Table of Contents

Glossary

A

attribute tree
Displays the attributes that are associated with the entity selected under the **entity tree**. For example, the Customer entity has attributes like Address, Phone, Country, etc. If no entity is selected, the attribute tree is empty.

D

drill down
Allows users to expand collapsed table rows to reveal more data. See also drill through.

drill through
Allows users to click links to reports with more detailed data. Located in the Design tab toolbar, the Drilldown button is enabled when you select a table cell or chart data point. Specify parameters in the target report to supply relevant detail data.

R

report design surface
Visible on the Design tab and the Report tab, it is a visual page designer where you can drag and drop entities and attributes to create tables and charts and design your reports.

Report Info
An insert that you can add to a textbox to show page numbering, report name, or run time. Located in the Report tab toolbox, it is enabled when you click inside a textbox.

ActiveReports 8 Server SDK

The Class Library contains documentation and code samples for the ActiveReports 8 Server API.

To learn about what's new, please see the ActiveReports 8 Server Releases page on our web site.

In This Documentation

Developer Concepts
Understand key concepts that help you to use the SDK.

How To
Learn to use the Web controls.

Samples and Walkthroughs
Learn to create and access Web services using the SDK.

Class Library
View the API documentation on the report list and designer controls.

Web Service Library
View the API documentation for the ActiveReports 8 Server ReportService.

Licensing Agreement

Please see the ActiveReports Server Licensing Agreement on our web site for full details about licensing for each edition.

Acknowledgements

Developer Concepts

These topics describe concepts that allow you to understand how to use the SDK.

Web Services
This topic explains the Web services that are installed with ActiveReports 8 Server.

Report Loading
This topic explains some of the properties that you can use to load reports into the Web Designer control.

Configuration Section Handlers
This topic explains the Web services that are installed with ActiveReports 8 Server.

URL Access to Reports and Designer
This topic explains some of the properties that you can use to load reports into the Web Designer control.

Web Services

When you install ActiveReports 8 Server, it runs an ASP.NET Web site that gives you access to the Reporting Portal and the Administrator Dashboard. The root of that Web site is located in:

```
C:\ActiveReports 8 Server\Site\
```

In the Site folder, you will find the **ReportService.svc** file. This allows you to call and access the report service.

> **Important**: If you use a virtual directory, you must copy the crossdomain.xml file (from C:\ActiveReports 8 Server\Site) to the full root of the parent application. If you have it only in the virtual directory, a Security sandbox violation occurs when you try to preview a report.

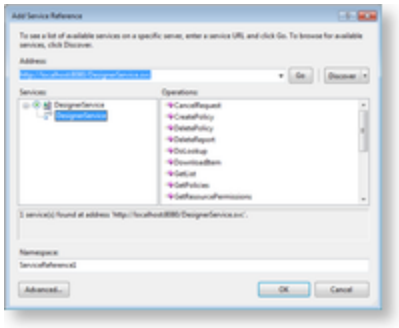
There is code in each of these Web service files that tells ASP.NET that when it encounters one of our file types, it can direct the call to ActiveReports 8 Server for handling.

To use one of the ActiveReports 8 Server Web services, you create a client of some sort, for example a **console application**, and add a Service Reference that points to the service address, by default:

```
http://localhost/ReportService.svc
```

> **Note**: If you entered a different site port, enter that value after localhost. For example, `http://localhost:8080/ReportService.svc`

In the Add Service Reference dialog, once you enter the correct address and click **Go**, you can select the service and view the available operations in the pane to the right.



ASP.NET responds differently depending on how you access the Web service URL.

- If you access it using a raw Web service client, it initiates an exchange of XML.
- If you access it with a browser it displays a Web page with information about using and testing the service.

When you add a service reference, Visual Studio calls the URL and appends ?WSDL. For example,

```
http://localhost/ReportService.svc?wsdl
```

WSDL, or Web Service Description Language, is XML that describes the methods and arguments that are available on the Web service, which is how Visual Studio generates the classes that you can access from code.

To see the Web services in action, check out the included sample located in:

```
C:\ActiveReports 8 Server\SDK\Samples
```

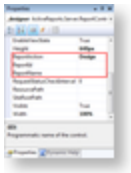
Report Loading

When you use the Designer Web control, you can load reports into it by specifying a value for one of two properties:

- **ReportName** loads the first report with the specified name.

  > **IMPORTANT**: Multiple reports can share the same name, so using the ReportName property does not guarantee that the same report is loaded every time.

- **ReportID** loads the unique report with the specified ID. When this is specified, the ReportName property is ignored.



A third property, **ReportAction**, tells the designer how to open the report. ReportAction has two enumerated values:

- **Design** is the default value for ReportAction, and tells the designer to open the report in Design view.
- **Preview** tells the designer to open the report in the Preview tab.

Configuration Section Handlers

You can use the ReportServiceProxy class to provide a handler for the ResolveRemoteEndPoint event using code in the web.config file.

Syntax

To provide a handler for the ResolveRemoteEndPoint event

ASP.NET code. Paste in the web.config file BETWEEN the <configSections> and </configSections> tags.

```
<sectionGroup name="activereports.server">
  <section name="reportServiceProxy"
type="ActiveReports.Server.ReportControls.Configuration.ActiveReportsServerSection,
  ActiveReports.Server.ReportControls, Version=x.x.xxxx.x, Culture=neutral,
PublicKeyToken=d557f2f30a260da2" allowDefinition="Everywhere" />
</sectionGroup>
```

ASP.NET code. Paste in the web.config file BELOW the </configSections> tag.

```
<activereports.server>
  <reportServiceProxy remoteReportServicePath="http://localhost:8080/" username="Admin"
password="1" />
</activereports.server>
```

Remarks

Default configuration:

- You must specify a value for the **RemoteReportServicePath** to use the section handlers.
- The **username** and **password** values are empty by default, and are optional.

Configurable Location

Use these handlers in the application-level web.config file.

> **Note**: While other locations may work, we do not support them.

URL Access to Reports and Designer

You can use a uniform resource locator (URL) request to preview or edit reports, or even design new ones from a specified model.

> **Important**: The logged-in user must have permission to perform the specified action on the report.

Syntax

http://*server***/?***param***="***string***"&ReportAction=***enum*

Arguments

Argument	Description
server	Replace this with the name or IP address of the computer on which you run the report server.
?	The question mark indicates to ActiveReports 8 Server that the rest of the URL contains parameters and an action to perform.
param	Replace this with the parameter that you want to use. Select from **ModelName**, or **ReportID**.
string	Replace this with the model name or ID, or report ID, depending on the parameter you use.
&	The ampersand indicates to ActiveReports 8 Server that the parameter is complete and the action to perform follows.
ReportAction	This is the designer property that indicates to ActiveReports 8 Server that an enumerated action value is to follow.
enum	Replace this with the enumerated action value that you want the designer to perform. Select from **Create**, **Design**, or **Preview**.

Examples

Create a New Report

To create a new report from a model that you specify without opening the Models page, replace the italicized portions of the following URL with your server name and model.

URL request syntax for creating a new report.

```
http://MyServer/?ModelName="MyModel"&ReportAction=Create
```

For the **ModelName** parameter, you can use the model name, the model ID, or the GUID. If you use the model name, and there is more than one model with the same name, ActiveReports 8 Server uses the first model of that name that it encounters.

> **Tip**: To create a quick link to a new report, from the Administrator Dashboard, open the Models list. To the right of the model you want to edit, right-click the **Create report** link and select **Copy link address**.

Edit a Report

To edit an existing report in the designer without opening the report list, replace the italicized portions of the following URL with your server name and report ID.

URL request syntax for editing a report.

```
http://MyServer/?ReportId="1"&ReportAction=Design
```

> **Tip**: To create a quick link to a report, from the Administrator Dashboard, open the Reports list. To the right of the report you want to edit, right-click the **Design** link and select **Copy link address**.

Preview a Report

To preview an existing report without opening the report list, replace the italicized portions of the following URL with

your server name and report ID.

URL request syntax for previewing a report.

```
http://MyServer/?ReportID="1"&ReportAction=Preview
```

Tip: To create a quick link to a report, from the Administrator Dashboard, open the Reports list. To the right of the report you want to preview, right-click the **Preview** link and select **Copy link address**.

