Name": "flightdemo",
    "EmailAddress": "flightdemo@flightdemo.com",
    "HomeFolder": "/Home/flightdemo"
  }
}
```

Next, the login function sends a request to the /files resource to search for the file id of flight delay.data using the following code:

```
if (dataResponse.AuthId) {
      $rootScope.userData = dataResponse;
      $rootScope.AuthId = dataResponse.AuthId;
      params = {
        search :"/Resources/Data Objects/flight delay.DATA",
        authId: $rootScope.AuthId
      };
API.DownloadReport.query(params, function (dataResponse) {
```

A successful file search returns a response body similar to the following:

```
{
  "ItemList": {
    "File": [
      {
        "Id": "410300000100",
        "Name": "flight delay.data",
        "FileType": "DATA",
        "PageCount": "0",
        "Size": "10647543",
        "Version": "1"
      }
    ]
  },
  "TotalCount": "1"
}
```

Then the login function sends a request to the /dataobject resource with the file id of flight delay.data. This request searches for the geographic data set in the flight delay.data file using the following code:

```
if (dataResponse.TotalCount > 0) {
    params = {
      dataObjectId : dataResponse.ItemList.File[0].Id,
      dataobjectElement : "geographic",
      authId: $rootScope.AuthId
    };
API.DownloadDataObjectElement.query(params, function
   (dataResponse) {
```

A successful data set search returns a response body similar to the following:

```
{
"data": [
  {
    "state": "AL",
    "full_state": "Alabama",
    "region": "South",
    "scale": "5"
  },
  {
    "state": "TN",
    "full_state": "Tennessee",
    "region": "South",
    "scale": "5"
  },
  {
    "state": "AR",
    "full_state": "Arkansas",
    "region": "South",
    "scale": "4"
  },
```

The login function finishes the login process by storing the state names in an array for the region associated with each state.

See the source code for the complete example.

Displaying a list with REST API

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

- Search for the flight delay.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 data set of locations from the flight delay.data file.
- Parse the JSON response from the iHub server into an array for each region.
- Build the state name navigation of the application using these names.

The index.html file displays region names for a user to select using the following HTML:

```
<div id="startNav">
  <div id="startWest" ng-click="button_clicked ||
   showState('West', 'start')"></div>
  <div id="startMidwest" ng-click="button_clicked ||
   showState('Midwest', 'start')"></div>
  <div id="startSouth" ng-click="button_clicked ||
   showState('South', 'start')"></div>
  <div id="startNortheast" ng-click="button_clicked ||
   showState('Northeast', 'start')"></div>
</div>
```

When the user selects a region, the showState function displays the states in the selected region using the following JavaScript code from controller.js:

```
$scope.showState = function(region, page) {
    $scope.statePopOverClass = "hide";
    if ($scope.selectedRegion != region) {
        $scope.highlightRegion(region, page);
        $scope.selectedRegion = region;
        $scope.statePopOverClass = region + "Class";
    } else {
        $scope.highlightRegion($scope.currentRegion, page);
        $scope.selectedRegion = "";
    }
}
```

When the user selects a state, the selectState function runs. This function populates the previous and next navigation buttons with the correct state to load, initializes the JSAPI BIRT viewer if it is not already initialized, runs the reportNeedsReRun function to prepare for loading report visualizations.

See the source code for the complete example.

Searching for files with REST API

The reportNeedsReRun function searches to see if the report for the selected state already exists using the following code:

```
var fileName = "/Home/" + region + "/" + state + ".rptdocument";
console.log("region " + "= " +  region);
console.log("state " + "= " +  state);
params = {
    search : fileName,
    authId: $rootScope.AuthId
};
API.DownloadReport.query(params,
     function (dataResponse) {
        if (dataResponse.TotalCount > 0) {
```

A successful data set search returns a response body similar to the following:

```
{
  "ItemList": {
    "File": [
      {
        "Id": "320610000100",
        "Name": "WA.rptdocument",
        "FileType": "RPTDOCUMENT",
        "PageCount": "1",
        "Size": "2369202",
        "Version": "1"
      }
    ]
  },
  "TotalCount": "1"
}
```

If the file exists, it is queried for meta-data about the file with the following code:

```
if (dataResponse.TotalCount > 0) {
  params = {
    fileId :
   dataResponse.ItemList.File[dataResponse.ItemList.File.length -
   1].Id,
    authId: $rootScope.AuthId
  };
  API.Files.query(params,
    function (dataResponse) {
      var ts = dataResponse.File.TimeStamp;
      $scope.timestamp = "Generated " + $filter('date')(new
   Date(ts), "MM/dd/yyyy 'at' h:mma");
```

A successful data set request returns a response body similar to the following:

