Welcome to Reveal SDK

Here you will find technical information on the Reveal SDK and API methods, including architecture information.

Where to start:

Reveal SDK Overview. Introduction to the SDK main concepts and features.

Installation and Requirements. How to install and what's required.

Desktop SDK Overview. Introduction to the Desktop SDK main concepts and architecture.

- Setup and configuration. Steps required to get the Desktop SDK ready to be used.

WEB .NET SDK Overview. Introduction to the Web .NET SDK main concepts and architecture.

- Setup and configuration. Steps required to get the Web .NET SDK ready to be used.
- Creating your First App. Walkthrough that guides you through the initial steps of showing a dashboard on your web page/application for the first time.

WEB JAVA SDK Overview. Introduction to the Web JAVA SDK main concepts and architecture.

- Setup and configuration. Steps required to get the Web JAVA SDK ready to be used.
- Running the UpMedia Samples. Walkthroughs that guide you run the UpMedia samples provided.

Reveal SDK Overview

- **Desktop SDK** - Reveal Desktop SDK allows you to embed Reveal inside an external (host) Windows application (WPF or WinForms).
- **Web SDK** - Reveal Web SDK allows you to embed Reveal inside an external (host) web application. The SDK for embedding the Reveal web viewer includes two components:
    - Reveal Web Client SDK,
    - Reveal Web Server SDK, supported in two different platforms (.NET and JAVA)

When installing Reveal's SDK, you install the .NET Web SDK and Desktop SDK at the same time. The JAVA SDK is distributed as a set of Maven modules.

Main Features

With Reveal SDK, developers can embed Reveal into their applications. And dashboards can be displayed and even modified by end users.

Reveal SDK can be used to integrate Reveal into applications developed in multiple platforms and technologies: Web, Windows WPF, and Windows Forms.

The containing app can use Reveal SDK to:

- Provide in-memory data to dashboards. If data is already loaded in the containing app there’s no need to store it before opening the dashboard, Reveal can use the built-in InMemory data provider to use that data as the input for the dashboard.
- Configure the dashboard before it gets rendered, for example changing the name of the database or table to use to get the data based on the current user.
- Change dashboard filters before the dashboard gets rendered or even while it’s visible. The containing app can use this feature to synchronize filters or selections in the app with the data visualized in the dashboard.
- Get notified when a data point is selected in the dashboard (like a bar in a chart is clicked), for example to display additional information or trigger an action in the app like navigating to a page with more details.
- And many other features...

System Requirements and Installation

Desktop SDK Requirements

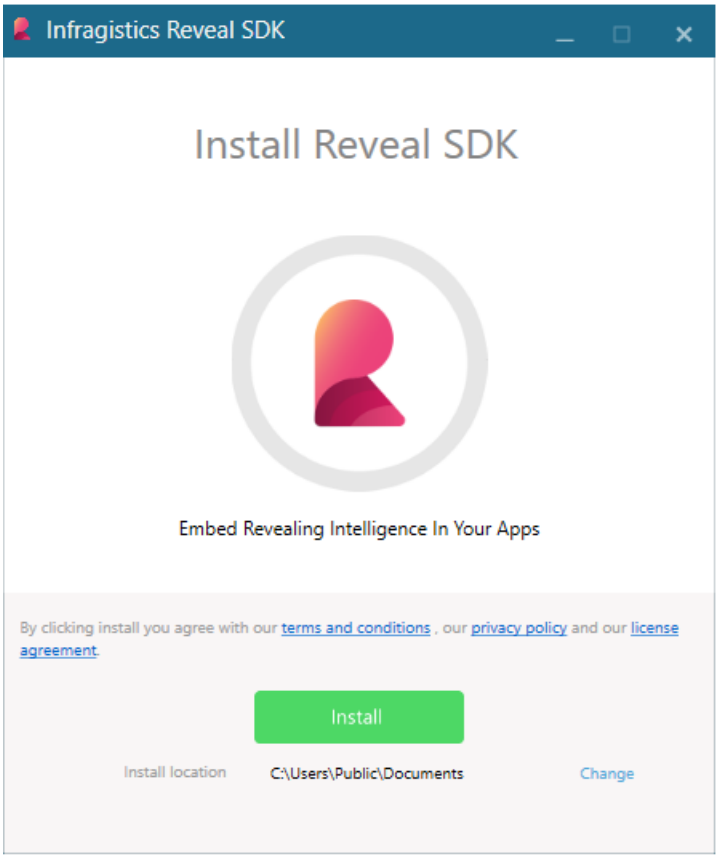
- The Reveal SDK requires .NET version 4.6.2+ and Visual Studio 2015 or up.

Web SDK .NET Requirements

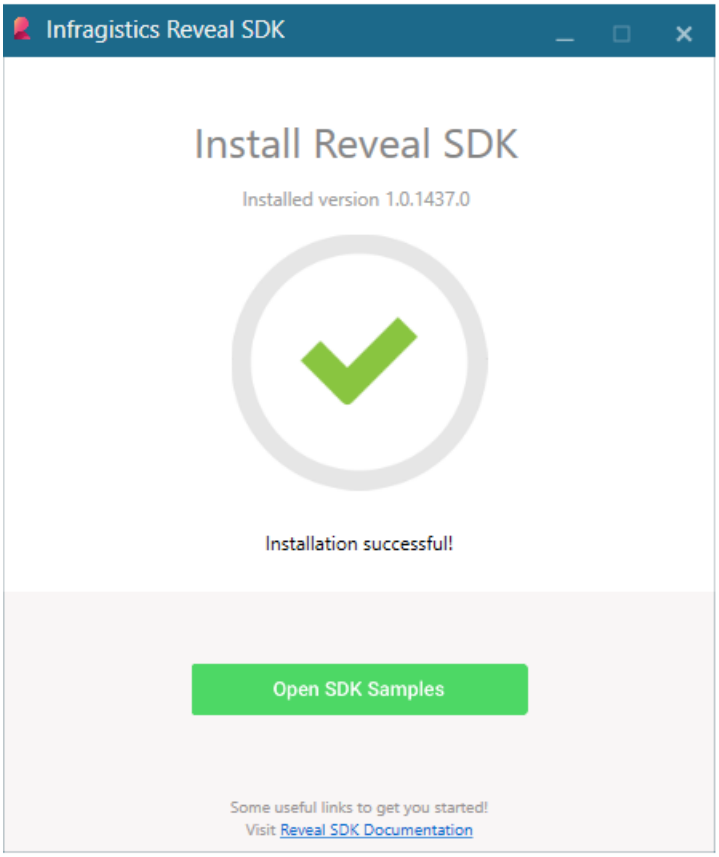
- The Reveal Server SDK requires .NET Core 2.2+ server-side projects targeting .NET framework 4.6.2+.

Installing Desktop and Web .NET SDK

To get the Reveal SDK for both Web and Desktop .NET platforms, sign up here. Once ready, follow through the provided installer:



After a successful installation, you can browse the installed samples by clicking the *Open SDK Samples* link.



Samples

In case you missed the samples link, you can find them in "%public%\Documents\Infragistics\Reveal\SDK\".

In this location you will find a solution file (Reveal.Sdk.Samples.sln). This project combines all Web, WPF, and WinForms samples.

For Web you need to restore the node packages in order to run the samples with IIS and change the StartUp project. To restore, just right click the solution in the Solution Explorer and select Restore packages.

Web SDK JAVA Requirements

- Java SDK 11.0.10 and up recommended.
- Maven 3.6.3 and up recommended

Installing JAVA SDK

Reveal Java SDK is distributed as a set of Maven modules. To work with the SDK libraries, you need to add a reference to Reveal's Maven Repository and also a dependency in your Maven pom.xml file. For further details, please refer to Setup and Configuration.

Samples

The **UpMedia samples** illustrate how to use the JAVA SDK, you can get them from GitHub here.

For details about how to run the UpMedia samples, please follow this link.

Getting Dashboards for the SDK

Introduction

Reveal is a business intelligence platform that was purposely designed to be embedded into applications. Embedded Reveal dashboards are a quick and simple way to display information to communicate the status, metrics, or performance of a business.

Let's take a look at the differences between Reveal SDK and the Reveal app.

The **Reveal Application** is a self-service business intelligence tool that enables you to make data-driven decisions faster. You can create, view and share dashboards in your workspaces. And it offers you an identical experience no matter what platform you are on: Web, Desktop, iOS, or Android. For further details about the Reveal app, you can access an **online demo** or browse the **Help Documentation**.