How To

See the topics with the SDK samples that describe how to run the Web service code examples in your ActiveReports 8 Server projects:

Use the Designer Web Control
This topic explains how to use the Designer control on a Web Form.

Hide Log In and Log Out Buttons
This topic explains how to hide authentication UI elements when you use the Designer in your own web applications.

Open the Designer without Selecting a Model
This topic explains how to open the Designer without first visiting the list of models.

Create a Custom Security Provider
This topic explains how to create a custom security provider and use it with a model.

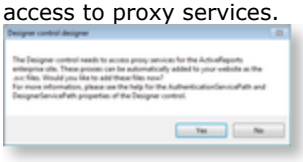
Debug a Security Provider
This topic explains how to debug a custom security provider.

Work with the HTML5 Viewer Using Javascript
This topic explains how to work with the HTML5 Viewer using the Javascript.

Use the Designer Web Control

The Web controls need to be directed to the server used for ActiveReports 8 Server in order to function. In your production application, specify the user name and password based on the current user and store information in the current session to persist these values.

To add the Designer control to a Web Form

1. In Visual Studio, create a new C# ASP.NET Web Site.
   If you need to add the control to your toolbox, drop down these steps.
   a. Right-click in the General tab and select **Choose items**.
   b. In the Choose Toolbox Items dialog that appears, the .NET Framework Components tab is selected by default. Click the **Namespace** column header to sort by namespace.
   c. The ActiveReports.Server.ReportControls namespace is at or near the top of the list. Select the checkbox next to **Designer** and click **OK**.
   d. The Designer control appears in your Visual Studio toolbox.
2. Open the Design view of your Web Form and from the toolbox, drag and drop the Designer control onto the body section of the Web Form.
3. On the message box that appears, click **Yes** to automatically add .svc files to your Web site to give the designer access to proxy services.
   
4. You can resize the designer using the Properties grid by changing the **Width** property.
5. So that the link at the top of the Designer will work for your Admin users, change the **AdminPath** property to the URL of your Admin site, for example, http://localhost:8080/Admin/.

To direct the Designer to the server used for ActiveReports 8 Server

You can specify the security token and ActiveReports 8 Server host using code in a Global Application Class.

1. From the Visual Studio **Website** menu, select **Add New Item**.
2. In the Add New Item dialog that appears, select Global Application Class and click **Add**.
3. In the Global.asax file that appears, provide a handler for the ResolveRemoteEndPoint event using code like the following in the Application Start event:
   To provide a handler for the event

   C# code. Paste INSIDE the Application_Start event.

```
ReportServiceProxy.ResolveRemoteEndpoint += ResolveRemoteEndpoint;
```

4. Below the Application Language tag at the top of the file, import the Servicing namespace so that you can use the ReportServiceProxy using a directive like the following:
To import the Servicing namespace

ASP.NET code. Paste in the Global.asax file on the line BELOW the Application Language line.

```
<%@ Import Namespace="ActiveReports.Server.ReportControls.Servicing" %>
```

5. Below the Application Start event, create the ResolveRemoteEndpoint event using code like the following, but with your address, user name, and password:
To create the ResolveRemoteEndpoint event

C# code. Paste in the Global.asax file AFTER the Application_Start event.

```
    static void ResolveRemoteEndpoint(RemoteEndpoint remoteEndpoint)
    {
        remoteEndpoint.Address = "http://localhost:8080";
        remoteEndpoint.SecurityToken = "MySecurityToken";
    }
```

You can also specify the user name, password, and ActiveReports 8 Server host in your web application's web.config file.

1. In the web.config file, provide a handler for the ResolveRemoteEndPoint event using code like the following.
To provide a handler for the event

C# code. Paste BETWEEN the <configSections> and </configSections> tags.

```
    <sectionGroup name="activereports.server">
      <section name="reportServiceProxy"
type="ActiveReports.Server.ReportControls.Configuration.ActiveReportsServerSection,
      ActiveReports.Server.ReportControls, Version=x.x.xxxx.x, Culture=neutral,
PublicKeyToken=d557f2f30a260da2" allowDefinition="Everywhere" />
    </sectionGroup>
```

2. Below the Application Start event, create the ResolveRemoteEndpoint event using code like the following, but with your address, user name, and password
To import the Servicing namespace

ASP.NET code. Paste in the web.config file BELOW the </configSections> tag.

```
  <activereports.server>
    <reportServiceProxy remoteReportServicePath="http://localhost:8080/"
username="Admin" password="1" />
  </activereports.server>
```

Hide Log In and Log Out Buttons

When you add the Designer control to your custom web application and use a custom security provider, you probably do not want to show our built-in Log In and Log Out buttons on the designer.

To hide the authentication buttons

1. On the Design view of your web form, select the Designer control.
2. In the Properties window, locate the **SecurityToken property (on-line documentation)**.
3. Enter a string value representing a valid security token.

You can obtain a security token from the IReportService **Login Method**.

Open the Designer without Selecting a Model

If you want to make report designing even faster for your business users, you can allow them to skip the steps involved in selecting a data model. Use the new DataModel property so they can just click a button to open the designer on a blank report with a pre-selected data model, or copy the Create report link from the models list.

This topic assumes that you are already familiar with how to **Use the Designer Web Control**.

To code a button click event to open a blank report with data

1. From the Visual Studio Standard toolbox, drag a Button control and drop it onto your Web form.
2. Double-click the button to create a Click event handling method in the code view.
3. Paste code like the following into the event to open a blank report in the designer with a pre-selected data model.

C# code. Paste INSIDE the Button Click event.

```
this.Designer1.ModelName = "ActiveTunes (Sample)";
this.Designer1.ReportAction = ActiveReports.Server.ReportControls.ReportAction.Create;
```

To copy the Create report link

1. From the Administrator Dashboard, open the Models page.
2. In the list of models, find the model that you want to use and, to the right of the model, click the **Create report** command.
3. In the browser's address bar, select the URL and copy it.
4. You can use this link as a hyperlink or in a response.redirect command.

Create a Custom Security Provider

You must create custom security providers in order to provide security filters for model entities. This allows you to add single sign-on functionality using LDAP and Active Directory, and to control access to data with row-level security.

Once you implement interfaces from the Extensibility assembly and configure the resulting assembly in the Administrator Dashboard by selecting it in the Security Provider list, you and your administrators can add security filters to entities when you edit a model. For more information, see the Managing Security Providers and Modifying an Entity topics in the Adminstrator Guide.

You can use your UserContext attributes in the connection string of your data models by encasing them in percent signs. For more information, see the Changing the Connection String topic in the Administrator Guide

To implement Extensibility interfaces in a .NET 3.5 Class Library

1. In Visual Studio, create a new C# .NET Framework 3.5 Class Library project.
2. From your ActiveReports 8 Server installation folder, copy the **ActiveReports.Server.Extensibility.dll** file into your project's **bin** folder.
3. In the Visual Studio Solution Explorer, right-click **References** and select **Add Reference**.
4. In the Add Reference dialog that appears, on the **Browse** tab, look in the bin folder, select the **ActiveReports.Server.Extensibility.dll** and click **OK**.
5. Add a class to your project that implements the ISecurityProvider interface, and another that implements the ISecurityProviderFactory interface.

   Note: For details on how to do this, please see the **ActiveTunes.SecurityProvider** sample in the C:\ActiveReports 8 Server\SDK directory.

   Important methods in the ISecurityProvider interface include:
   - **CreateToken** creates the security token that the server holds to identify the current logged in session with **username** and **password** parameters, as well as a **custom** parameter for any other credentials data used by the provider.
   - **FilterRoles** gets the collection of **roles** associated with the specified security **token**.
   - **GetUserContext** returns the UserContext object for the specified security **token**.
   - **GetUserDescription** returns two useful objects:
     - **UserDescription.Email** automatically fills the email field in the error log submission dialog.
     - **UserDescription.FriendlyUserName** automatically fills the CreatedBy and ModifiedBy fields for reports.
6. Once all of the necessary classes are in place with your custom data, from the Build menu, select **Build ClassLibrary1**. A DLL is created in your project's **bin > Release** folder.

