

Creating a custom web page using the Actuate JavaScript API

This chapter contains:

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

About the Actuate JavaScript API

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

To use the Actuate JavaScript API, connect to Actuate iHub Visualization Platform client or Deployment Kit for BIRT Reports.

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

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

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

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

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

Accessing the Actuate JavaScript API

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

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

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

```

<meta http-equiv="content-type" content="text/html;
charset=utf-8" />
</head>
<body>
    <div id="viewer1">
        <script type="text/javascript" language="JavaScript"
            src="http://127.0.0.1:8700/iportal/jsapi"></script>
        <script type="text/javascript" language="JavaScript">
            ... <!--functionality goes here-->
        </script>
    </div>
</body>
</html>

```

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

```

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

```

where

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

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

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

About the DOCTYPE tag

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

About UTF8 character encoding

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

Establishing an HTTP session with an Actuate web application

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

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

where

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

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

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

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

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

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

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

About Actuate JavaScript API security integration

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

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

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

Establishing a secure connection to more than one web service

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

Use `authenticate()` as shown in the following code:

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

where

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

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

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

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

Using a login servlet to connect to an Actuate web application

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

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

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

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

Using a custom servlet to connect to an Actuate web application

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

Unloading authentication information from the session

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

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

where

- `serviceurl` is a URL for the Actuate web application service to log out from. This URL must correspond to an Actuate Deployment Kit for BIRT Reports or iHub Visualization Platform client application.

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

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

Viewing reports

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

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

Load the viewer component to use the viewer on the page. Call actuate.Viewer functions to prepare a report, then call the viewer's submit function to display the report in the assigned <div> element.

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

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

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

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

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

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

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

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

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

The submit() function submits all the asynchronous operations that previous viewer functions prepare and triggers an AJAX request for the report. The

Actuate web application returns the report and the page displays the report in the assigned <div> element.

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <meta http-equiv="content-type" content="text
        /html; charset=utf-8" />
    <title>Viewer Page</title>
</head>
<body onload="init( )">
<div id="viewerpane">
    <script type="text/javascript" language="JavaScript"
        src="http://127.0.0.1:8700/iportal/jsapi"></script>

    <script type="text/javascript" language="JavaScript">
        function init( ){
            actuate.load("viewer");
            var reqOps = new actuate.RequestOptions( );
            actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,
                null, null, runReport);
        } function runReport( ) {
            var viewer = new actuate.Viewer("viewerpane");
            viewer.setReportName("/Applications/BIRT Sample App/Top 5
                Sales Performers.rptdocument");
            viewer.submit(callback);
        }
    </script>
</div>
</body>
</html>
```

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

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

```
function runReport( ) {
var viewer = new actuate.Viewer("viewerpane");
viewer.setReportName("/Applications/BIRT Sample App/Top 5 Sales
Performers.rptdocument");
viewer.setReportletBookmark("FirstTable");
viewer.submit(callback);
}
```

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

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

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

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
  charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
    src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

Report Viewer Page

3 Navigate to the following line:

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

In id, change:

sample

to:

content

4 Navigate to the empty line after the following line:

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

5 Add the following code:

```
function init( ) {  
    actuate.load("viewer");  
    var reqOps = new actuate.RequestOptions( );  
    actuate.initialize( "http://127.0.0.1:8700/iportal", reqOps,  
        null, null, runReport);  
}  
function runReport( ) {  
    var myviewer = new actuate.Viewer("content");  
    myviewer.setReportName("/Applications/BIRT Sample App/Top 5  
        Sales Performers.rptdesign");  
    myviewer.submit( );  
}
```

6 Save the file as reportviewer.html.

7 In Internet Explorer, open reportviewer.html.

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

8 A Login required dialog appears, as shown in Figure 2-5. Type administrator into the User name field and choose Sign in.

Figure 2-7 Login required dialog

If the page is blank, refresh the browser.

Controlling viewer user interface features

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

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

Set the user interface options with the enable functions in the `actuate.viewer.UIOptions` class. For example, a toolbar appears in the viewer by default, as shown in Figure 2-1.



Figure 2-1 The default toolbar for the JavaScript API viewer

To disable this toolbar, use the following code:

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

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

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

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

Accessing report content

Use the `actuate.report` subclasses to access report content that is displayed in the viewer. For example, use the `actuate.report.Table` subclass to manipulate a

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

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

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

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

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

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

Accessing HTML5 Chart features

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

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

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

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

Using a filter

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

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

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

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

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

Using a sorter

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

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

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

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

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

Using dashboards and gadgets

The `actuate.Dashboard` class loads and displays dashboards and provides access to the gadgets contained in dashboards. The `Dashboard` class requires a

pre-existing `actuate.RequestOptions` object with the repository type set loaded with `initialize`. To use the `RequestOptions` to access dashboard content residing in an encyclopedia volume, use the `RequestOptions` constructor, use `setRepositoryType()` to set the repository to encyclopedia, and provide the object as a parameter to the `initialize` call, as shown in the following code:

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

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

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

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

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

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

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

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

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

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

The `setReportName()` function accepts the path and name of a report file in the repository as the only parameter. In this example, `"/public/sharedtab.dashboard"` indicates the shared tab dashboard in the `/public` directory.

To display the dashboard, call `dashboard.submit()` as shown in the following code:

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

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

This is a complete example that displays a dashboard:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text
/html; charset=utf-8" />
  <title>Dashboard Page</title>
</head>
<body onload="init( )">
<div id="dashboardpane">
  <script type="text/javascript" language="JavaScript"
src="http://127.0.0.1:8700/iportal/jsapi"></script>

  <script type="text/javascript" language="JavaScript">
function init( ){
  actuate.load("dashboard");
  var reqOps = new actuate.RequestOptions( );
  reqOps.setRepositoryType(
    actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA);
  actuate.initialize("http://127.0.0.1:8700/iportal", reqOps,
    null, null, runDashboard );
} function runDashboard( ){
  var dash = new actuate.Dashboard("dashboardpane");
  dash.setDashboardName("/sharedtab.dashboard");
  dash.setIsDesigner(false);
  dash.submit( );
}
</script>
</div>
</body>
</html>
```

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

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
    charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

Dashboard Display Page

- 3 Navigate to the following line:

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

In id, change:

sample

to:

dashboard

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

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

5 Add the following code:

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

6 Save the file as dashboarddisplay.html.

7 In Internet Explorer, open dashboarddisplay.html.

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

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

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

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

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

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/
    html; charset=utf-8" />
  <title>Basic Viewer Example</title>
</head>
<body onload="init( )">
<table>
<tr><td class="jsapi_example_label">Report File:<td
  colspan="4">
<input type="text" size="80" id="reportfile" value="/
  Applications/BIRT Sample App/Custom Dashboard.rptdesign"
  name="Report_name">
<tr><td><input type="button" class="btn" id="run" value="Run
  Viewer" onclick="run( )" disabled="true">
</table>

<div id="content">
  <script type="text/javascript" language="JavaScript"
    src="http://127.0.0.1:8700/iportal/jsapi"></script>
  <script type="text/javascript" language="JavaScript">
    var viewer;
    function init( ){
      actuate.load("viewer");
      actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        null, null, initView);
    }
    function initView( ) {
      viewer = new actuate.Viewer("content");
      document.getElementById("run").disabled = false;
    }
    function run(){
      viewer.setReportName(document.getElementById(
        "reportfile").value);
      viewer.submit();
    }
    <!-- write an event handler to catch exceptions -->
  </script>
</div>
</body>
</html>
```

For more information about implementing the JSAPI viewer, see *Viewing reports*.

- 2 Navigate to the following line:

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

Replace the line with the following code:

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

- 3 Navigate to the following line:

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

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

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

- 4 Save the file as basicexception.html.
- 5 In Internet Explorer, open basicexception.html.

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

- 6 Open basicexception.html. The errorHandler function runs because no login credentials have been provided, as shown in Figure 2-1.

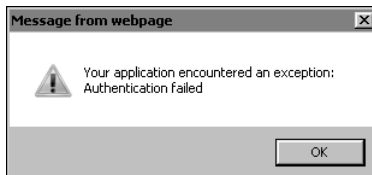


Figure 2-1 Authentication message from errorHandler

Navigating repository content using ReportExplorer

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

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

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

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