With **Reveal SDK** developers can embed Reveal into their applications developed in multiple platforms and technologies. And Reveal dashboards can be displayed and even modified by end users.

Overview

In order to display dashboards in your application you first need to create them using the Reveal Application on your preferred platform.

You can also use the Reveal SDK to create dashboards from scratch, but the recommended approach when you are first evaluating the SDK is to start with dashboards created with the app.

Getting your Dashboard File

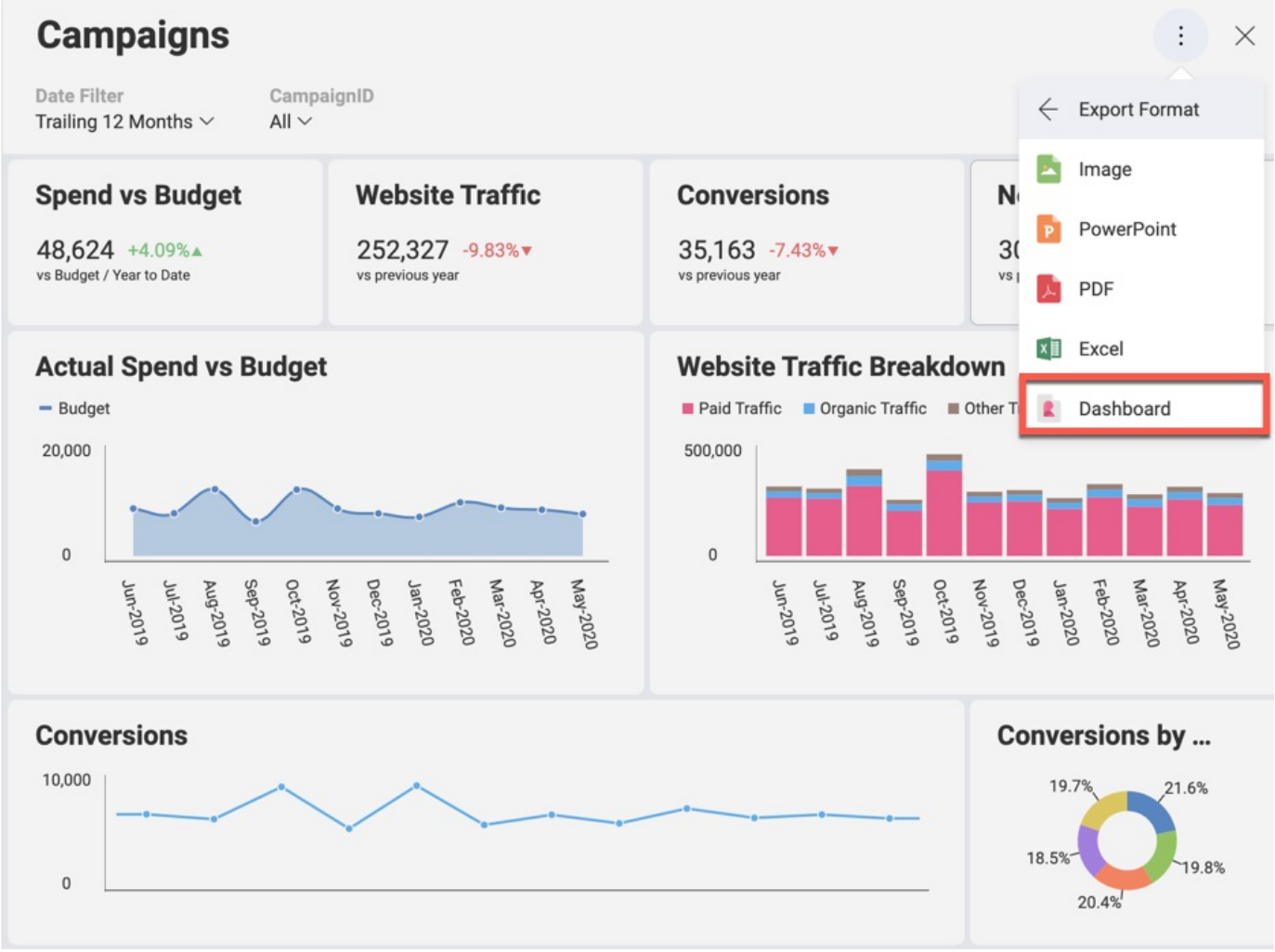
Once you **created your dashboard**, below you can find how to get it:

1. **Open your Dashboard in the Reveal app**

   After installing the Reveal Application in any platform, you can either create your own dashboard or use one of the sample dashboards provided with the app.

2. **Access the Export Options**

   Go to the overflow menu, select *Export*, then *Dashboard*, this will generate a file with extension ".rdash" that you will use later in your application when integrating the SDK.



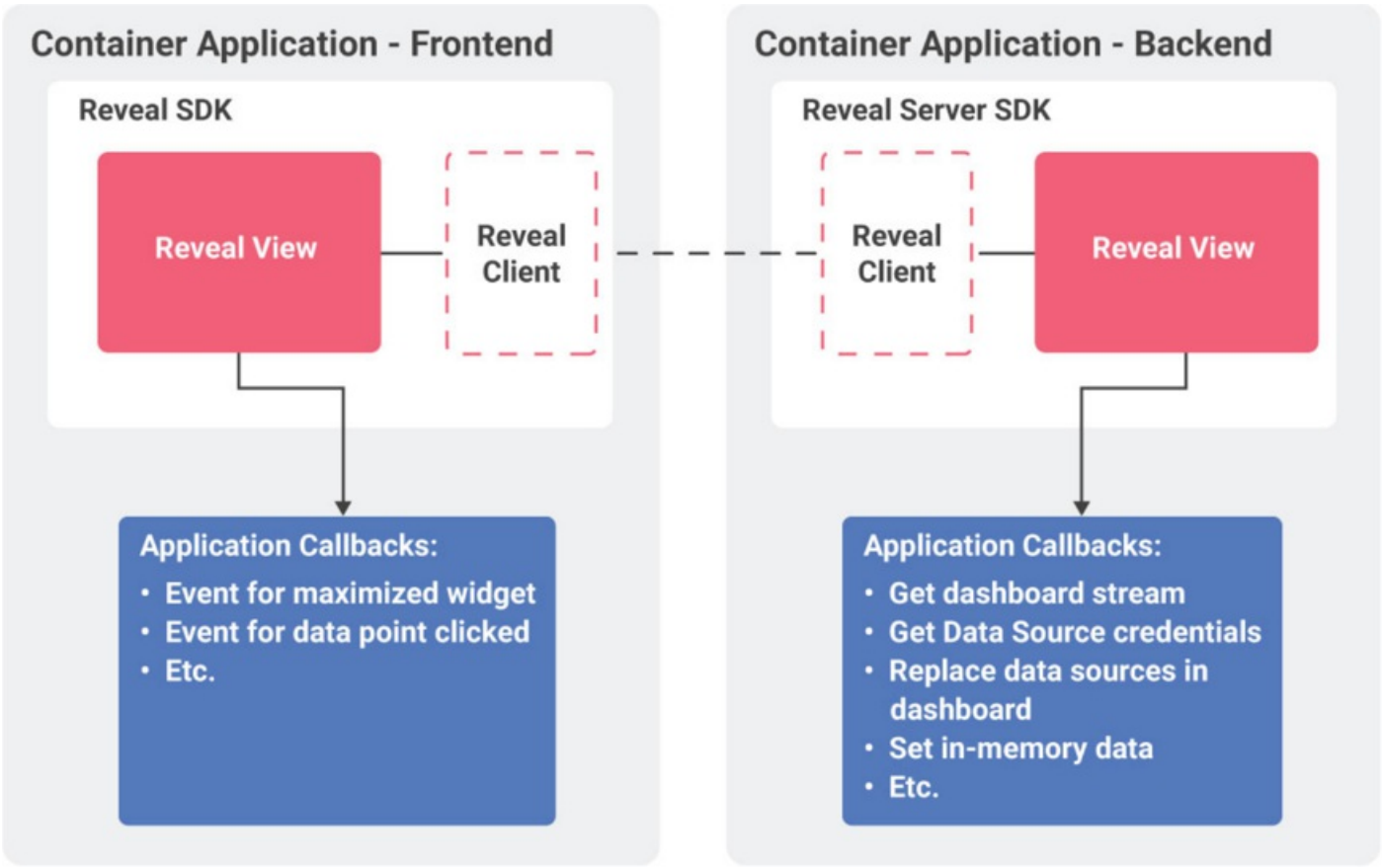
You can export the dashboard file through email (Android and iOS), or as a .rdash file on your computer (Desktop and Web).