To configure the provider

1. In Windows Explorer, create a SecurityProviders folder on the same level as your site, for example, if you have C:\ActiveReports 8 Server, create the folder C:\ActiveReports 8 Server\SecurityProviders.
2. Copy your custom security provider DLL file into the new folder.
3. On the Administrator Dashboard, in the Configuration section, click **Security Provider**.
4. From the **Custom security provider** drop-down that appears, select your new provider.
5. Select values for any properties that may be available, depending on your provider.
6. Click the **Apply changes** button.

To add a row-level security filter

1. On the Administrator Dashboard, in the Administration section, click **Models**.
2. In the list of models that appears, next to the model containing the entity that you want to filter, click the **Edit** command.
3. In the model editor that appears, from the **Entities** list to the left, click the entity that you want to filter.
4. In the workspace at the center, next to the **Security Filter** property, click the **Add** command.
5. In the Edit Security Filter dialog that appears, select **Filter Expression**, and click the ellipsis button next to the box under that label.
6. In the Select Attribute dialog that appears, expand nodes as necessary to select the attribute to which you want to apply a filter condition, and click **OK**.
7. In the center, you can choose whether the attribute **equals** or does **not equal** the value to the right.
8. In the drop-down box to the right, select the value from your security provider to compare to the attribute on the left.
9. Click **Apply** to add the filter to the entity.

Debug a Security Provider

Once you have created a custom security provider, you may need to debug it. The following steps walk you through how to do this using the sample custom security provider project installed with ActiveReports 8 Server. By default, this sample is in the following folder.

C:\ActiveReports 8 Server\SDK\ActiveTunes.SecurityProvider

To deploy the custom security provider for debugging

1. If you have not done so already, open the project in Visual Studio and from the Build menu, select **Build Solution**.
2. If you have not done so already, in the C:\ActiveReports 8 Server directory, create a new folder named **SecurityProviders**.
3. From your project bin\Debug\ folder, copy the following files, and paste them into C:\ActiveReports 8 Server\SecurityProviders\ to make the security provider and debugging file available to ActiveReports 8 Server.
   - ActiveTunes.SecurityProvider.dll
   - ActiveTunes.SecurityProvider.pdb
   - System.Data.SQLite.dll
4. Open the ActiveReports Administrator Dashboard in your browser, and under Configuration, select **Security Provider**.
5. Next to **Custom security provider**, drop down the list and select ActiveTunesSecurityProviderFactory. The page refreshes to allow you specify security provider settings and to test the configured provider.

   ⚠ **Important**: If you need to update the DLL at any point, you must first change this value back to **Default (builtin)**. You can change it back after you finish updating the DLL.

6. Under Security Provider Settings, for the ConnectionString property, enter the following connection string and click **Apply changes**.

   Connection string

   ```
   Data Source=C:\ActiveReports
   8 Server\SDK\ActiveTunes.SecurityProvider\ActiveTunes.sqlite
   ```

To debug the custom security provider

In order to perform the following steps, you must run Visual Studio as Administrator.

1. From the Visual Studio **Tools** menu, select **Attach to Process**.
2. In the Attach to Process dialog that appears, select the **Show processes in all sessions checkbox**, or navigate to the remote machine where the server web site resides.
3. Scroll down and select the **w3wp.exe** process for server's web site and click **Attach**. The project goes into Debugging mode.

   > **Caution**: There may be multiple w3wp.exe processes, so be sure to attach the one on the server machine running the ActiveReports services.

4. In the ActiveTunesSecurityProvider.cs file, locate the **ActiveTunesSecurityProvider.CreateToken** method and click in the margin to the left to set a breakpoint on it.
5. In your browser, on the Administrator Dashboard Security Provider page in the **Login to test** and **Password to test** boxes, enter test values from the table below.

Login	Password
andrew@chinookcorp.com	1
nancy@chinookcorp.com	2
jane@chinookcorp.com	3
margaret@chinookcorp.com	4
steve@chinookcorp.com	5
michael@chinookcorp.com	6
robert@chinookcorp.com	7
laura@chinookcorp.com	8

6. Click the **Test provider** link. The Visual Studio window takes focus and shows the breakpoint, allowing you to debug the code.

To debug your custom security provider with Visual Studio on the server

Here is another way to debug your security provider if you have Visual Studio installed on the server and have direct access to the server machine.

> **Caution**: Ensure that you remove this code before you go live with your deployment.

1. Open your SecurityProvider implementation, and in one of its methods, for example, ISecurityProviderFactory.Create, add the following code.

   C# code. Paste inside the method to debug.

   ```
   System.Diagnostics.Debugger.Break();
   ```

2. From the Build menu, select **Rebuild Solution**.
3. Copy the resulting DLL and PDB files and paste them into C:\ActiveReports 8 Server\SecurityProviders\.
4. Open the ActiveReports Administrator Dashboard in your browser, and under Configuration, select **Security Provider**.
5. Next to **Custom security provider**, drop down the list and select your security provider. The page refreshes to allow you specify any security provider settings specified in your provider.

   > **Important**: If you need to update the DLL at any point, you must first change this value back to **Default (builtin)**. You can change it back after you finish updating the DLL.

6. When you run a test or try to log in as a user, The Visual Studio Just-In-Time Debugger prompt appears asking whether you want to debug w3wp.exe.
7. Click **Yes** to open Visual Studio and begin debugging.

Work with the HTML5 Viewer using Javascript

Following are the options, methods and properties available to work with HTML5 Viewer using the Javascript.

Initializing HTML5 Viewer

1. Copy and place GrapeCity.ActiveReports.Viewer.Html.js, GrapeCity.ActiveReports.Viewer.Html.min.js and GrapeCity.ActiveReports.Viewer.Html.css files at a suitable location near the target HTML page.

   > **Note**: In ActiveReports 8 Server, these files are located in the ...\ActiveReports 8 Server\SDK\HTML5 Viewer.Source folder.

2. In the target HTML page, add the references to the GrapeCity.ActiveReports.Viewer.Html.css, GrapeCity.ActiveReports.Viewer.Html.js and its following dependencies:

- jQuery 1.9.0 or higher
- Bootstrap 3.0
- Knockout.js 2.3.0 or higher

Note: You can obtain the dependencies like jQuery from Content Delivery Network (CDN) or copy them locally.

HTML

```
<link rel="stylesheet" href="GrapeCity.ActiveReports.Viewer.Html.css" >
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
<script src="//netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/knockout/2.3.0/knockout-min.js" type="text/javascript"></script>
<script src="Scripts/GrapeCity.ActiveReports.Viewer.Html.js" type="text/javascript"></script>
<script type="text/javascript">
</script>
```

3. In the target HTML page, add the DIV element that will contain the HTML5 Viewer.

HTML

```
<div id="viewer" style="width:600px;height:480px;"></div>
```

4. Add the following code to initialize the HTML5 Viewer. The code might vary depending on the technology that you use to develop the HTML5 Viewer component. To learn how to get a security token for the reportService, see **Get a Security Token**.

Javascript

```
$(function ()
{
   var viewer = GrapeCity.ActiveReports.Viewer(
   {
      element: '#viewer',
      report: {
         id: "myreport"

      },

      reportService: {
          url: 'http://myar8server.com/ReportService.svc/json/',
          securityToken: securityToken,
          resourceHandler : 'http://myar8server.com/TemporaryResource.axd/'
      },
      uiType: 'desktop',
      documentLoaded: function reportLoaded()
      {
         console.log(viewer.pageCount);
      },
      reportLoaded: function (reportInfo)
      {
         console.log(reportInfo.parameters);
      },

      error: function (error)
      {
         console.log("error");
      }

   });
});
```

5. Add the following code to the system.webServer section of the web.config file of your ActiveReports 8 Server web site.

XML

```
<system.webServer>

        <httpProtocol>

                <customHeaders>

                        <add name="Access-Control-Allow-Origin" value="*" />

                        <add name="Access-Control-Allow-Headers" value="Authorization, Origin, Content-Type, Accept, X-Requested-With" />

                </customHeaders>

        </httpProtocol>

...

</system.webServer>
```

Initialization Options

The following options can be set during initialization or at runtime while working with the HTML5 Viewer.

uiType

Description: Sets the UI mode of the HTML5 Viewer.

Type: String

Accepted Value: 'Custom', 'Mobile' or 'Desktop'

Example:

```
viewer.option('uiType', 'Mobile');
```

element

Description: JQuery selector that specifies the element that hosts the HTML5 Viewer control.