```
requestOpts = new actuate.RequestOptions( );
actuate.initialize( "http://127.0.0.1:8700/iportal", requestOpts,
    null, null, runReportExplorer);
```

Displaying ReportExplorer

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

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

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

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

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

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

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

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

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

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

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

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

The `submit()` function submits all the asynchronous operations that previous `ReportExplorer` functions prepare and triggers an AJAX request for the file information. The Actuate web application returns the list according to the results

definition and the page displays the report explorer in the assigned <div> element.

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <meta http-equiv="content-type" content="text
        /html; charset=utf-8" />
    <title>Report Explorer Page</title>
</head>
<body onload="init( )">
<div id="explorerpane">
    <script type="text/javascript" language="JavaScript"
        src="http://127.0.0.1:8700/iportal/jsapi"></script>

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

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

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

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

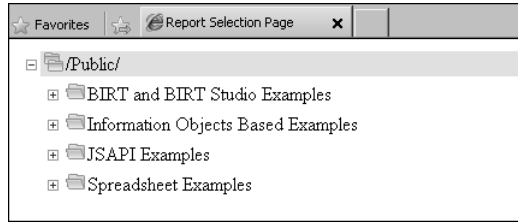


Figure 2-2 Report Explorer page

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

Opening files from ReportExplorer

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

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

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

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

```
<input type="button" style="width: 150pt;" value="View Report"
    onclick="javascript:displayReport( )"/>
...
function displayReport( ){
    var viewer = new actuate.Viewer("explorerpane");
    try {
        viewer.setReportName(file);
        viewer.submit( );
    }
```

```

    } catch (e) {
        alert("Selected file is not viewable: " + file);
        runReportExplorer( );
    }
}

```

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

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

```

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

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

var file = "unknown";

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

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

```



```

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

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

```

Tutorial 5: Displaying repository contents and opening files

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

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

Task 1: Display the report explorer

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
    charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

Report Explorer Page

- 3 Navigate to the following line:

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

In id, change:

sample

to:

explorer

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

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

5 Add the following code:

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

6 Save the file as repositorydisplay.html.

7 In Internet Explorer, open repositorydisplay.html.

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

8 If a Login required dialog appears, as shown in Figure 2-5, type administrator into the User name field and choose Sign in.

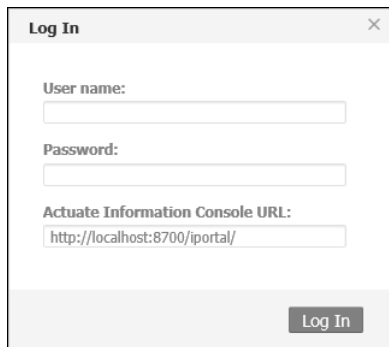
A screenshot of a 'Log In' dialog box. The dialog has a title bar with 'Log In' and a close button. It contains three text input fields: 'User name:', 'Password:', and 'Actuate Information Console URL:'. The 'Actuate Information Console URL' field has the text 'http://localhost:8700/iportal/' entered. At the bottom right is a 'Log In' button.

Figure 2-3 Login required dialog

Task 2: Track user selections

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

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

2 Navigate to the following line:

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

In title, change:

```
JSAPI Template
```

to:

```
Report Selection Page
```

3 Navigate to the following line:

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

In id, change:

```
sample
```

to:

```
explorer
```

4 Navigate to the empty line after the following line:

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

5 Add the following code:

```
function init( ){
    actuate.load("reportexplorer");
    requestOpts = new actuate.RequestOptions();
    actuate.initialize( "http://127.0.0.1:8700/iportal",
        requestOpts, null, null, runReportExplorer);
}
function runReportExplorer( ) {
    var explorer = new actuate.ReportExplorer("explorer");
    explorer.registerEventHandler(