Overview

When embedding Reveal into web applications, the architecture is slightly more complex than with native apps, as two components are always involved:

- **Reveal Client SDK**: a set of JavaScript libraries that needs to be integrated into the web application. The frameworks supported today are: jQuery, Angular and React.
- **Reveal Server SDK**: the server-side component to be integrated into the server application, currently this is an ASP.NET Core application targeting .NET Runtime (v4.6.2 or later) and .NET Core (2.2, 3.1, and 5.0).

In the following diagram you visualize the architecture for a web application embedding Reveal Web SDK:



As shown above, the SDK works pretty much the same way as with native apps. The difference is that some of the callbacks are invoked in the client side (like the event sent when a data point is clicked) and others are invoked server side (like the callback to load the dashboard or to provide in-memory data).

Hosting the Client-side and Server-Side Parts on Different Servers

You can host the client-side and the server-side parts separately i.e. on different urls.

To achieve this, set a property on the window object, as shown below:

```
$.ig.RevealSdkSettings.setBaseUrl("{back-end base url}");
```

Please, note that the **trailing slash symbol is required in the URL** in order to set the property successfully.

Set this property **prior to the** *instantiation of the $.ig.RevealView*.

Setup and Configuration

Prerequisites

The Reveal Server SDK requires .NET Core 2.2+ or .NET Framework 4.6.2 ASP MVC application projects.

In case you are targeting .NET Framework 4.6.2+, the Reveal Server SDK supports a win7-x64 runtime environment. To debug your web project you need to add a win7-x64 compatible *RuntimeIdentifier* platform:

```
<PropertyGroup>
  <TargetFramework>net462</TargetFramework>
  <RuntimeIdentifier>win7-x64</RuntimeIdentifier>
</PropertyGroup>
```

Setup and Configuration (Server)

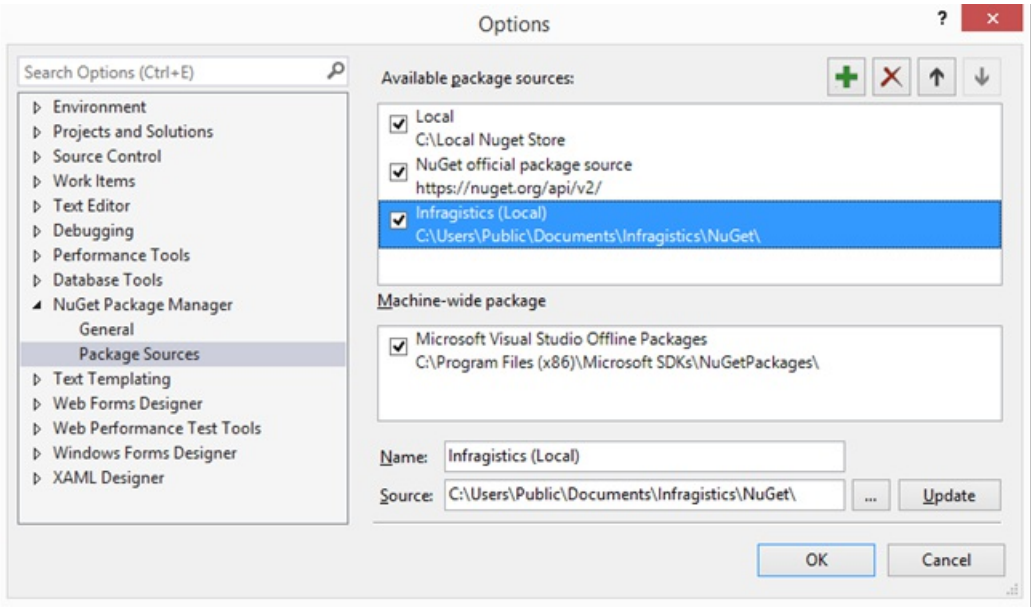
To set up the Reveal Web Server SDK you need to:

1. **Add references to assemblies and install dependency packages.**
2. **Define the Server Context.**
3. **Initialize the Server SDK.**
4. **Enable server-side screenshot generation**.

1. Getting Assemblies and Dependency Packages ready

To add references to assemblies and install dependency packages we recommend using **NuGet** package manager. The easiest way to setup your project is installing **Reveal.Sdk.Web.AspNetCore** (Trial) NuGet package.

After installing the Reveal SDK, you should be able to find a new NuGet package source added to your **nuget.config** called *Infragistics (Local)* that points to "%public%\Documents\Infragistics\NuGet".



After ensuring you have the Infragistics (Local) feed properly configured by the installer, you need to:

- install the **Reveal.Sdk.Web.AspNetCore** NuGet package to your application project.

- add a NuGet package reference to System.Data.SQLite version 1.0.111+

If you are having issues with the build, follow this **link**.

2. Defining the Server Context

After referencing the required DLLs, you need to create a class that inherits the **RevealSdkContextBase** abstract class. This class allows the Reveal SDK to run inside of your host application and provides callbacks for working with the SDK.

```
using Reveal.Sdk;
public class RevealSdkContext : RevealSdkContextBase
{
  public override IRVDataSourceProvider DataSourceProvider => null;

  public override IRVDataProvider DataProvider => null;

  public override IRVAuthenticationProvider AuthenticationProvider => null;

  public override Task<Dashboard> GetDashboardAsync(string dashboardId)
  {
    var fileName = $"C:\\Temp\\{dashboardId}.rdash";
    var fileStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    return Task.FromResult(new Dashboard(fileStream));
  }

  //This callback is used only when “onSave” event is not installed on the
  //RevealView object client side. For more information see the web client SDK documentation
  public override Task SaveDashboardAsync(string userId, string dashboardId, Dashboard dashboard)
  {
    return Task.CompletedTask;
  }
}
```

The implementation above will load dashboards from “C:\Temp” folder, looking for a .rdash file that depends on the *dashboardId* variable. In your application, you may want to change this to load dashboards from another directory, from the database, or even from an embedded resource.

Note

> **Properties returning null:** The first three properties, *DataSourceProvider*, *DataProvider*, and *AuthenticationProvider*, are all implemented to return null. In this guide you can find information about how to implement each of the interfaces for these properties, so they will no longer be implemented to return null.

3. Initializing the Server SDK

In the **Startup.cs**, in the **ConfigureServices** method of the application, call the services extension method *AddRevealServices*, passing in the *RevealEmbedSettings* class.

The *AddRevealServices* extension method is defined in the **Reveal.Sdk** namespace, so you will need to add a using directive. In addition, you also need to set the **CachePath** property as shown below.

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

Note

> **LocalFileStoragePath** is only required if you are using local Excel or CSV files as dashboard data source, and the *RevealSdkContext* class inherits *RevealSdkContextBase* as described above.

Finally, you need to add Reveal endpoints by calling the **AddReveal** extension method when adding MVC service. Similar to the following code snippet:

```
services.AddMvc().AddReveal();
```

Like *AddRevealServices*, the *AddReveal* method is defined in the *Reveal.Sdk* namespace, so you need a using directive too.

4. Enabling server-side screenshot generation

In order to use the **export to image** functionality (either programmatically or through user interaction), you need to perform the steps below:

1. Get the following three files from **<InstallationDirectory>\SDK\Web\JS\Server**:
    - package.json
    - packages-lock.json
    - screenshoteer.js
2. Copy the files to the root level of your project (parent folder of "wwwroot").
3. Make sure you have **npm** (the package manager for Node.js) installed.

If **you don't need the export to image** functionality, you don't need to copy the files to your projects. However, when trying to build the project, it will fail complaining that it cannot find **npm**.

To solve this error, add the following property to your project:

```
<PropertyGroup>
  <DisableRevealExportToImage>true</DisableRevealExportToImage>
</PropertyGroup>
```

Build Issues using NuGet

To handle a deployment issue related to **SQLite.Interop.dll**, custom .targets file are used in the NuGet package.

If you are having build issues, you can disable this behavior by adding the following property to your project:

```
<DisableSQLiteInteropFix>true</DisableSQLiteInteropFix>
```

Setup and Configuration (Client)

To set up the Reveal Web Client SDK you need to:

1. **Check Dependencies**.
2. **Reference the Web Client SDK**.
3. **Instantiate the Web Client SDK**.

1. Checking Dependencies

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

- jQuery 2.2 or greater
- Day.js 1.8.15 or greater
- Quill RTE 1.3.6 or greater
- Marker Clusterer v3 or greater
- Google Maps v3 or greater

2. Referencing the Web Client SDK

Enabling **$.ig.RevealView** component in a web page requires several scripts to be included. These scripts will be provided as part of Reveal Web Client SDK.