Note: This option is used during initialization only.

Type: String

Example:

```
var viewer = GrapeCity.ActiveReports.Viewer( { element: '#viewerContainer2', reportService: { url: '/ActiveReports.ReportService.asmx' }, });
```

reportService

Description: The report service that can use ActiveReports 8 Server or ActiveReports Web Report Service.

Type: Object that has the url and optional securityToken properties

Example:

```
reportService: { url: 'http://remote-ar-server.com', securityToken: '42A9CD80A4F3445A9BB60A221D042FCC', resourceHandler: 'http://remote-ar-server.com/resourceHandler.aspx' };
```

reportService.url

Description: The url of ActiveReports 8 Server instance of the ActiveReports Web service that provides the reportInfo and output.

Type: String

Example:

```
reportService: { url: 'http://remote-ar-server.com' };
```

reportService.securityToken

Description: The security key needed to login to the ActiveReports 8 Server.

Type: String

Example:

```
reportService: { securityToken: '42A9CD80A4F3445A9BB60A221D042FCC' };
```

reportService.resourceHandler

Description: The url of the ActiveReports 8 Server resource handler.

Type: String

Example:

```
reportService: { resourceHandler: 'http://remote-ar-server.com/resourceHandler.aspx' };
```

report

Description: The report that is displayed in ActiveReports 8 Server or ActiveReports Web Report Service.

Type: An object that has id and parameters properties.

Example:

```
report: { id: 'CustomersList', parameters: [ { name: 'CustomerID', value: 'ALFKI' }] };
```

reportID

Description: The id of the report to be shown by the HTML5 Viewer.

Type: String

Example:

Example:

```
report: { id: 'CustomersList', parameters: [ { name: 'CustomerID', value: 'ALFKI' }] };
```

reportParameters

Description: The array of the {name, value} pairs that describe the parameters values used to run the report.

Type: Array

Example:

```
report: { id: 'CustomersList', parameters: [ { name: 'CustomerID', value: 'ALFKI' }] };
```

reportLoaded

Description: The callback that is invoked when the HTML5 Viewer obtains the information about the requested report. The reportInfo object is passed in the callback including the TOC info, Parameters info and the link to the rendered report result.

Type: function(reportInfo)

Example:

```
var reportLoaded = function reportLoaded(reportInfo) { console.log(reportInfo.parameters); } viewer.option('reportLoaded', reportLoaded);
```

action

Description: The callback that is invoked before the HTML5 Viewer opens the hyperlink, bookmark link, drill down report or toggles the report control visibility.

Type: function(actionType, actionParams)

Example:

```
function onAction(actionType, actionParams) { if (actionType === 0) { window.open(params.url, "Linked from report", "height=200,width=200"); } } viewer.option('action', onAction);
```

availableExports

Description: The array of export types available via Export functionality of HTML5 Viewer. By default, PDF, Word, Image, Mht, and Excel exports are available.

Type: Array

Example:

```
viewer.option("availableExports", ['Pdf']);
```

maxSearchResults

Description: The number of search results received for a single search invoke.

Type: Number

Example:

```
maxSearchResults: 10
```

error

Description: The callback that is invoked when an error occurs in the process of displaying the report.

Type: function(error)

Example:

```
var onError = function onError(errorInfo) { console.log(errorInfo); return true; } viewer.option('onError', onError);
```

documentLoaded

Description: The callback that is invoked when a document is loaded entirely on the server.

Type: function()

Example:

```
var documentLoaded = function documentLoaded() { setPaginator(); } viewer.option('documentLoaded', documentLoaded);
```

localeUri

Description: The url of the file containing the localization strings.

> **Note**: This option is used during initialization only.

Type: String

Example:

```
var viewer = GrapeCity.ActiveReports.Viewer( { localeUri: 'Scripts/i18n/ru.txt' });
```

Public API Methods and Properties

After initializing the HTML5 Viewer, the following API methods and properties can be used.

Methods

option

Description: Gets or sets the option value by name if the value parameter is specified.

Syntax: `option(name,[value])Object`

Parameters:

- **name**: The option name to get or set.
- **value**: (optional) The option value to set. If this argument is omitted than the method returns the current option value.

Example:

```
viewer.option('uiType', 'mobile'); viewer.option('report', { id: 'my report' });
```

Return Value: The current option value.

refresh

Description: Refreshes the report preview.

Syntax: `option(name,[value])Object`

Example:

```
viewer.refresh()
```

Return Value: Void

print

Description: Prints the currently displayed report if any.

Syntax: `print()Void`

Example:

```
viewer.print()
```

Return Value: Void

goToPage

Description: Makes the viewer to display the specific page, scroll to the specific offset (optional) and invokes the callback once it's done.

Syntax: `goToPage(number, offset, callback)Void`

Parameters:

- **number**: The number of pages to go to.
- **offset object**: The object such as {left:12.2, top:15}.
- **callback**: The function to call after perform action.

Example:

```
viewer.goToPage(1, { 2, 3 }, function onPageOpened() {});
```

Return Value: Void

backToParent

Description: Makes the viewer to display the parent report of the drill-down report.

Syntax: `backToParent()Void`

Example:

```
viewer.backToParent()
```

Return Value: Void

destroy

Description: Removes the viewer content from the element.

Syntax: `destroy()Void`

Example:

```
viewer.destroy()
```

Return Value: Void

export

Description: Exports the currently displayed report.

Syntax: `export(exportType,callback,saveAsDialog,settings)Void`

Parameters:

- **exportType**: Specifies export format.
- **callback**: Function that is invoked once the export result is available (its Url is passed in the callback).
- **saveAsDialog**: Indicates whether the save as dialog should be shown immediately once the export result is ready.
- **settings**: The export settings, vary for each export type.

Example:

```
viewer.export('Word', function () { console.log('export callback'); }, true, { FileName: 'Document.doc' })
```

Return Value: Void

search

Description: Performs the search of a specific term with specific search options (match case, whole word) and invokes the specific callback with the search result passed.

Syntax: `search(searchTerm, searchOptions, callback)Void`

Parameters:

- **searchTerm**: String to find.
- **searchOptions**: The object optionally defines the search options:
  - **matchCase**: Whether the search should respect the case.
  - **wholePhrase**: Whether the search should look for a whole phrase.
- **callback**: The function to call after performing search.

Example:

```
viewer.search('a', { matchCase: true, wholePhrase: false }, function (results) { console.log(results); });
```

Return Value: Void

getToc

Description: Obtains the part of the report TOC specified by parent(node), startChild(index), count parameters and invokes callback function passing the result as parameter.

Syntax: `getToc(callback) Void`

Parameters:

- **callback**: The callback to handle TOC tree.

Example:

```
viewer.getToc(function (toc) { console.log(toc); })
```

Return Value: Void

Properties

pageCount

Description: Gets the page count of the currently displayed report.

Syntax: `viewer.pageCount`

Example:

```
console.log(viewer.pageCount)
```

Return Value: An integer representing page count.

currentPage

Description: Gets the currently displayed page number.

Syntax: `viewer.currentPage`

Example:

```
console.log(viewer.currentPage)
```

Return Value: An integer representing currently displayed page number.

Toolbar

Description: Returns the HTML element that displays the toolbar in desktop UI mode. The developer may use it to add the custom elements or remove the existing ones using jQuery/Html/Css capabilities.

Syntax: `viewer.Toolbar`

Example:

```
// Toolbar, MobileToolbarTop, MobileToolbarBottom $(viewer.toolbar).hide(); $(viewer.toolbarTop).hide(); $(viewer.toolbarBottom).hide();
```

ToolbarTop

Description: Returns the HTML element that displays the top toolbar in mobile UI mode. The developer may use it to add the custom elements or remove the existing ones using jQuery/Html/Css capabilities.

Syntax: `viewer.ToolbarTop`

Example:

```
// Toolbar, MobileToolbarTop, MobileToolbarBottom $(viewer.toolbar).hide(); $(viewer.toolbarTop).hide(); $(viewer.toolbarBottom).hide();
```

ToolbarBottom

Description: Returns the HTML element that displays the bottom toolbar in mobile UI mode. The developer may use it to add the custom elements or remove the existing ones using jQuery/Html/Css capabilities.