        actuate.reportexplorer.EventConstants.ON_SELECTION_CHANGED,
        selectionChanged );
    explorer.setFolderName( "/Applications" );
    var resultDef = "Name|FileType|Version|VersionName|
    Description";
    explorer.setResultDef( resultDef.split("|") );
    explorer.submit( );
}
function selectionChanged( selectedItem, pathName ){
    file = pathName;
    alert ("File selected: " + file);
}
```

6 Save the file as reportselection.html.

7 In Internet Explorer, open reportselection.html.

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

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

- 8 On the report explorer, select a file. An alert for the file selected appears, as shown in Figure 2-4.

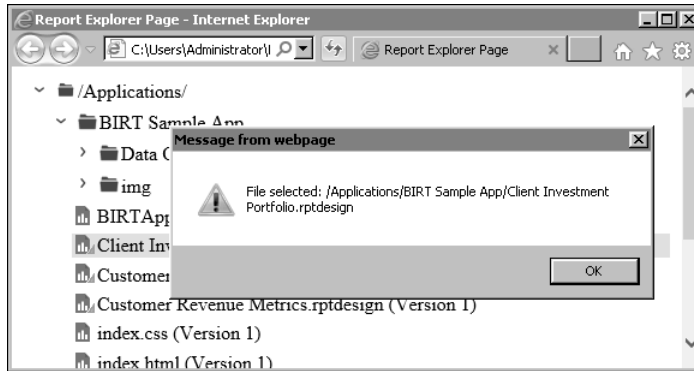


Figure 2-4 File selected alert

Task 3: Open a selected file

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

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

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

In title, change:

Report Selection Page

to:

View Selection Page

- 3 Navigate to the line after the following line:

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

Add the following code:

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

- 4** Navigate to and delete following line:

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

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

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

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

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

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

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

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

Using and submitting report parameters

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

The actuate.Parameter class handles parameters and parameter values. The actuate.Parameter class enables a web page to display and gather parameters from users before processing and downloading a report to the client. Load the actuate.Parameter class with actuate.load(), as shown in the following code:

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

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

Using a parameter component

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

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

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

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

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

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

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

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

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

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

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

```
<div id="param1">
  <script type="text/javascript" language="JavaScript"
    src="http://127.0.0.1:8700/iportal/jsapi"></script>

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

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

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

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

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

Accessing parameter values from the viewer

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

move the parameter name and value array into the viewer with
`actuate.Viewer.setParameterValues()`:

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

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

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

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

```

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

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

```

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

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

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

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
    charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

Report Parameters Page

- 3 Navigate to the following line:

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

In id, change:

sample

to:

parameters

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

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

5 Add the following code:

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

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

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

6 Save the file as parameterviewer.html.

7 In Internet Explorer, open parameterviewer.html.

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

8 Choose Run Report to view the report.

Tutorial 7: Changing parameter values and definitions

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

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

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

Task 1: Create a web page that processes parameters

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
  www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
    charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

```
JSAPI Template
```

to:

Viewer with Parameters Page

3 Navigate to the following line:

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

In id, change:

sample

to:

parampane

4 Navigate to the empty line after the following line:

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

5 Add the following code:

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

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

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

6 Save the file as processparameters.htm.

7 In Internet Explorer, open processparameters.htm.

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

8 On Please specify a territory, choose Japan, then choose Run Report. The Sales by Territory report for Japan appears, as shown in Figure 2-5.