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

JavaScript files can be found in "<InstallationDirectory>\SDK\Web\JS\Client".

3. Instantiating the Web Client SDK

Reveal's Dashboard presentation is handled natively through the Web Client SDK.

To get started follow these steps:

1. Define a element with "id" and invoke the **$.ig.RevealView** constructor.

   Note

   > **Hosting Client-Side and Server-Side Parts Separately** If you want to host client-side and server-side parts on different servers, please read here **before** you continue to next step.

2. Call **$.ig.RVDashboard.loadDashboard** providing the *dashboardId* and success and error handlers.

3. In the success handler instantiate the **$.ig.RevealView** component by passing a selector for the DOM element where the dashboard should be rendered into. Finally you should use the retrieved dashboard and set it to the dashboard property of the **$.ig.RevealView**

Sample Code

```
<!DOCTYPE html>
<html>
  <head>
    ⋮
   <script type="text/javascript">
     var dashboardId = "dashboardId";

     $.ig.RVDashboard.loadDashboard(
       dashboardId,
       function (dashboard) {
         var revealView = new $.ig.RevealView("#revealView");
         revealView.dashboard = dashboard;
       },
       function (error) {
         //Process any error that might occur here
       }
     );
   </script>
  </head>
  <body>
   <div id="revealView" style="height:500px;" />
  </body>
</html>
```

Creating Your First App

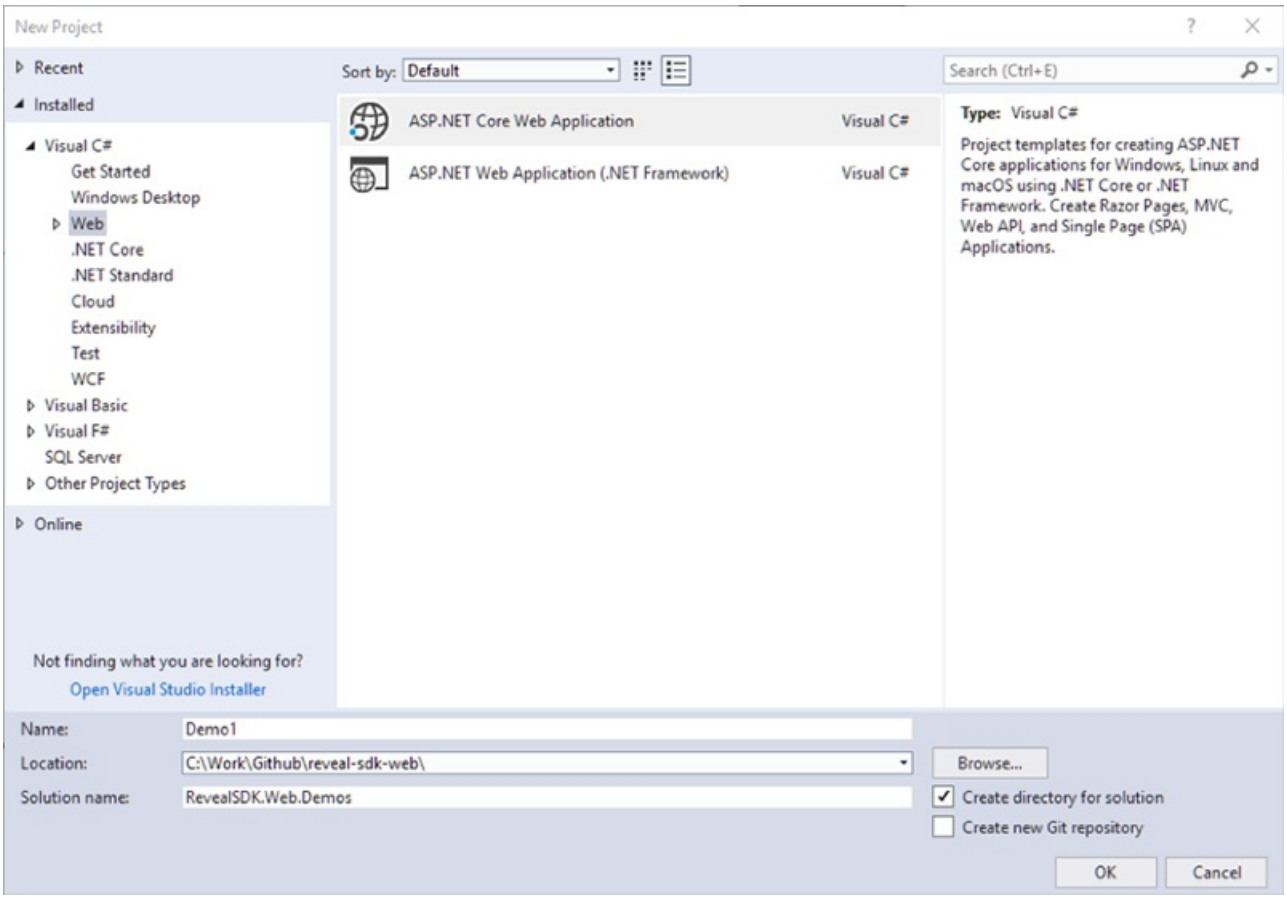
This part aims to guide you through the initial steps of showing a dashboard on your web page/application for the first time.

Steps

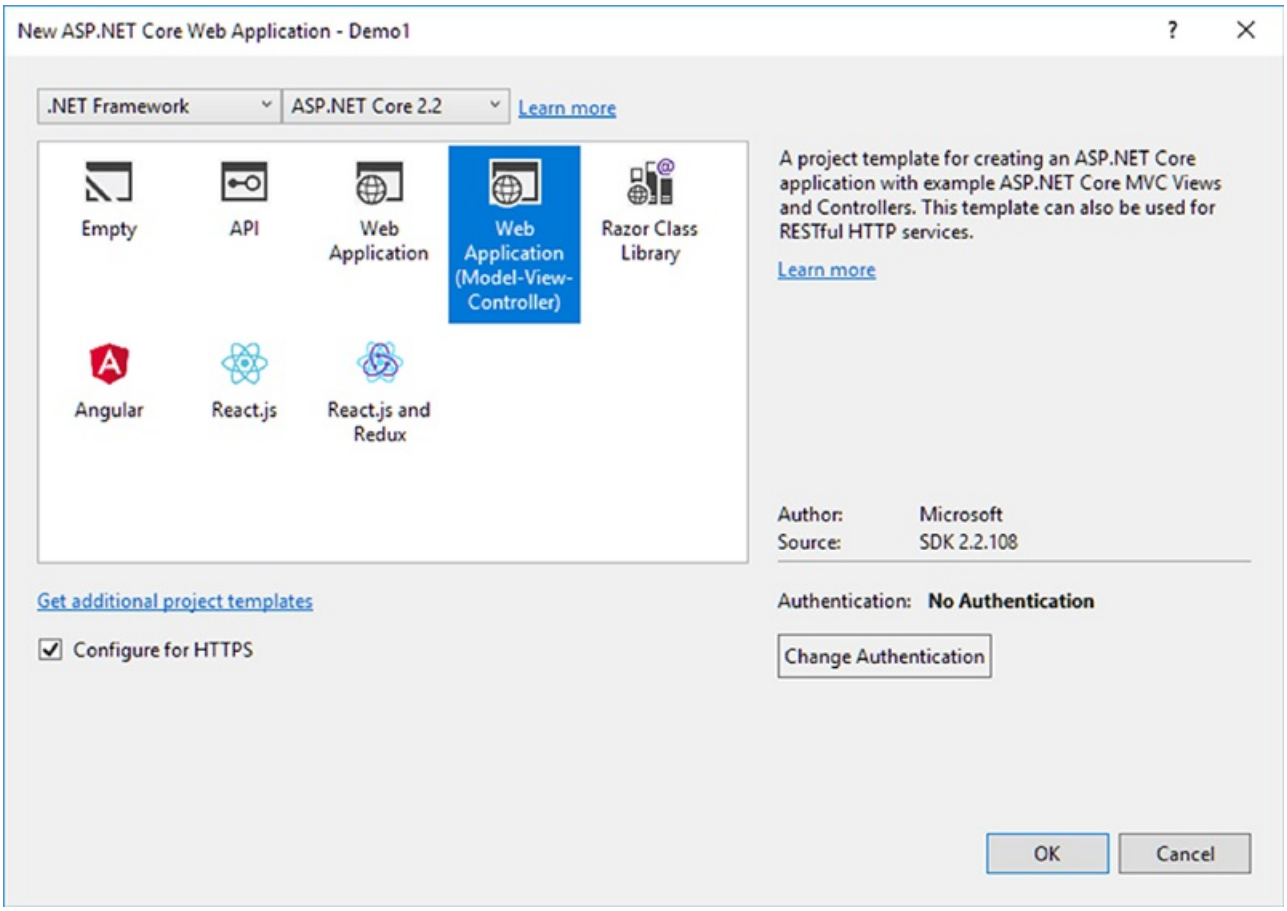
1. Creating the Project
2. Installing Reveal SDK
3. Working on Server configuration
4. Embedding Reveal in your Client Application
5. Using Reveal Fonts
6. Styling the Client Application