Syntax: `viewer.ToolbarBottom`

Example:

```
// Toolbar, MobileToolbarTop, MobileToolbarBottom $(viewer.toolbar).hide(); $(viewer.toolbarTop).hide(); $(viewer.toolbarBottom).hide();
```

Get a Security Token

When you use the HTML5 Viewer to access the ActiveReports 8 Server web site, you need to provide the Report Service with a security token to use.

Below are the steps to retrieve a security token.

To get a security token by the Javascript function

1. In the target HTML page, add the following code to get a security token. The code might vary depending on the technology that you use to develop the HTML5 Viewer component. For more information, see the **HTML5 Viewer Sample**. By default, this sample is in the following folder.
   C:\ActiveReports 8 Server\SDK\Samples\HTML5 Viewer.

   > **Note:** You need to change these settings in the code below.
   >
   > - **url** parameter represents the location (reportservice.svc/json/login) which stays constant where as the arsEndpoint represents the URL used to access the Report Portal Website. **Example:** http://<ActiveReports8ServerName>:<portnumber>.
   > - **username** is your username to log into the ActiveReports 8 Server web site.
   > - **password** is your password to log into the ActiveReports 8 Server web site.

   Javascript

```
function getSecurityToken() {
    if (_securityToken) return _securityToken;
    _securityToken = "error";

    try {
        $.ajax({
             async: false,
             type: "POST",
             url: arsEndpoint + "ReportService.svc/json/Login",
             data: JSON.stringify({
                   username: "username",
                   password: "password"
```

```
            }),
            contentType: "application/json",
            dataType: "json"
        }).done(function(res) {
              _securityToken = res.d;
        }).fail(function() {
              _securityToken = "error";
        });
    } catch(e) {
          return "error";
    }

    return _securityToken;
}
```

To get a security token from the cache

1. Open your ActiveReports 8 Server web site and then open the web browser **Developer Tools**.

   Note: Press F12 to open the web browser Developer Tools in Google Chrome, Internet Explorer, or Mozilla Firefox.

2. In the **Developer Tools** window, go to the **Resources** tab and then to the **Cookies > ActiveReports 8 Server name**.
3. In the **Reports** list of the Administrator Dashboard, click **Preview** to the right of any report to open it.
4. Find the string that starts with **SECURITYTOKEN=**.
5. Copy the value till the next delimiter and paste it to the Javascript code used for the HTML5 Viewer options.

Samples and Walkthroughs

These topics describe the SDK samples and provide step-by-step walkthroughs detailing ActiveReports 8 Server project creation.

Samples
These topics explain how to access the sample files installed with ActiveReports 8 Server.

Walkthroughs
Learn to work with ActiveReports 8 Server controls in Visual Studio.

Samples

The ActiveReports 8 Server installation includes several sample projects in a Visual Studio 2008 solution. You can find the solution in:

C:\ActiveReports 8 Server\SDK\Samples

To open the solution, double-click the **ActiveReports 8 Server Samples.sln** file.

Sample Project	Description
ASP.NET Integration	Shows how you can embed the Report Designer control in your own applications, and how you can customize the Report List control. It also demonstrates how to use the Web service features using javascript.
Console Web Service Client	Shows how you can use the Web service features to serve reports to Windows applications.

Also included is a separate Security Provider Sample. See **Security Provider Sample** for details.

Security Provider Sample

The ActiveReports 8 Server installation includes a security provider sample that you can find in:

C:\ActiveReports 8 Server\SDK\ActiveTunes.SecurityProvider

To open the solution, double-click the **ActiveTunes.SecurityProvider.sln** file.

Sample Project	Description
ActiveTunes.SecurityProvider	Shows how you can provide proprietary or single-sign-on authentication instead of the built-in ActiveReports 8 Server security and use row-level security.
ActiveTunes.SecurityProvider.UnitTests	Shows how you can test the row-level security using sample data.

To compile and use the Security Provider sample

1. With the sample open in Visual Studio, from the **Build** menu, select **Build Solution**.
2. In Windows Explorer, open the following folder, and copy the **ActiveTunes.SecurityProvider.dll** assembly.
   C:\ActiveReports 8 Server\SDK\ActiveTunes.SecurityProvider\ActiveTunes.SecurityProvider\bin\Debug
3. Paste the assembly into the following folder in your ActiveReports 8 Server installation folder.
   C:\ActiveReports 8 Server\SecurityProviders
4. On the Administrator Dashboard, in the Configuration section, click **Security Provider**.
5. From the **Custom security provider** drop-down that appears, select **ActiveTunesSecurityProviderFactory**.
6. In the **ConnectionString** property that appears, enter the connection string found in the ActiveTunes.SecurityProvider.UnitTests project file App.config:

ConnectionString

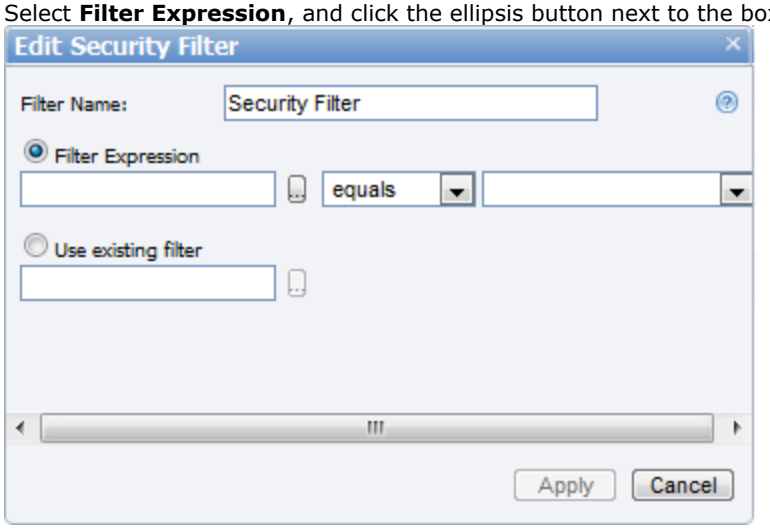
```
Data Source=C:\ActiveReports 8
Server\SDK\ActiveTunes.SecurityProvider\ActiveTunes.sqlite
```

7. Click the **Apply changes** button.

Now, when you edit the ActiveTunes sample model, you can apply either the EmployeeID or CustomerID filter to any entity.

To add row-level security to the ActiveTunes sample model

1. On the Administrator Dashboard, in the Administration section, click **Models**.
2. In the list of models that appears, next to the **ActiveTunes (Sample)** model, click the **Edit** command.
3. In the model editor that appears, from the **Entities** list to the left, click the entity that you want to filter.
4. In the workspace at the center, next to the **Security Filter** property, click the **Add** command.
5. In the Edit Security Filter dialog that appears, in the box next to **Filter Name**, enter a name for your filter.
6. Select **Filter Expression**, and click the ellipsis button next to the box under that label.



7. In the Select Attribute dialog that appears, select the attribute to which you want to apply the filter condition, and click **OK**.
8. To the right of the **equals** option, click the drop-down arrow and select **UserContext.CustomerID** or **UserContext.EmployeeID**.
9. Click **Apply** to add the filter to the entity.

Walkthroughs

Learn to use ActiveReports 8 Server controls and API in Visual Studio.

Accessing the Web Service
Learn to access the Web service from a console application

Using the ReportList Web Control
This topic explains how to use the ReportList control on a Web Form.

Uploading a Code-Based Section Report
This topic explains how to upload a code-based section report by the UploadResource method.

Accessing the Web Service

This topic explains how to access the ActiveReports 8 Server Web service from a Visual Studio console application.

To create a console application that consumes the Web service

You can use a console application to create and access the ActiveReports 8 Server Web service.

1. In Visual Studio, create a new console application:
   Steps to create a console application
   a. From the **File** menu, select **New Project**.
   b. In the New Project dialog that appears, in the left pane, select **Visual C#**, then **Windows**.

c. In the center pane, select **Console Application**.
d. Enter a name for the application, and click **OK**.

2. Add a service reference to the ActiveReports 8 Server service:
Steps to add a service reference

a. In the Solution Explorer, right-click the console application and select **Add Service Reference**.
b. In the Add Service Reference dialog that appears, for the Address, enter:

Paste in the Address box, replacing 8080 with the site port where you installed ActiveReports 8 Server.

```
http://localhost:8080/ReportService.svc
```

c. Click **Go**, and when the ReportService appears in the Services pane, you can expand it and select the service interface to reveal its available operations in the pane to the right.
d. Enter a Namespace to use, for example, ARSReportService and click **OK**. The reference appears in the Solution Explorer.