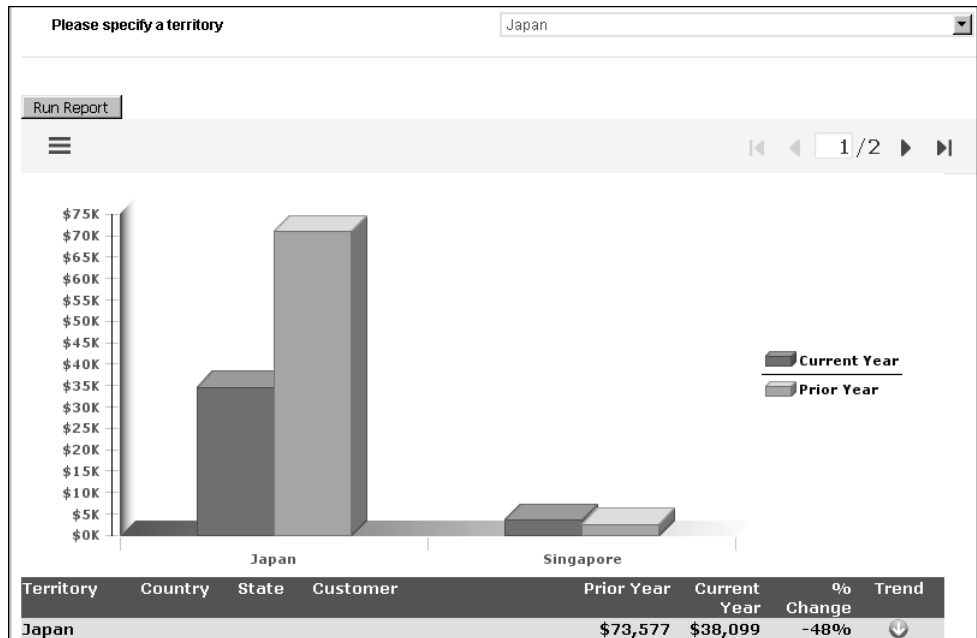


Figure 2-5 Sales by Territory for Japan

Task 2: Display a hidden parameter using ParameterDefinition

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

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

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

Replace the line with the following two lines:

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


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

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

- 4 Save the file as `displayparameters.htm`.
- 5 In Internet Explorer, open `displayparameters.htm`.

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

- 6 The `currentYear` parameter for `Sales by Territory.rptdesign` appears, as shown in Figure 2-6.



currentYear	****
Please specify a territory	NA
<input type="button" value="Run Report"/>	

Figure 2-6 `currentYear` parameter displayed on a web page

Retrieving report content as data

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

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

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

Using a data service component

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

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

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

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

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

where

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

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

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

where

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

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

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

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

where `myResultSet` is the `ResultSet` object retrieved by `downloadResultSet`.

Using a result set component

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

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

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

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

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

Controlling Interactive Viewer user interface features

The BIRT Interactive Viewer enables users to perform a number of custom operations on a BIRT design or document and save or print changes as a new design or document. The file and print features for Interactive Viewer are available in the main menu of the BIRT viewer, as shown in Figure 2-7.

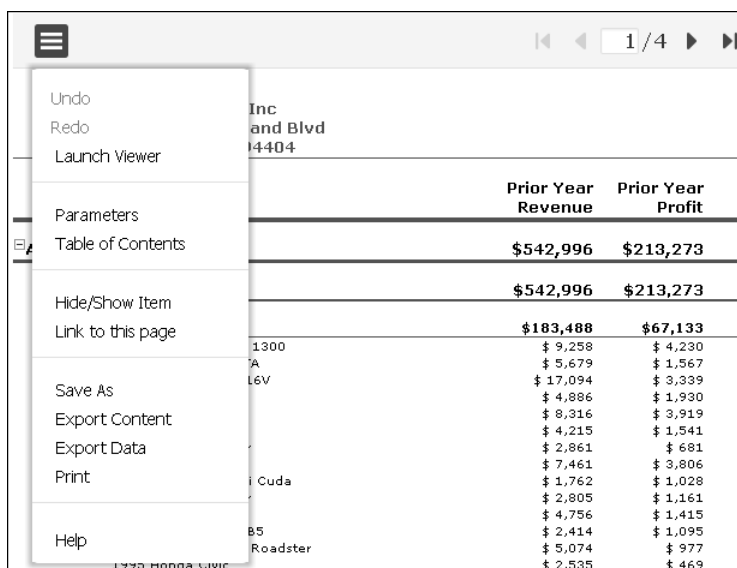


Figure 2-7 Interactive Viewer menu

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

All `actuate.viewer.UIOptions` enable functions accept a Boolean input parameter, which if true enables an interactive feature and if false disables an interactive feature. To display the list of currently enabled and disabled features, use `actuate.viewer.UIOptions.getFeatureMap()`. Table 2-1 contains a complete list of `actuate.viewer.UIOptions` enable functions that control features.

Table 2-1 UIOptions enable feature functions

Function	Interactive feature
<code>enableAdvancedSort()</code>	Enables the advanced sort feature

Table 2-1 UIOptions enable feature functions

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

Table 2-1 UIOptions enable feature functions

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

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

Disabling UI features in a custom web page

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

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

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

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

The viewer configured with the parameter page feature disabled does not show the Parameters option in the main menu, as shown in Figure 2-8.