Step 1 - Create the Project

Open Visual Studio 2017 and create new project of type **ASP.NET Core Web Application**:

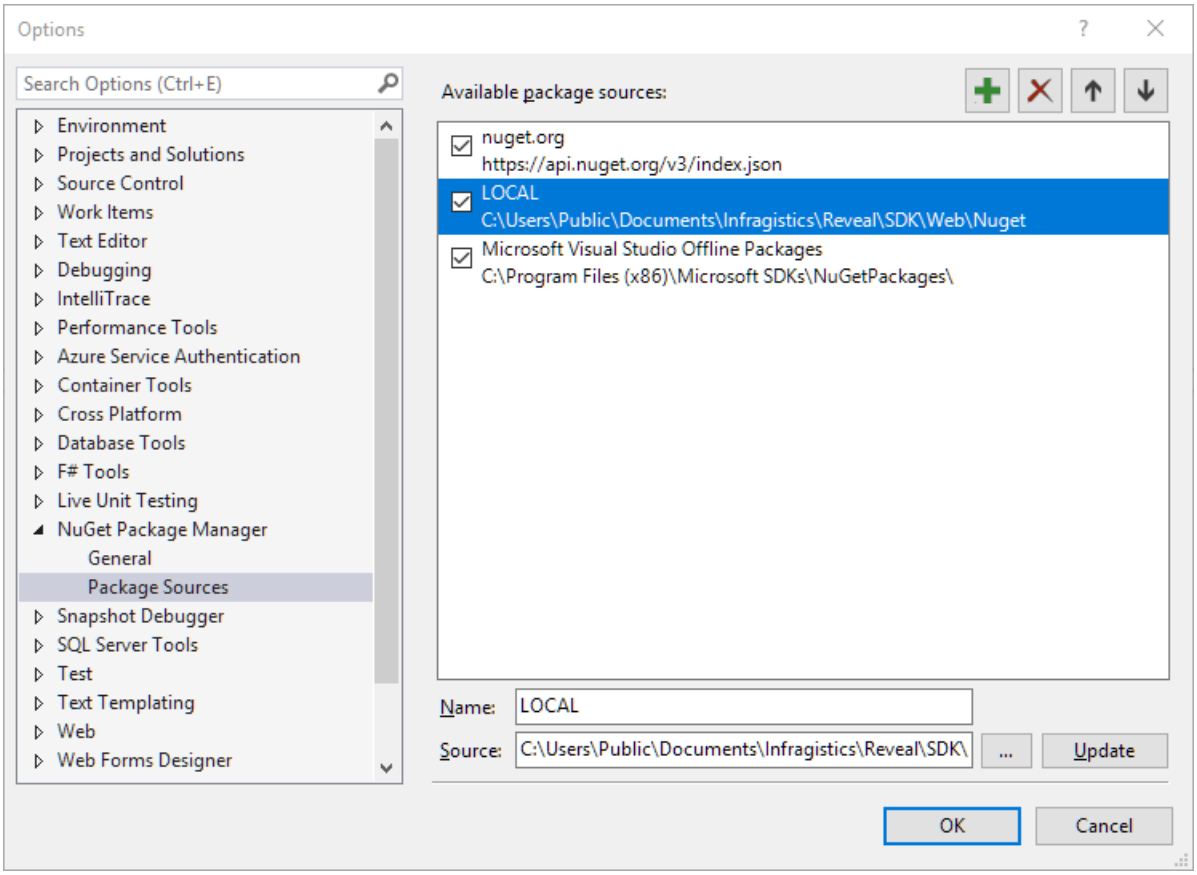


Select **.NET Framework** and **ASP.NET Core 2.2** as follows:

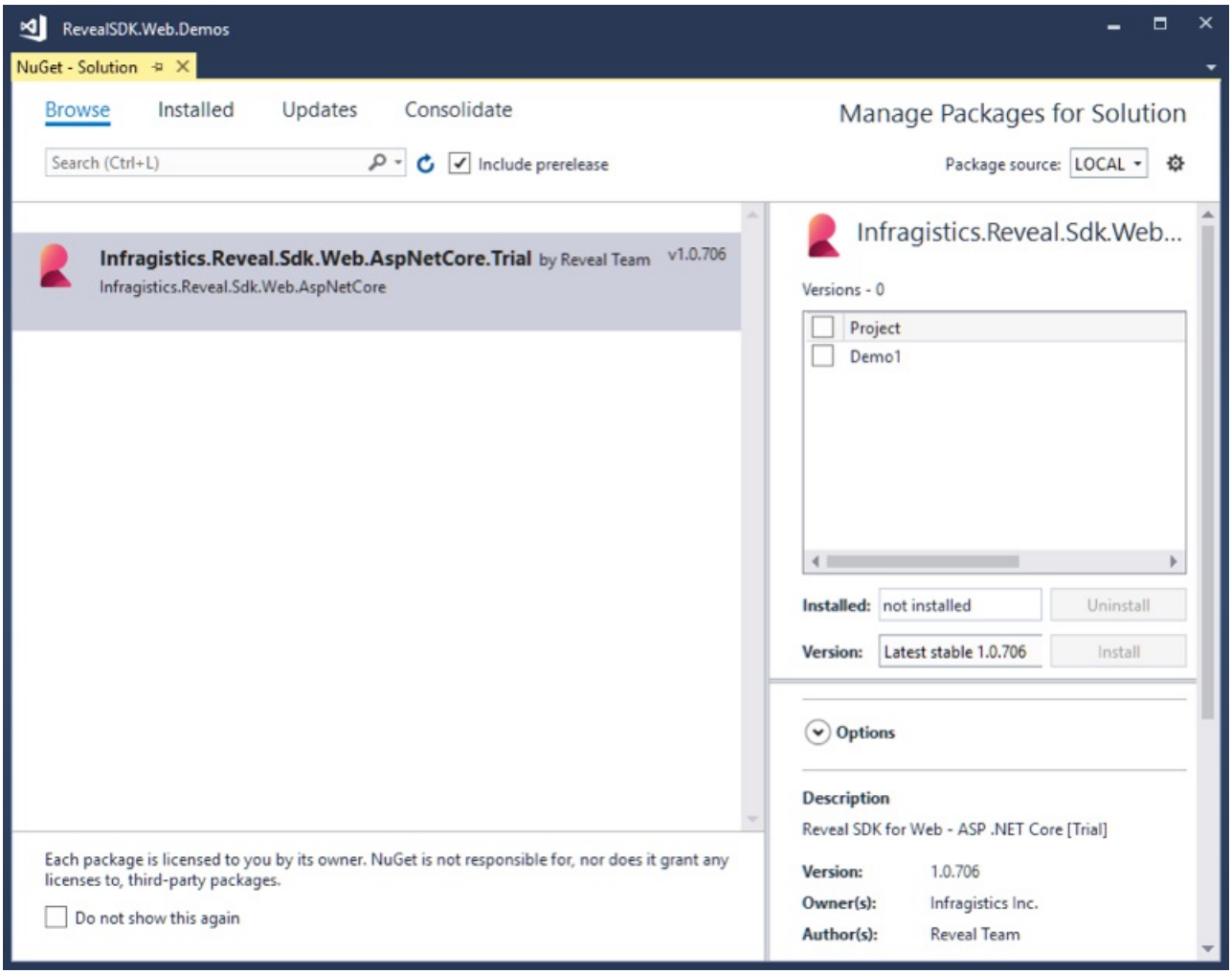


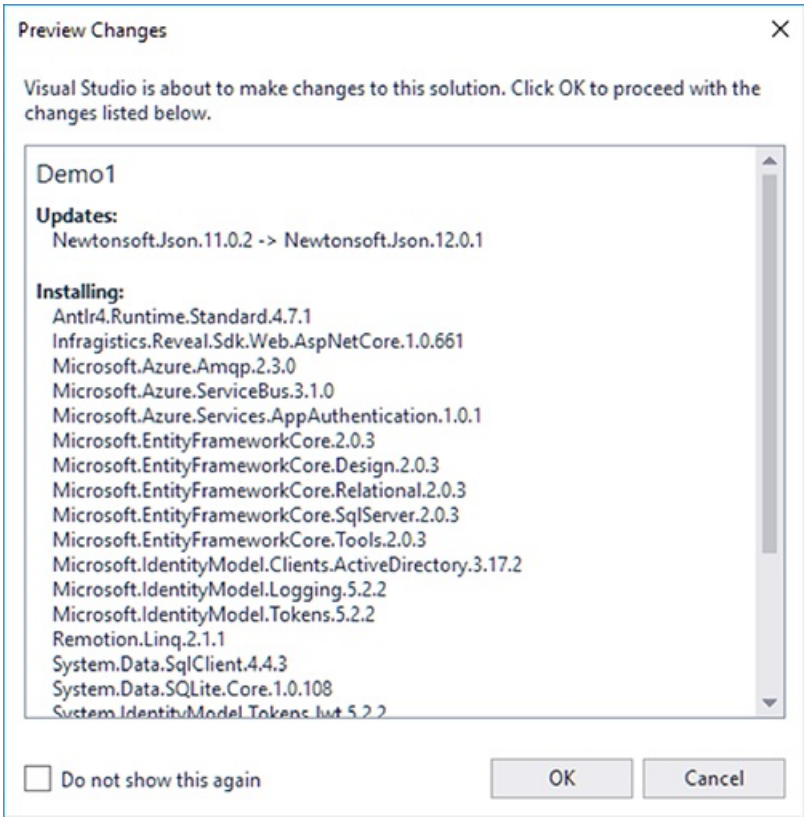
Step 2 - Install Reveal SDK

Download the *Infragistics Reveal SDK* from https://www.revealbi.io/ and install it on your machine. In **Visual Studio** go to **Tools > Options > Nuget Package Manager > Package Sources**. Add a new source pointing to the Nuget folder of the installed SDK:



After that you can install the Nuget by changing the package source to the one you added:





Step 3 - Work on Server Configuration

Create a new *Reveal SDK* folder in the project and add the **RevealSdkContext.cs** class, which implements the **RevealSdkContextBase** abstract class:

```
using Reveal.Sdk;
using System;
using System.IO;
using System.Reflection;
using System.Threading.Tasks;

namespace Demo1.RevealSDK
{
    public class RevealSdkContext : RevealSdkContextBase
    {
        public override IRVDataSourceProvider DataSourceProvider => null;

        public override IRVDataProvider DataProvider => null;

        public override IRVAuthenticationProvider AuthenticationProvider => null;

        public override Task<Dashboard> GetDashboardAsync(string dashboardId)
        {
            var dashboardFileName = dashboardId +".rdash";
            var resourceName = $"Demo1.Dashboards.{dashboardFileName}";
            var assembly = Assembly.GetExecutingAssembly();
            return Task.FromResult(new Dashboard(assembly.GetManifestResourceStream(resourceName)));
        }

        public override Task SaveDashboardAsync(string userId, string dashboardId, Dashboard dashboard)
        {
            return Task.CompletedTask;
        }

    }
}
```

In the code above **Demo1.Dashboards** indicates the location where the dashboard files will be contained, so let's create a new Dashboards folder in the project and leave it empty for now.

To do this, add the following code to **ConfigureServices** method in **Startup.cs**:

```
services.AddRevealServices(new RevealEmbedSettings
{
    CachePath = @"C:\Temp"
}, new RevealSdkContext());

services.AddMvc().AddReveal();
```

And the necessary references in the same file:

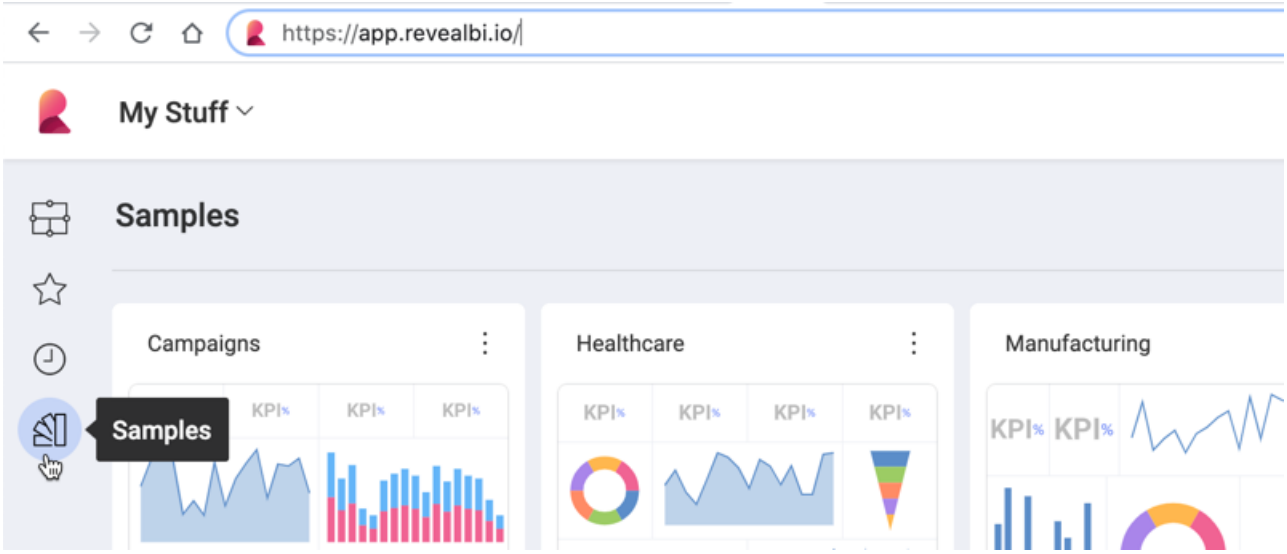
```
using Demo1.RevealSDK;
using Reveal.Sdk;
```

If you experience any issues, please refer to the **Setup and Configuration (Web)** topic.

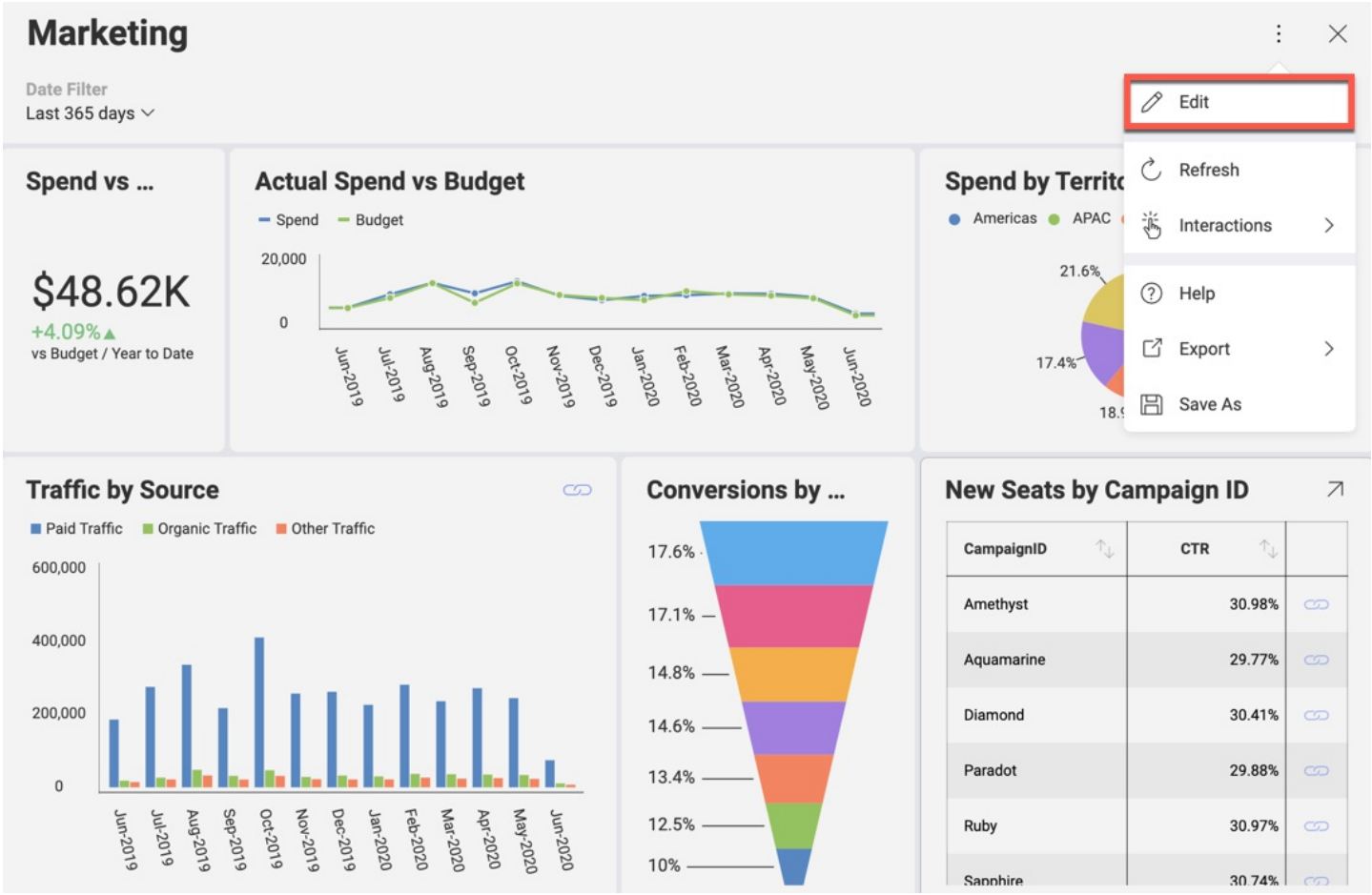
Step 4 - Embed Reveal in your Client application