3. Add using directives to the files:
Steps to add using directives

a. In the **Program.cs** file, paste the following using directives under the existing ones, changing the name of the application to reflect the name of your application:

Paste at the top of the Program.cs file, replacing ConsoleApplication1.ARS*Service with the name of your application and service namespaces.

```
using ConsoleApplication1.ARSReportService;
using ActiveReports.Server.ServiceClients;
```

b. Paste the same using directives in the **DefaultClientFactory.cs** file as well.

4. Add code to the **Program.cs** file to log in to the services:
Steps to add code to log in

a. In the **Program.cs** file, paste the following code in place of the existing *static void Main(string[] args) {}*:

Paste in the Program.cs file inside the class Program brackets, replacing the default static void Main.

```
private static readonly DefaultClientFactory ClientFactory = new
DefaultClientFactory();

static void Main(string[] args)
{
    var server = new Uri("http://localhost:8080");
    var userName = "MyUserName";
    var password = "MyPassword";

    using (var authenticationService =
ClientFactory.CreateAuthenticationServiceClient(server))
    {
        var isLogged = authenticationService.Login(userName, password,
string.Empty, false);
        Console.WriteLine(isLogged != null
            ? "Logged in."
            : "Wrong user name or password.");
    }
    Console.ReadLine();
}
```

b. Change the number **8080** after localhost to the site port where you installed ActiveReports 8 Server.
c. Change **MyUserName** and **MyPassword** to your own user name and password.

5. Run the project to see the console application flash a "Logged in." message.

Using the ReportList Web Control

The Web controls need to be directed to the server used for ActiveReports 8 Server in order to function. In your

production application, specify the user name and password based on the current user and store information in the current session to persist these values.

To create a Log In page

1. In Visual Studio, create a new C# ASP.NET Web Site.
2. Open the Design view of your default Web Form and from the Standard section of the Toolbox, drag the following controls and drop them onto the body section of the Web Form, setting their properties as in the table below.
   Controls for the Log In page

Control	Text Property
Label	User Name:
TextBox	?
Label	Password:
TextBox	?
Button	Log In

3. Double-click the button to create the button click event. The *.cs file for the page appears with the cursor inside the button click event.
4. In the button click event, paste the following code to get the security token from the user.
   To get the security token from the user

   C# code. Paste INSIDE the button click event.

```
var username = TextBox1.Text;
var password = TextBox2.Text;

var reportService = new ReportServiceProxy();
HttpContext.Current.Session["arsSecurityToken"] = null;

var securityToken = reportService.Login(username, password, null, false);

HttpContext.Current.Session["arsSecurityToken"] = securityToken;
Server.Transfer("Default2.aspx");
```

5. At the top, with the other using directives, paste the following code to use the Servicing namespace.
   To get the security token from the user

   C# code. Paste BELOW the other using directives.

```
using ActiveReports.Server.ReportControls.Servicing;
```

To add the ReportList control to a Web Form

1. Add a second Web Form to your project, **Default2.aspx**.
2. Open the Design view of your Web Form and from the toolbox, drag and drop the ReportList control onto the body section of the Web Form.
   If you need to add the control to your toolbox, drop down these steps.
   a. Right-click in the General tab and select **Choose items**.
   b. In the Choose Toolbox Items dialog that appears, the .NET Framework Components tab is selected by default. Click the **Namespace** column header to sort by namespace.
   c. The ActiveReports.Server.ReportControls namespace is at or near the top of the list. Select the checkbox next to **ReportList** and click **OK**.
   d. The ReportList control appears in your Visual Studio toolbox.
3. On the message box that appears, click **Yes** to automatically add .svc file to your Web site to give the report list access to proxy services.
4. In the Properties window, set the **ServerEndpointRootPath** property to the URL that you want to use as the basis for all of the other URLs in your site, which can then be set as relative URLs.
5. You can resize the control using the Properties grid by changing the **Width** property.

To direct the ReportList to the server used for ActiveReports 8 Server

You can specify the security token and ActiveReports 8 Server host using code in a Global Application Class.

1. From the Visual Studio **Website** menu, select **Add New Item**.
2. In the Add New Item dialog that appears, select Global Application Class and click **Add**.
3. Below the Application Language tag at the top of the file, import the Servicing namespace so that you can use the ReportServiceProxy with a directive like the following:
   To import the Servicing namespace

   ASP.NET code. Paste on the line BELOW the Application Language line.

   ```
   <%@ Import Namespace="ActiveReports.Server.ReportControls.Servicing" %>
   ```

4. Below the Application Start event, create the ResolveRemoteEndpoint event using code like the following, but with your address and security token:
   To create the ResolveRemoteEndpoint event

   C# code. Paste in the Global.asax file AFTER the Application_Start event.

   ```
   static void ResolveRemoteEndpoint(RemoteEndpoint remoteEndpoint)
   {
       remoteEndpoint.Address = "http://localhost:8080";
       remoteEndpoint.SecurityToken = "SecurityToken";
   }
   ```

5. Provide a handler for the ResolveRemoteEndPoint event using code like the following in the Application Start event:
   To provide a handler for the event

   C# code. Paste INSIDE the Application_Start event.

   ```
   ReportServiceProxy.ResolveRemoteEndpoint += ResolveRemoteEndpoint;
   ```

You can also specify the user name, password, and ActiveReports 8 Server host in your web application's web.config file.

1. In the web.config file, provide a handler for the ResolveRemoteEndPoint event using code like the following.
   To provide a handler for the event

   C# code. Paste BETWEEN the <configSections> and </configSections> tags.

   ```
   <sectionGroup name="activereports.server">
     <section name="reportServiceProxy"
type="ActiveReports.Server.ReportControls.Configuration.ActiveReportsServerSection,
     ActiveReports.Server.ReportControls, Version=x.x.xxxx.x, Culture=neutral,
PublicKeyToken=d557f2f30a260da2" allowDefinition="Everywhere" />
   </sectionGroup>
   ```

2. Below the Application Start event, create the ResolveRemoteEndpoint event using code like the following, but with your address, user name, and password
   To import the Servicing namespace

   ASP.NET code. Paste in the web.config file AFTER the </configSections> tag.

   ```
   <activereports.server>
     <reportServiceProxy remoteReportServicePath="http://localhost:8010/"
username="MyUserName" password="MyPassword" />
   </activereports.server>
   ```

At run time, the ReportList control retrieves all of the reports from ActiveReports 8 Server, and provides buttons to export each to PDF, Word, and Excel.

Uploading a Code-Based Section Report

You can upload all types of developer-created ActiveReports to the server using the **UploadResource** method. This topic explains how to upload a code-based section report.

1. From the Visual Studio **File** menu, select **New Project**.
2. In the **New Project** dialog that appears, in the list of templates under Visual C#, then Windows, select **Console Application**.
3. Rename the project to **UploadReport** and click **OK**.

> **Note**: The target framework of the project must be set to .NET Framework 4.0

4. From the Visual Studio **Project** menu, select **Add Service Reference**.
5. In the **Add Service Reference** dialog that appears, enter the service URL on a server in the **Adress** field and rename the namespace to **ReportService**.
6. In Program.cs, add the following using statements to the list of using statements at the top of the code.