```
{
  "File": {
    "Id": "320610000100",
    "Name": "/Home/West/WA.rptdocument",
    "FileType": "RPTDOCUMENT",
    "PageCount": "1",
    "Size": "2369202",
    "TimeStamp": "2014-11-11T15:55:24.000Z",
    "Version": "1",
    "Owner": "flightdemo"
  },
  "ACL": {},
  "ArchiveRules": {}
}
```

If the report time stamp validates within the report refresh time, then the visualizations of the report load into the application. If the report is older than the report refresh time, it is run again. The following code shows this validation check:

```
if((new Date().getTime() - new Date(ts).getTime()) >
   mainAppCtrl.report_refresh_Time * 60 * 1000 ) {
        $scope.executeReport(fileName,region, state);
   } else {
      $scope.loadBookmarks(fileName);
   }
},
```

See the source code for the complete example.

Running reports with REST API

When a user makes a selection from a list of locations, the application verifies if the selected report already exists and if it is recent. If the report is older than the report refresh time or if the report does not exist, the report is generated with the executeReport function.

The following overview describes this functionality in the example application:

- Check if the report design file id exists.
- If the file id is not known, a search is made for the file id of Flight Performance.rptdesign; the design file that generates the BIRT report visualizations.
- Define parameters to run the report design and generate the report for the selected state.

- Run the loadBookmarks function to display the report visualizations on the web page.

The executeReport function searches for the BIRT report design file id if the id does not already exist with the following code:

```
$scope.executeReport = function(fileName, region, state) {
   console.log("fileName = " + fileName);
   var params = {
      search : '/Home/Flight Delay App/Flight
   Performance.rptdesign',
      authId: $rootScope.AuthId
   };
   if (!$rootScope.reportId )  {
      API.DownloadReport.query(params,
      function (dataResponse) {
      $rootScope.reportId = dataResponse.ItemList.File[0].Id;
```

A successful file search returns a response body similar to the following:

```
{
  "ItemList": {
    "File": [
      {
        "Id": "882200000100",
        "Name": "Flight Performance.rptdesign",
        "FileType": "RPTDESIGN",
        "PageCount": "0",
        "Size": "400400",
        "Version": "1"
      }
    ]
  },
  "TotalCount": "1"
}
```

Once the file id is known, the id is used to generate a report from the report design file using the following code:

```
var paramValues = {"ParameterValue" :
                      [
                         {"Name" : "Region","Value": region},
                         {"Name" : "State", "Value": state}
                      ]
                   };
params = {
    reportsId : dataResponse.ItemList.File[0].Id,
    paramValues : angular.toJson(paramValues, false),
    saveOutputFile : true,
    replaceExisting : true,
    requestedOutputFile : fileName,
    authId: $rootScope.AuthId
};
$scope.timestamp = "Generated " + $filter('date')(new Date(), "MM/
   dd/yyyy 'at' h:mm a");
API.ExecuteReports.post(params,
    function (dataResponse) {
        $scope.loadBookmarks(fileName);
    },
```

A successful job execution returns a response body similar to the following:

```
{
  "Status": "FirstPage",
  "ObjectId": "220710000100",
  "OutputFileType": "RPTDOCUMENT",
  "ConnectionHandle": "bAq+y9HJrv5y7XC/gX7MO721WvOI/
   w6jmbZwg7PYF19Cq3057d+JTeUKSSPpu8CJEdtWledo6r5Fb4yX4Ucvyt84/
   3qNdvlyqKFp41x9MxeCmRgzAYm90ztm3Lyfjqx82DvON58eYK8rSKFmLiwjk8G3
   yjwNvQ0UG0eLmAm6yieKHTpWon9M95UJcKNnCtdE6HFYH+iv3kbbC1t2L7opIIn
   DUZnxKjZ6YZgrmUYKgYPKVi0rSGjMWS1obFISqHPASoQcmOHMSkyry94aXKCn+e
   LVBZJu9VSY"
}
```

The executeReport function then extracts the bookmarked visualizations from the generated file.

See the source code for the complete example.

Chapter

4

Integrating JavaScript API

This chapter contains the following topics:

- Reviewing JSAPI integration
- Displaying BIRT content in a web page
- Loading visualizations by bookmark name

Reviewing JSAPI integration

This application uses JSAPI to display BIRT visualizations in DIV tags of a web page. The application loads JSAPI using the jQuery method getScript, located in controller.js.

JSAPI communicates with the iHub server using the authentication values used in the REST API login request. The JSAPI then extracts content from the report generated by the REST API and displays each item in the report in a different DIV tag. Bookmarks are a method to identify content in a BIRT report, such as charts, tables, or a grid including both charts and tables.

This application uses JSAPI to complete the following tasks:

- Create a BIRT viewer in different DIV elements on the web page
- Load report items for the selected state into each instance of BIRT viewer

For more information about embedding BIRT visualizations in HTML see *Integrating Applications into BIRT iHub*. More information about using JSAPI can also be found at the following URL:

```
http://developer.actuate.com/resources/documentation/ihubftype/
   integration/
```

Displaying BIRT content in a web page

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 document or part of a BIRT document into an HTML web page.

BIRT Aviatio loads the Actuate JSAPI library from the iHub server as part of the login function. The following script is used to load the Actuate JSAPI:

```
$.getScript(  mainAppCtrl.jsapiUrl )
  .done(function( script, textStatus ) {
   console.log( textStatus );
  })
  .fail(function( jqxhr, settings, exception ) {
     console.log( textStatus );
});
```

When a user selects a state name, the selectState function checks to see if a JSAPI BIRT viewer is already loaded. If there is no viewer available, the initViewer

function is called. This function uses the following Actuate JSAPI code to initialize the viewer:

```
$scope.initViewer = function (region, state) {
  if ($scope.viewerInitialized) {
    return;
  }
  var reqOps = new actuate.RequestOptions( );
  reqOps.setVolumeProfile( "default volume" );
  reqOps.setVolume( "default volume" );
  actuate.load("viewer");
  $rootScope.idleTime = new Date();
  actuate.initialize(
    mainAppCtrl.iportalUrl,
    reqOps,
    mainAppCtrl.username,
    mainAppCtrl.password,
    function () {
      $scope.viewerInitialized = true;
      $scope.reportNeedsReRun(region, state);
    }
  );
}
```

See the source code for the complete example.

Loading visualizations by bookmark name

BIRT report visualizations are loaded using their bookmark names. After locating or generating a valid report, BIRT Aviatio loads the visualizations from the report into the web page. The following loadBookmarks function is used to load the visualizations into the DIV tags of the application:

```
$scope.loadBookmarks = function (fileName) {
  $scope.loadBookmark(fileName, "percentage", "percentage", 575,
   245);
  $scope.loadBookmark(fileName, "average", "average", 575, 250);
  $scope.loadBookmark(fileName, "causes", "causes", 130, 515);
  $scope.loadBookmark(fileName, "busiest_worst", "busiest_worst",
   732, 70);
```

The loadBookmark function uses Actuate JSAPI to load each visualization into the correct DIV tag of the HTML page. The following code loads each bookmark from the selected state report:

```
$scope.loadBookmark = function (fileName, bookmark, divId, width,
   height) {
  var viewer = new actuate.Viewer( divId);
  viewer.setReportName( fileName );
  viewer.setSize(width, height);
  var options = new actuate.viewer.UIOptions( );
  options.enableToolBar(false);
  options.enableToolbarContextMenu(false);
  viewer.setUIOptions( options );
  viewer.setReportletBookmark(bookmark);

   viewer.registerEventHandler(actuate.viewer.impl.EventConstants.
   ON_EXCEPTION,
     function() {
       console.log(divId + " render error");
       $scope.done.push(divId);
     }
  );
```

See the source code for the complete example.

Chapter

5

Extending web functionality

This chapter contains the following topics:

- Optimizing BIRT content for web viewing
- Using external authentication
- Changing application default values
- Additional optimizations

Optimizing BIRT content for web viewing

The report documents displayed in BIRT Aviatio use a single report design template. The page layout of the report is not used. Instead, each report item is loaded into a different HTML DIV in the report output. The JavaScript framework Angular JS rearranges these DIV elements to best fit the current orientation of the application.

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.

Using external authentication

The BIRT Aviatio 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 authentication to an iHub server are set in the main.js file of the project. You can build the application for use with your own iHub server by changing the values in this file. You can also change the file path location of BIRT resources used in the application in this file.

The following values are contained in this file:

- ihubHost , you can change this value using the following URL format:

  `aviatioexample.actuate.com`

- ihubPort, you can change the port to access the actuate JSAPI library:

  `:8700`

- restPort, you can change the port to access the REST API endpoints:

  `\\:5000`

- username, the user name to log in to the iHub server:

  `flightdemo`

- password, the password to log in to the iHub server:

  `Demo1234`

- report_refresh_Time, number of minutes before requesting new data:

  ```
  15
  ```

- idle_time, number of minutes to maintain a connection to the iHub server:

  ```
  20
  ```

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 an alternative JavaScript frameworks such as:
  - Backbone
  - Ember
  - Sencha Ext JS
- Store data requests in the web browser's local storage to speed up user selection.

Chapter

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/`

A public version of this URL is available at the following URL:

`http://restapitest.actuate.com:5000/ihub/v1/ihubrestdocs/`

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

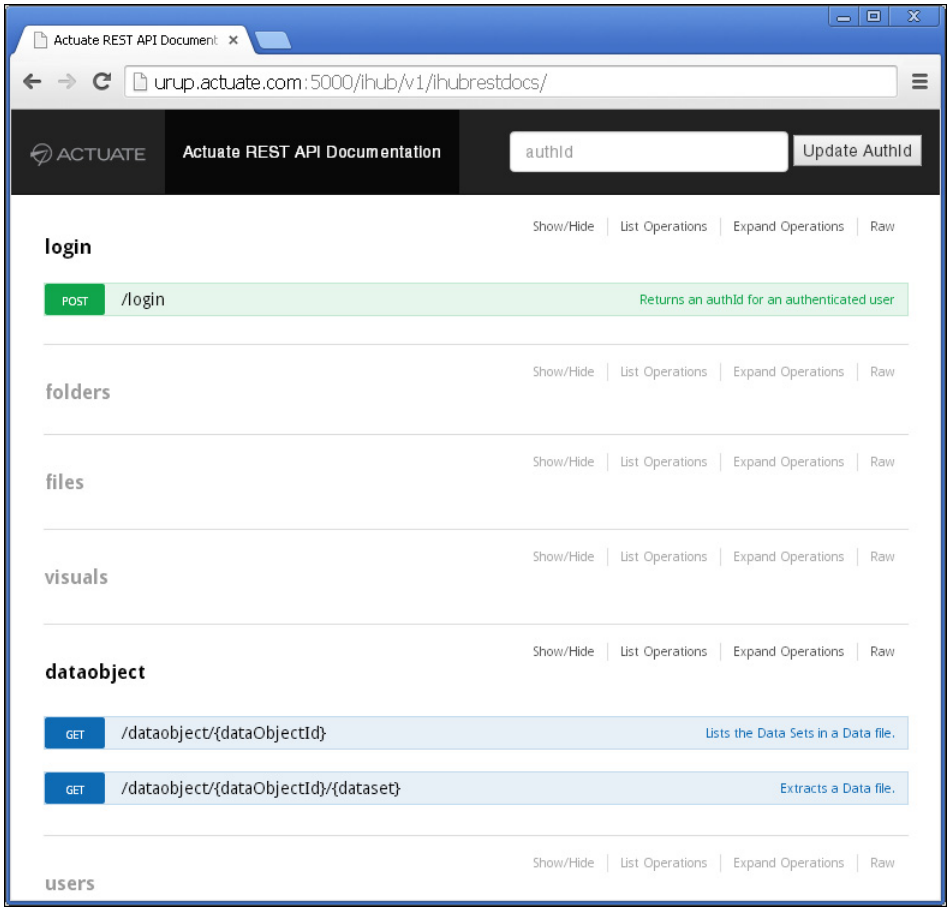

Figure 6-1 Reviewing the Actuate REST API documentation

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.

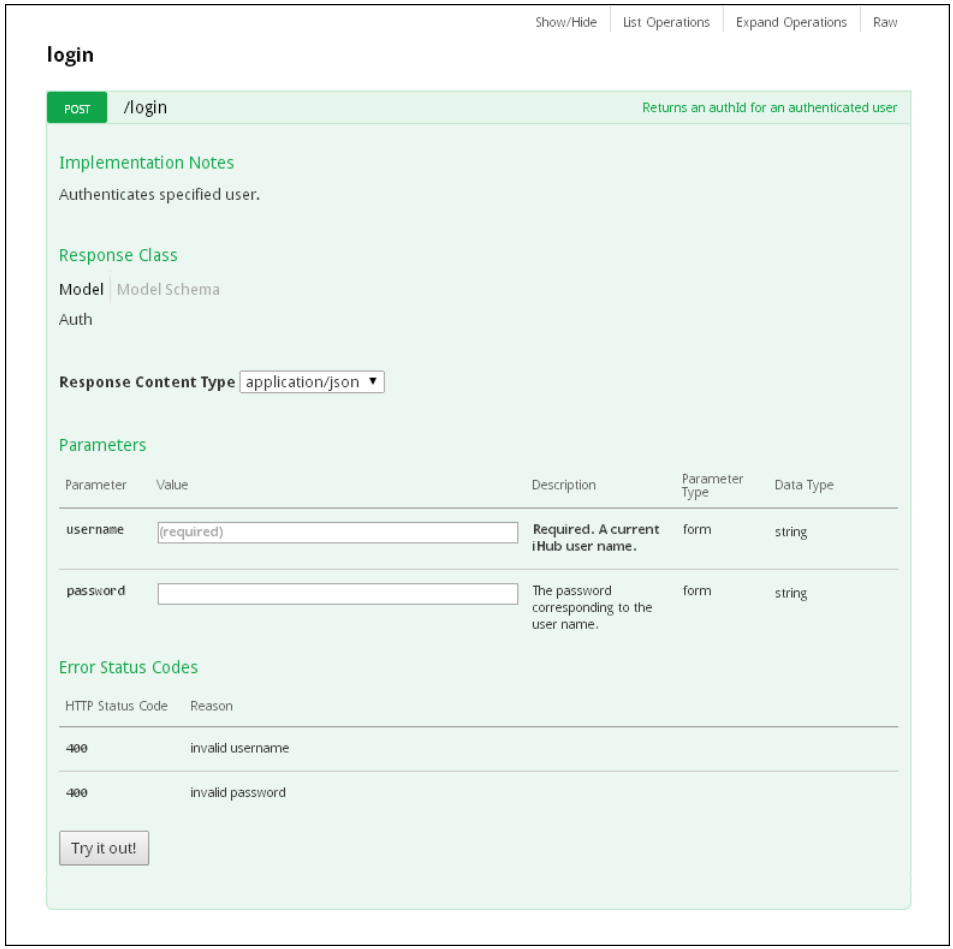

Figure 6-2 Testing the login operation

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.

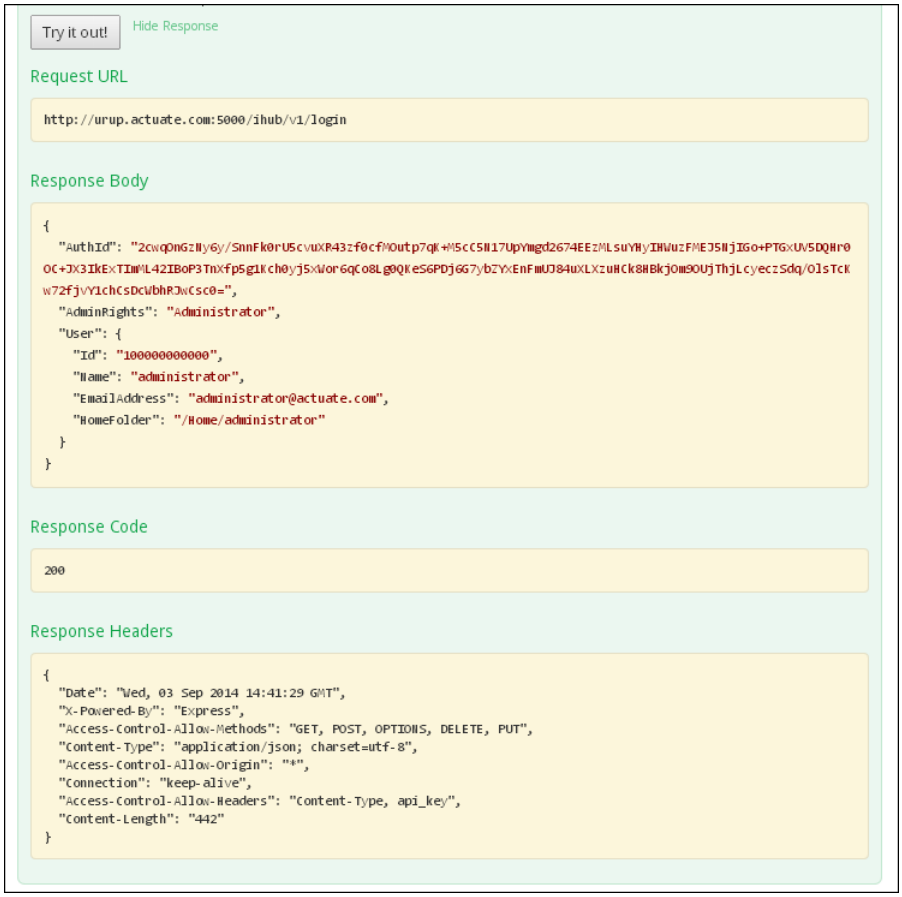

Figure 6-3 Reviewing results from the REST API

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

D

E

F

G

H

I

J

L

M

S

T

U

V

W

X