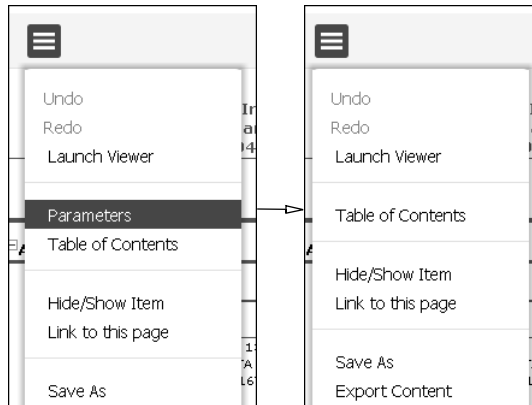


Figure 2-8 The main menu with parameters disabled

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

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

Set the user interface options with the enable functions in the `actuate.viewer.UIOptions` class. For example, a toolbar appears in the viewer by default, as shown in Figure 2-1.



Figure 2-9 The default toolbar for the JavaScript API viewer

To disable this toolbar, use the following code:

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

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

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

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

Tutorial 8: Control the BIRT Interactive Viewer user interface

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

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

Task 1: Adjust UI configuration according to browser

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

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
  charset=utf-8" />
  <title>JSAPI Template</title>
</head>
<body onload="init( )">
  <div id="sample">
    <script type="text/javascript" language="JavaScript"
    src="http://127.0.0.1:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
    <!-- Insert code here -->
    </script>
  </div>
</body>
</html>
```

- 2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

Interactive Viewer with browser-specific Controls

3 Navigate to the following line:

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

In id, change:

sample

to:

viewer

4 Navigate to the empty line after the following line:

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

5 Add the following code:

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

6 Save the file as browsercontrols.html.

7 In Internet Explorer, open browsercontrols.html.

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

Task 2: Adjust UI configuration according to user actions

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

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

2 Navigate to the following line:

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

In title, change:

```
JSAPI Template
```

to:

```
Viewer Options Page
```

3 Navigate to the empty line after the following line:

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

4 Add the following code:

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

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

5 Navigate to the following line:

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

In id, change:

```
sample
```

to:

```
contentpane
```

6 Navigate to the empty line after the following line:

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

7 Add the following code:

```
var myViewer;
function init(){
    actuate.load("viewer");
    actuate.initialize( "http://127.0.0.1:8700/iportal", null,
        "administrator", "", initView);
}
function initView(){
    myviewer = new actuate.Viewer("contentpane");
}
function runViewer(){
    myviewer.setReportName("/Applications/BIRT Sample App/
        Sales by Customer.rptdesign");
    myviewer.submit(function() {myviewer.disableIV();});
}
function runInteractive(){
    myviewer.setReportName("/Applications/BIRT Sample Apps/
        Sales by Customer.rptdesign");
    myviewer.submit(function() {myviewer.enableIV();});
}
```

8 Save the file as vieweroptions.html.

9 In Internet Explorer, open vieweroptions.html.

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

Task 3: Adjust viewer options based on a user input

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

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

2 Navigate to the following line:

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

In title, change:

JSAPI Template

to:

User Restricted Page

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

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

- 4** Add the following code:

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

- 5** Navigate to the following line:

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

In id, change:

sample

to:

contentpane

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

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

- 7** Add the following code:

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

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

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

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

- 10 For User Name, use the default value, Guest, and choose Run Viewer. The viewer loads but does not display the main menu, as shown in Figure 2-10.

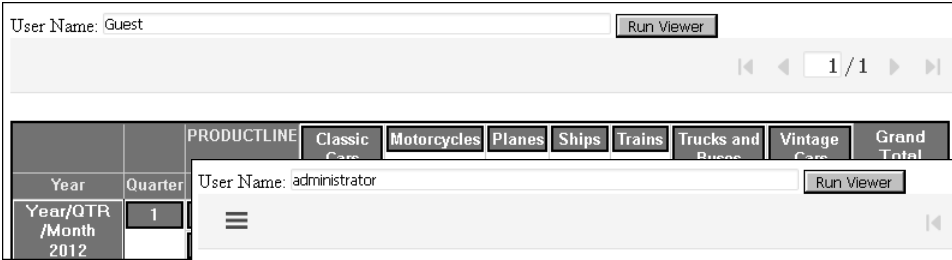


Figure 2-10

- 11 To enable the main menu in the viewer. The viewer displays the main menu, as shown in Figure 2-11.

Year	Quarter	Month	Revenue
Year/QTR /Month 2012	1	1	\$109,562
		2	\$108,232

Entering Administrator enables the main menu in the viewer