Add to the list of using statements at the top of the code.

```
using System.IO;

using UploadReport.ReportService;
```

7. In Program.cs, add the following code into the Main method of the Program class declaration.

Paste the following code into the Main method of the Program class declaration

```
    var reportAssemblyName = "ReportLib, Version=1.0.0.0"; // the name of assembly
containing the section report to upload
    var reportsAssemblyPath = @"D:\temp\ReportLib.dll"; // the path to assembly
containing the section report to upload
    var reportClassName = "ReportLib.SectionReport"; // the fully qualified class name
of the section report to upload
    var serverUserName = "admin";
    var serverUserPwd = "1";
    var uploadOptions = new UploadOptions {Overwrite = true};
    var reportAssemblyContent =
Convert.ToBase64String(File.ReadAllBytes(reportsAssemblyPath));
    var reportService = new ReportServiceClient("WSHttpBinding_IReportService");
    var securityToken = reportService.Login(serverUserName, serverUserPwd, null,
true);
    var result = reportService.UploadResource(securityToken, new AssemblyDescription
{Name = reportAssemblyName},
                                              reportAssemblyContent, uploadOptions);
    var assemblyId = (result.ItemDescriptions[0] as AssemblyDescription).Id;
    var reportDescription = new ReportDescription()
                                {
                                    Name = "Section Report",
                                    ReportType = ReportType.CodeBasedSectionReport,
                                    ClassName = reportClassName,
                                    AssemblyResourceId = assemblyId
                                };

    reportService.UploadResource(securityToken, reportDescription ,
Convert.ToBase64String(new byte[0]), uploadOptions);
```

8. Run the project.

Class Library

The Class Library contains documentation and code samples of the ActiveReports 8 Server API.

In This API Documentation

ActiveReports.Server.ReportControls ('ActiveReports.Server.ReportControls Namespace' in the on-line documentation)
View the API documentation on the report list and designer controls.

ActiveReports.Server.ReportControls.Servicing ('ActiveReports.Server.ReportControls.Servicing Namespace' in the on-line documentation)
View the API documentation on the service proxies.

ActiveReports.Server.Security ('ActiveReports.Server.Security Namespace' in the on-line documentation)
View the API documentation on the security interfaces.

Web Service Library

The Web Service Library contains documentation and syntax for the ActiveReports 8 Server ReportService.

Service

ReportService
Checks user credentials and creates a security token if the credentials are valid.

Objects

DataResult Object
Represents an object containing the status of an IReportService operation.

EMailDistribution Object
Defines e-mail options to use in distributing the report.

ExceptionDetail Object
Defines detailed exception information for an IReportService request operation error.

ItemDescription Object
Contains descriptive information about stored resources.

ItemDescriptionsResult Object
Represents an object that is returned as a result of an Upload operation and contains an array of item descriptons.

Query Object
Defines a query used to find reports in the Select operation.

RenderOptions Object
Defines options for how to render a report.

ReportDescription Object
Contains descriptive information about stored reports.

ReportParameter Object
Defines a parameter for the report.

RequestInfo Object
Defines status information for an IReportService request operation.

RequestResult Object
Represents an object containing the status, or result contract, of an IReportService request operation.

Result Object
Represents an object that is returned as a result of an IReportService operation.

UploadOptions Object
Defines options for how to upload a resource.

UserCapabilities Object
Represents a user capabilities object that is contained in a result from an IReportService get user capabilities operation.

UserCapabilitiesResult Object
Represents a user capabilities result object that is returned as a result of an IReportService get user capabilities operation.

Enumerations

ModelPermission Enumeration
Specifies the byte value of the enumerated level of permissions granted for the model associated with the report.

ReportParameterDomain Enumeration
Specifies the byte value of the enumerated report parameter domain.

ReportType Enumeration
Specifies the byte value of the enumerated report type for all supported reports.

RequestState Enumeration
Specifies the state of an IReportService request operation for a RequestInfo object.

ReportService

Checks user credentials and creates a security token if the credentials are valid.

Syntax

C#

```
public interface IReportService
```

Parameters

username
A string representing the user name to validate.

password
A string representing the password for the specified user.

custom
A string representing the value or values to validate in addition to user name and password, if any.

isPersistent
A Boolean value that indicates whether the security token remains valid across sessions.

Return Value

A security token if user credentials are valid; otherwise, **null**.

IReportService Methods

Name	Description
CancelRequest Method	Cancels the specified request.
Delete Method	Deletes the report specified in the ID parameter.
Download Method	Downloads the specified report from storage.
GetRequestStatus Method	Retrieves the status of the specified request.
GetUserCapabilities Method	Retrieves the capabilities of the user with the specified security token.
IsLoggedIn Method	Determines whether the security token is valid.
Login Method	Checks user credentials and creates a security token if the credentials are valid.
Logout Method	Recalls the security token.
RenderReport Method	Renders the report specified in the description using the specified options.
ResolveParameters Method	Populates the parameters for the specified report.
Select Method	Retrieves an array of report descriptions matching the criterion defined by the specified query.
SendReportEmail Method	Renders the specified report and sends it via email according to the specified options.
Upload Method	Uploads the specified report to storage.
UploadResource Method	Uploads the specified resource to storage.

CancelRequest Method

Cancels the specified request.

Syntax

C#

```
Result CancelRequest(string token, string id);
```

Parameters

token
The security token to use.

id
The identifier of the request to cancel.

Return Value

A **Result** object.

Delete Method

Deletes the report specified in the ID parameter.

Syntax

C#

```
DataResult Delete(string token, string id);
```

Parameters

token
The security token to use.

id
The identifier of the report to delete.

Return Value

A **DataResult** object containing the status of the delete operation.

Download Method

Downloads the specified report from storage.

Syntax

C#

```
DataResult Download(string token, string id);
```

Parameters

token
The security token to use.

id
The identifier of the report to download.

Return Value

A **DataResult** object containing the report downloaded by the operation.

GetRequestStatus Method

Retrieves the status of the specified request.

Syntax

C#

```
RequestResult GetRequestStatus(string token, string requestId);
```

Parameters

token
The security token to use.

id
The identifier of the request for which to return a status.

Return Value

A **Result** object.

GetUserCapabilities Method

Retrieves the capabilities of the user with the specified security token.

Syntax

C#

```
UserCapabilitiesResult GetUserCapabilities(string token);
```

Parameters

token
The security token to use.

Return Value

A **UserCapabilitiesResult Object**.

IsLoggedIn Method

Determines whether the security token is valid.

Syntax

C#

```
bool IsLoggedIn(string token);
```

Parameters

token
The security token to use.

Return Value

A Boolean value indicating whether the security token is valid.

Login Method

Checks user credentials and creates a security token if the credentials are valid.

Syntax

C#

```
string Login(string username, string password, string custom, bool
isPersistent);
```

Parameters

username
A string value indicating the user name to validate.

password
A string value indicating the password for the specified user.

custom
Other string values to validate in addition to the user name and password, if any.

isPersistent
A Boolean value indicating whether the security token remains valid across sessions.

Return Value

A string value containing the security token if the user credentials are valid; otherwise, **null**.

Logout Method

Recalls the security token.

Syntax

C#

```
void Logout(string token);
```

Parameters

token
The security token to recall.

RenderReport Method

Renders the report specified in the description using the specified options.

Syntax

C#

```
RequestResult RenderReport(string token, ReportDescription description,
RenderOptions options);
```

Parameters

token
The security token to use.

description

The **ReportDescription** object containing information about the report.

options
The **RenderOptions** object containing information about how to render the report.

Return Value

A **RequestResult** object.

ResolveParameters Method

Populates the parameters for the specified report.

Syntax

C#

```
RequestResult ResolveParameters(string token, ReportDescription description,
RenderOptions options);
```

Parameters

token
The security token to use.

description
The **ReportDescription** object containing information about the report.

options
The **RenderOptions** object containing information about how to render the report.

Return Value

A **RequestResult Object**.

Select Method

Retrieves an array of report descriptions matching the criterion defined by the specified query.

Syntax

C#

```
ReportDescription[] Select(string token, Query query);
```

Parameters

token
The security token to use.

query
The **Query** object to use in finding reports to select.

Return Value

A array of **ReportDescription** objects.

SendReportEmail Method

Renders the specified report and sends it via email according to the specified options.

Syntax

C#

```
Result SendReportEmail(string token, ReportDescription description,
RenderOptions options, EMailDistribution distribution);
```

Parameters

token
The security token to use.

description
The **ReportDescription** object containing information about the report.

options
The **RenderOptions** object containing information about how to render the report.

distribution
The **EMailDistribution** object containing information about how to distribute the report.

Return Value

A **Result** object.

Upload Method

Uploads the specified report to storage.

Syntax

C#

```
ItemDescriptionsResult Upload(string token, ReportDescription description,
string data);
```

Parameters

token
The security token to use.

description
The **ReportDescription** object containing information about the report to upload.

data
The string value representing the report data encoded in base 64 digits to upload.

Return Value

An **ItemDescriptionsResult** object.

