Reveal

Web Embedding Developer's Guide

Introduction

Welcome to the Reveal Web Embedding Developer Guide.

The goal of this document is to describe how to embed Reveal inside an external (host) web application. The SDK for embedding the Reveal web viewer includes two components: **Reveal Web Client SDK** and Reveal backend. In order to load dashboards, the Reveal Web Client needs to invoke the web services of the Reveal Server. The server handles all the heavy lifting of loading a dashboard, calling its data sources for each visualization and applying filtering, sorting and other data processing behaviors for the web client.

The Reveal Server can be embedded directly hosting the Reveal Server SDK in your ASP.NET application. Embedding the full Reveal Server is ideal for scenarios where you want to host the web client in an external application (e.g. SharePoint) and cannot directly work with the Server SDK.

Disclaimer

For the Reveal SDK preview, you will need to host the Server SDK in an ASP.NET Core461 web application. The public API of the SDK is subject to change in future releases and will not be finalized until the final release.

Chapter 1

Using the Server SDK

Included with the SDK document is a sample showing how to load dashboards, modify dashboard settings and dynamically load data using the in-memory data source. The sample is a .NET 4.6.1 application, which is configured to use the Reveal Server SDK. It is highly recommended that this sample is used as guide when using the Reveal SDK in your own custom application.

Referencing the Server SDK

Prior to using the Reveal SDK, a developer must reference the Web API assemblies and data layer components using the standard .NET DLL assembly reference tools. The files can be found under the References folder in the included sample.

- Core.WPF.dll
- DataQueryService.Model.dll
- Infragistics.EM.DataProviders.dll
- Infragistics.EM.DataQueryService.dll
- Infragistics.ReportPlus.Common.dll
- Infragistics.ReportPlus.Connector.MsSql.WPF.dll
- Infragistics.ReportPlus.Connector.MySql.WPF.dll
- Infragistics.ReportPlus.Connector.Postgres.WPF.dll
- Infragistics.ReportPlus.Connectors.Excel.WPF.dll
- Infragistics.ReportPlus.Connectors.Ssas.WPF.dll
- Infragistics.ReportPlus.Connectors.WebConnectors.WPF.dll
- Infragistics.ReportPlus.DataLayer.WPF.dll
- InfragisticsWPF4.Documents.Core.v17.2.dll
- InfragisticsWPF4.Documents.Excel.v17.2.dll
- ReportPlus.DataQuery.exe
- ReportPlus.DataQuery.Shared.dll
- Requests.WPF.dll
- ServiceFabricStandalone.dll
- System.Data.SQLite.dll

Note: For SQL Lite make sure the x64 and x86 SQLite.Interop.DLL are included when shipping your application.

Defining the Server Context

After referencing the required DLLs, you will need to create a class that implements the IRevealSdkContext interface. This interface allows the Reveal SDK to run inside of your host application and provides callbacks for working with the SDK.

```
    public class RevealSdkContext : IRevealSdkContext
    {
        public IRVDataSourceProvider DashboardDataProvider =>
                                                  new EmbedDataSourceProvider();

        public IRVDataProvider DataProvider =>
                                                  new EmbedDataProvider();

        public IRVAuthenticationProvider AuthenticationProvider =>
                                                  new EmbedAuthenticationProvider();

        public async Task<Stream> GetDashboardAsync(string dashboardId)
        {
            return LoadDashboardStream(dashboardId);
        }

        public async Task SaveDashboardAsync(string dashboardId, Stream dashboardStream)
        {
            //Save edited dashboard here
        }
    }
```

Initialize the Server SDK

In the Startup.cs ConfigureServices method of the application, call the services extension method AddRevealServices, passing in the EmbedSettings class. The AddRevealServices extension method is defined in the Reveal.DataQuery.Utility namespace, so you will need to add a using directive.

```
services.AddRevealServices(new EmbedSettings
{
    LocalFileStoragePath = @"C:\Temp\Reveal\DataSources",
}, new RevealSdkContext());
```

Note: LocalFileStoragePath is only required if you are using local Excel or CSV files as dashboard data source.

Server API

Loading RDash File

In order to view a dashboard you must supply its RDash file as stream to the SDK. Using the Reveal app, you can export a dashboard to its RDash file and then include it in your project for easier access.

```
public Stream GetDashboardStream(string dashboardId)
{
    var assembly = Assembly.GetExecutingAssembly();
    var resourceName = $"ReportPlus.Embed.Sample.Dashboards.{dashboardId}.rdash";
    return assembly.GetManifestResourceStream(resourceName);
}
```

Change data sources and In-Memory data support

Prior to Reveal Server SDK loading and processing the data for a dashboard, you can override the configuration or data to use for the specified visualization.

```
public RVDataSourceItem ChangeVisualizationDataSourceItem(string dashboardId,
RVVisualization visualization, RVDataSourceItem dataSourceItem)
{
    var sqlServerDsi = dataSourceItem as RVSqlServerDataSourceItem;
    if (sqlServerDsi != null)
    {
        // Change SQL Server host url
        var sqlServerDS = (RVSqlServerDataSource)sqlServerDsi.DataSource;
        sqlServerDS.Host = "http://10.0.0.20";

        // Change SQL Server database and table/view
        sqlServerDsi.Database = "Adventure Works";
        sqlServerDsi.Table = "Employees";

        return sqlServerDsi;
    }

    // Fully replace a data source item with a new one
    if (visualization.Title == "")
    {
        var sqlDs = new RVSqlServerDataSource();
        sqlDs.Host = "rpluste01";
        var sqlDsi = new RVSqlServerDataSourceItem(sqlDs);
        sqlServerDsi.Table = "Customers";
```

```
            return sqlServerDsi;
        }

    // provide in-memory data
    if (globalFilter.Title == "application_name" && dashboardId.EndsWith("InMemDF"))
    {
        return await Task.Run(() => new RVInMemoryDataSourceItem("application_name"));
    }
    else
    {
        return await Task.Run(() => dataSourceItem);
     }
```