UploadResource Method

Uploads the specified resource, a valid .NET assembly containing compiled ActiveReports, to storage.

Syntax

C#

```
ItemDescriptionsResult UploadResource(string token, ItemDescription
description, string data, UploadOptions options);
```

Parameters

token
The security token to use.

description
The **ItemDescription** object containing information about the resource to upload.

data
The string value representing the resource data encoded in base 64 digits to upload.

options
The **UploadOptions** object containing optional settings such as Overwrite and SkipResourceValidation.

Return Value

An **ItemDescriptionsResult** object.

DataResult Object

Represents an object containing the data returned by an IReportService operation.

Object Properties

	Name	Description
	Data Property	Gets the base 64 encoded data returned by the operation.

EMailDistribution Object

Defines e-mail options to use in distributing the report.

Object Properties

	Name	Description
	To Property	Gets or sets the string containing the address collection that contains the e-mail recipients.
	Subject Property	Gets or sets the string containing the subject for the e-mail.
	MessageBody Property	Gets or sets the string containing the message body of the e-mail.
	AsLink Property	Gets or sets a Boolean value indicating whether to add a link to the report within the e-mail.
	AsAttachment Property	Gets or sets a Boolean value indicating whether to attach the report to the e-mail.
	BaseUri Property	Gets or sets the base URI to use for the report link in the e-mail.

ExceptionDetail Object

Defines detailed exception information for an IReportService request operation error.

Object Properties

	Name	Description
	HelpLink Property	Gets the string for the help link from the exception.
	InnerException Property	Gets the ExceptionDetail object that represents the inner exception.

Message Property	Gets the string message from the exception.
StackTrace Property	Gets the string stack trace information from the exception.
Type Property	Gets the string representing the type of the exception.

ItemDescription Object

Contains descriptive information about uploaded resources.

Object Properties

Name	Description
CreatedBy Property	Gets the string containing the name of the user who uploaded the resource.
CreationDate Property	Gets the DateTime object containing the date on which the resource was uploaded.
Id Property	Gets the string containing the ID for the resource.
ModifiedBy Property	Gets the string containing the name of the user who last modified the resource.
ModifiedDate Property	Gets the DateTime object containing the date on which the resource was last modified.
Name Property	Gets the string value representing the name of the resource.
Permissions Property	Gets the byte value representing the permissions for the resource.

ItemDescriptionsResult Object

Represents an object that is returned as a result of an Upload operation and contains an array of item descriptons.

Object Properties

Name	Description
isAuthenticated Property	Gets the Boolean value indicating whether the user has been authenticated.

Query Object

Defines a query used to find reports in the Select operation.

Object Properties

Name	Description
FilterBy Property	Gets the string containing the comma-separated role values by which to filter reports.
OrderBy Property	Gets the string representing the property name to use in ordering the report descriptions returned by the Select operation.

RenderOptions Object

Defines options for how to render a report.

Object Properties

Name	Description
Extension Property	Gets the string value representing the rendering extension to use.
ReportParameters Property	Gets any **ReportParameter** objects associated with the report.
ReportId Property	Gets the string containing the identity of the report to render.
Name Property	Gets the string containing the name of the preview file to render.
ReportType Property	Gets the enumerated **ReportType** value for the report to render.

ReportDescription Object

Contains descriptive information about stored reports.

Object Properties

Name	Description
CreatedBy Property	Gets the string containing the name of the user who created the report.
CreationDate Property	Gets the DateTime object containing the date on which the report was created.
Description Property	Gets the string containing a description of the report.
IsBroken Property	Gets the Boolean value indicating whether the report is broken by recent model version changes.
IsParametrized Property	Gets the Boolean value indicating whether the report contains parameters.
MasterReportIds Property	Gets the string value representing any master report used by the report. For internal use only.
ModelId Property	Gets the string containing the identity of the model associated with the report.
ModelName Property	Gets the string containing the name of the model associated with the report.
ModelPermissions Property	Gets the enumerated **ModelPermission** value for the model associated with the report
ModelVersion Property	Gets the int value for the history version of the model associated with the report.
ModifiedBy Property	Gets the string containing the name of the user who last modified the report.
ModifiedDate Property	Gets the DateTime object containing the date on which the report was last modified.
Name Property	Gets the string value representing the name of the report.
Permissions Property	Gets the byte value representing the permissions for the report.
Id Property	Gets the string value representing the ID of the report.
ReportType Property	Gets the enumerated **ReportType** value for the report.
SubReportIds Property	Gets the string value representing any subreport used by the report. For internal use only.
ThemeId Property	Gets the int value representing the theme used by the report.

ReportParameter Object

Defines a parameter for the report.

Object Properties

Name	Description

Name Property	Gets the string containing the name of the report parameter.
Domain Property	Gets the enumerated **ReportParameterDomain** value for the report parameter.
Values Property	Gets the string representing values to supply for the report parameter.

RequestInfo Object

Defines status information for an IReportService request operation.

Object Properties

Name	Description
Exception Property	Gets the **ExceptionDetail** object containing error information for the request.
PrimaryUrl Property	Gets the URL of a primary resource for execution results.
RequestId Property	Gets the string that identifies the request.
State Property	Gets the enumerated **RequestState** value signifying the current request state.

RequestResult Object

Represents an object containing the status, or result contract, of an IReportService request operation.

Object Properties

Name	Description
Info Property	Gets the **RequestInfo** object containing status information for a request.

Result Object

Represents an object that is returned as a result of an IReportService operation.

Object Properties

Name	Description
Error Property	An Error object that represents an error that occurs during operation execution.
IsAuthenticated Property	A Boolean value that indicates whether the request has been authenticated.

UploadOptions Object

Defines options for how to upload a resource.

Object Properties

Name	Description
ConnectionString Property	Gets the string value representing the connection string to use.
Overwrite Property	Gets the Boolean value indicating whether to overwrite existing resources.

SkipResourceValidation	Gets the Boolean value indicating whether to validate that all requirements are met for using the uploaded resource to serve compiled reports.

UserCapabilities Object

Represents a user capabilities object that is contained in a result from an IReportService get user capabilities operation.

Object Properties

Name	Description
Email Property	A string value that represents the e-mail address of the user.
IsAdministrator Property	A Boolean value indicating whether the user is the administrator.
IsEvaluationVersion Property	A Boolean value indicating whether the system is in evaluation mode.
IsSBEVersion Property	A Boolean value indicating whether the system is the small business edition.
IsSchedulePermitted Property	A Boolean value indicating whether the user has permission to scheduling.
IsUploadPermitted Property	A Boolean value indicating whether the user has permission to upload reports.

UserCapabilitiesResult Object

Represents a user capabilities result object that is returned as a result of an IReportService get user capabilities operation.

Object Properties

Name	Description
Capabilities Property	A **UserCapabilities** object that contains the capabilities for the user.

ModelPermission Enumeration

Specifies the byte value of the enumerated level of permissions granted for the model associated with the report.

Enumeration Members

Member	Description
None	The indicated security token has no permission to access the data model.
Read	The indicated security token has only read access to the data model.
CreateReport	The indicated security token has both read and reporting access to the data model.

ReportParameterDomain Enumeration

Specifies the byte value of the enumerated report parameter domain.

Enumeration Members

Member	Description
SpecifiedValues = 0	Indicates that the parameter values are specified in the value collection.

SelectAll = 1	Indicates that all valid values are set.
AcceptingDynamicValues = 2	Indicates that the specified values include values to be evaluated dynamically at run time.

ReportType Enumeration

Specifies the byte value of the enumerated report type for all supported reports.

Enumeration Members

Member	Description
Unknown = 0	Indicates that the report type is unknown.
SW = 1	Indicates a semantic report, created with ActiveReports 8 Server.
AR = 2	Indicates an ActiveReports developer report.
DDR = 3	Indicates a Data Dynamics Reports developer report.

RequestState Enumeration

Specifies the state of an IReportService request operation for a RequestInfo object.

Enumeration Members

Member	Description
Accomplished	Indicates that the request operation completed successfully.
Cancelled	Indicates that the request operation was cancelled.
Pending	Indicates that the request operation is in a request queue.
Rejected	Indicates that the request operation was rejected.
Running	Indicates that the request operation execution has started, but has not yet completed.
Unavailable	Indicates that no request status information is available due to connection problems or other issues.