IRVDataProvider's implementation provides the actual data when needed based on the datasetId passed through the constructor of RVInmemoryDataSourceItem

```
public class EmbedDataProvider : IRVDataProvider
{
    public IRVInMemoryData GetData(RVDataSourceItem dataSourceItem)
    {
        if (datasetId == "application_name")
        {
            var data = new List<ApplicationName>()
                {
                    new ApplicationName(){ application_name = "App3"},
                    new ApplicationName(){ application_name = "App4"},
                    new ApplicationName(){ application_name = "App5"}
                };
            return new RVInMemoryData<ApplicationName>(data);
        }
    }
}
```

Providing credentials to data sources

For dashboard data sources such as SQL Server or OAuth the Server SDK provides a way to pass in the credentials to use when accessing the data source.

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
    public Task<NetworkCredential> ResolveCredentials(RVDashboardDataSource dataSource)
    {
        NetworkCredential userCredential = null;
        if (dataSource is RVPostgresDataSource)
        {
            userCredential = new NetworkCredential("UserName", "Password");
        }

        return Task.FromResult(userCredential);
    }
}
```

Chapter 2

Using the Web Client SDK

Dependencies

The Reveal Web Client SDK has the following 3rd party references.

- JQuery 2.2 or greater
- Google Maps (Required for dashboard with map) – Developer must provide key

Referencing the Web Client SDK

Enabling Reveal Embed View in a Web page requires several scripts to be included. These scripts will be provided as part of Reveal Embed.

```
<script src="~/Reveal/infragistics.util.js"></script>
<script src="~/Reveal/infragistics.corecp.js"></script>
<script src="~/Reveal/infragistics.ui.js"></script>
<script src="~/Reveal/infragistics.requests.js"></script>
<script src="~/Reveal/infragistics.requests.strings.js"></script>
<script src="~/Reveal/infragistics.em.js"></script>
<script src="~/Reveal/infragistics.em.strings.js"></script>
<script src="~/Reveal/infragistics.datalayer.js"></script>
<script src="~/Reveal/infragistics.datalayer.strings.js"></script>
<script src="~/Reveal/infragistics.chart.js"></script>
<script src="~/Reveal/infragistics.rpui.js"></script>
<script src="~/Reveal/infragistics.rpembedded.js"></script>
```

Instantiate the Web Client SDK

Reveal Dashboard presentation is handled natively through the Web Client SDK. To get started define a <div /> with “id” and invoke the RevealView constructor. Create an instance of RevealSettings providing the dashboardId in the constructor. Then call RevealUtility’s LoadDashboard providing the dahsboardId and success and error handlers. In the success handler you should use the retrieved dashboard and set it to the dashboard property of the RevealSettings object. Finally you instantiates the RevealView component by passing a selector for the DOM element where the dashboard should be rendered into as well as the settings object.

```
<!DOCTYPE html>
<html>
<head>
    ⋮
    <script type="text/javascript">
        var dashboardId = "dashboardId";
        var revealSettings = new RevealSettings(dashboardId);

        RevealUtility.LoadDashboard(dashboardId, function (dashboard) {
            revealSettings.dashboard = dashboard;
            var revealView = new RevealView("#revealView", revealSettings);
        }, function (error) {
         //Process any error that might occur here
        });
    </script>
</head>
<body>
    <div id="revealView"/>
</body>
</html>
</html>
```

Using RevealSettings

RevealSettings object is used to enable/disable some features for the end user. For example, its showFilters property is read by the reveal view on initialization time and based on its value shows or hide the global filters UI. Other settings are showExportButton, canEdit, maximizedVisualization, etc.

Reveal settings also have a dashboard property, which is the way to specify which dashboard should be rendered. As shown on the snippet above the dashboard must be retrieved by using the RevealUtility.LoadDahsboard method which receives a dashboardId, and success callback which will be called when the dashboard is loaded.

Another capability exposed through the settings object is the way to specify what values are selected for the global filters in the dashboard. The following snippet shows loading a dashboard “AppsStats” and sets the “application_name” selected value to be “App2” thus the dashboard will be showing data about “App2”

```
var dashboardId = "AppsStats";
var revealSettings = new RevealSettings(dashboardId);

RevealUtility.LoadDashboard(dashboardId, function (dashboard) {
    revealSettings.dashboard = dashboard;

    var applicationNameFitler = dashboard.getFilterByTitle("application_name");
    revealSettings.setFilterSelectedValues(applicationNameFitler, ["App2"]);
    window.revealView = new RevealView("#revealView", revealSettings);
}, function (error) {
});
```

Note: The RevielView picks up RevealSettings on inti time. Changing the settings object after the dashboard was already rendered will not affect the already loaded dashboard. If it is needed a setting to be changed the new instance of RevealView needs to be instantiated.