Let's start this step by getting a dashboard ready. For the purpose of this demo, you can use the **Marketing dashboard** from the **Samples** section in Reveal, but with a different theme.

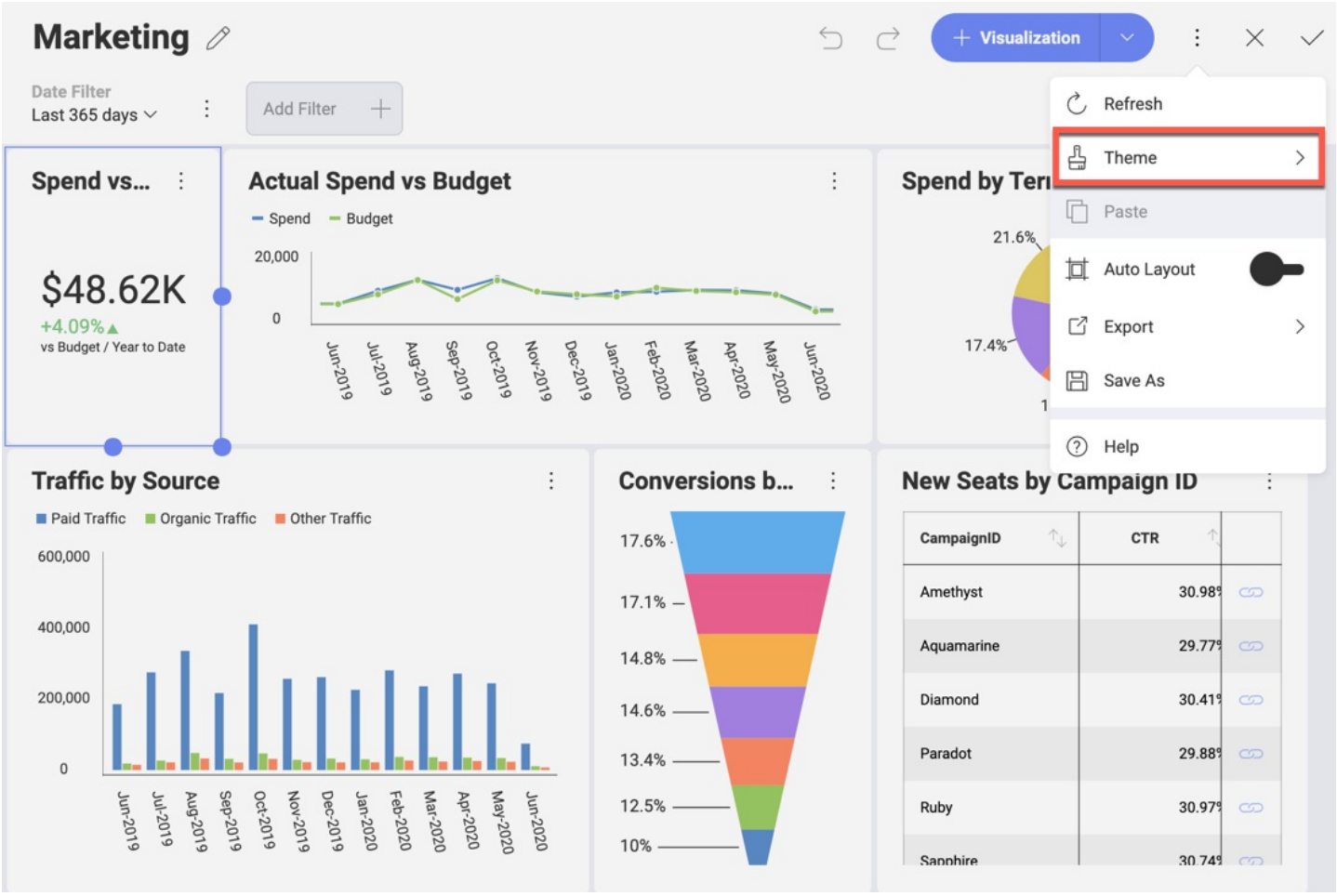
Open the Reveal app (https://app.revealbi.io) and go to the **Samples**.



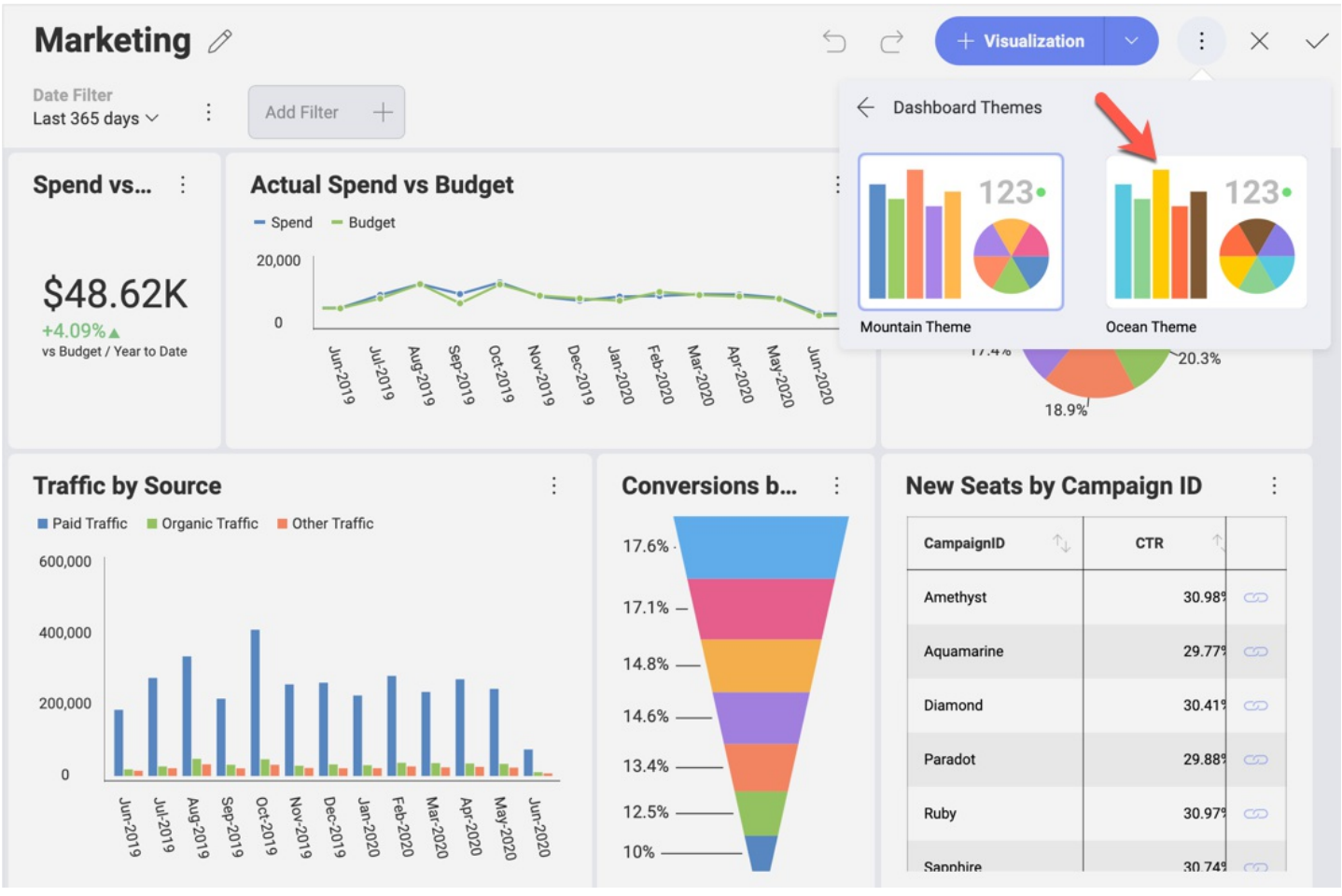
Select the Marketing dashboard and enter **edit mode**:



Once in Edit mode click the *Theme* button:



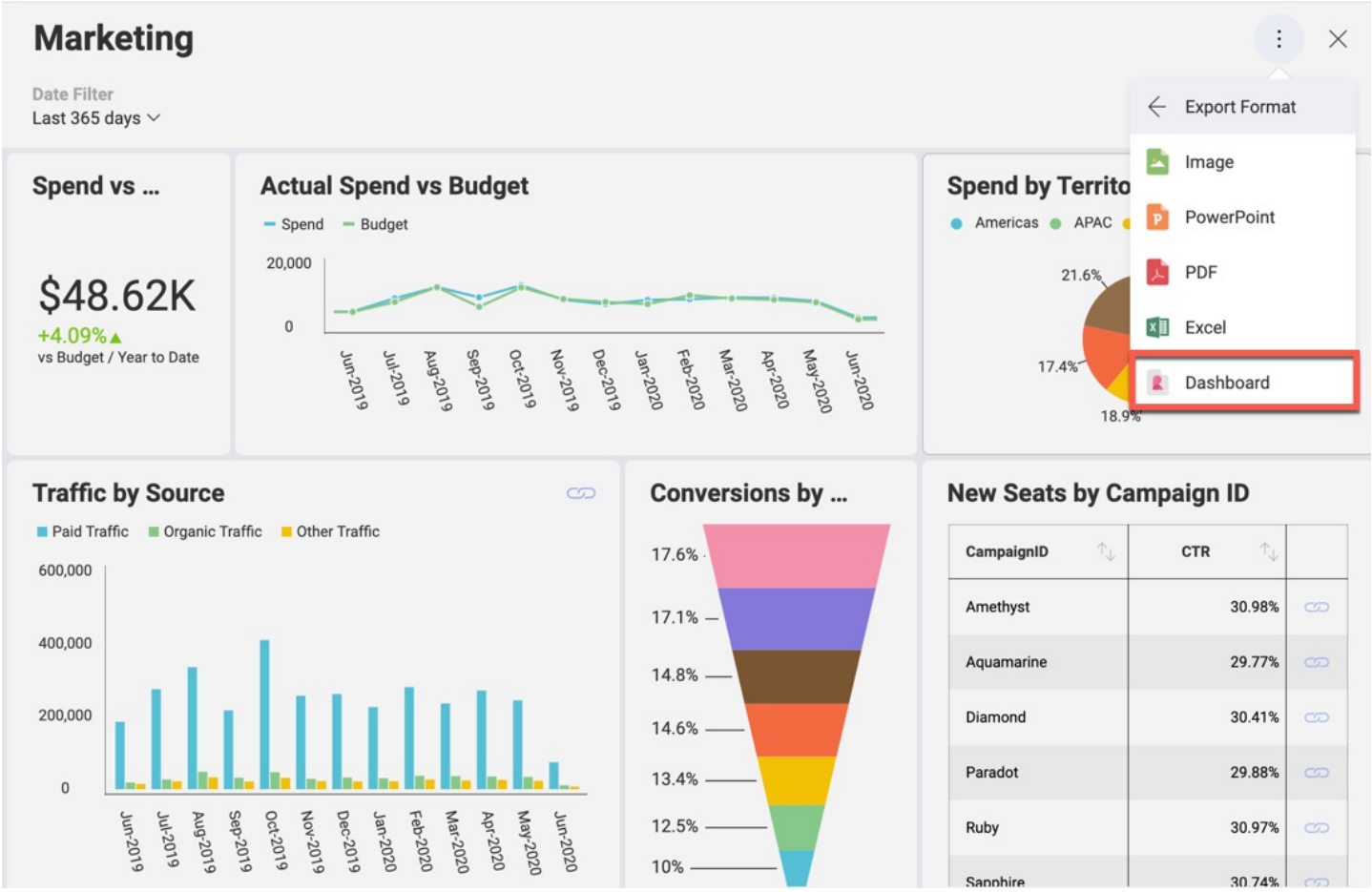
And choose the *Ocean Theme*:



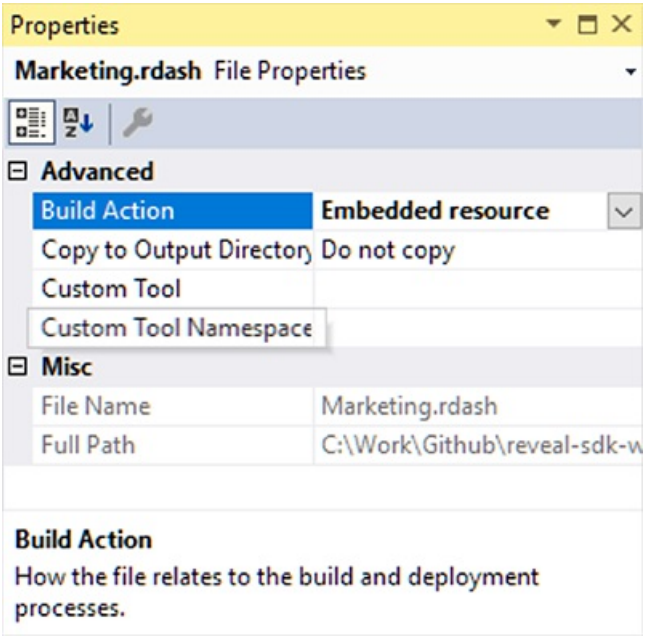
Save the modified dashboard and *Export* it:

Note

As the Marketing dashboard is part of the Reveal App **Samples**, you cannot save it the same way as a regular dashboard. Instead, you need to use **Save As** and choose a location.



Move the **Marketing.rdash** dashboard file to the Dashboards folder, which you created in step 3, and set the Build Action for this item to **Embedded resource** in Visual Studio:



Now let's add a new page *Marketing.cshtml* **to the Views folder** in order to visualize the downloaded dashboard:

```
@{
    ViewData["Title"] = "Marketing";
}

@section Scripts
    {
    <script type="text/javascript">
        // Load dashboard in #revealView element
    </script>
}

<section>
    <div id="revealView" style="height:800px;"></div>
</section>
```

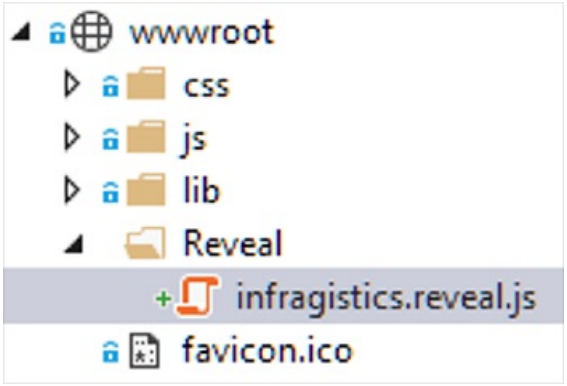
Then, add a new action method in **HomeController.cs**:

```
public IActionResult Marketing()
{
    return View();
}
```

Let's add some references to scripts & css files for some third party dependencies of Reveal in **_Layout.cshtml** :

```
<script src="https://unpkg.com/dayjs"></script>
<link rel="stylesheet" href="https://code.jquery.com/ui/1.10.2/themes/smoothness/jquery-ui.css" />
<script type="text/javascript" src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.2.1.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
<script src="https://cdn.quilljs.com/1.3.6/quill.js"></script>
<link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet">
```

To continue, create a new Reveal folder in the *wwwroot* folder of the project. Copy there **infragistics.reveal.js**, which you can find in the **<InstallationDirectory>\SDK\Web\JS\Client** of the *Reveal SDK*:



And then reference this library in **_Layout.cshtml** after the scripts for Day.js:

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

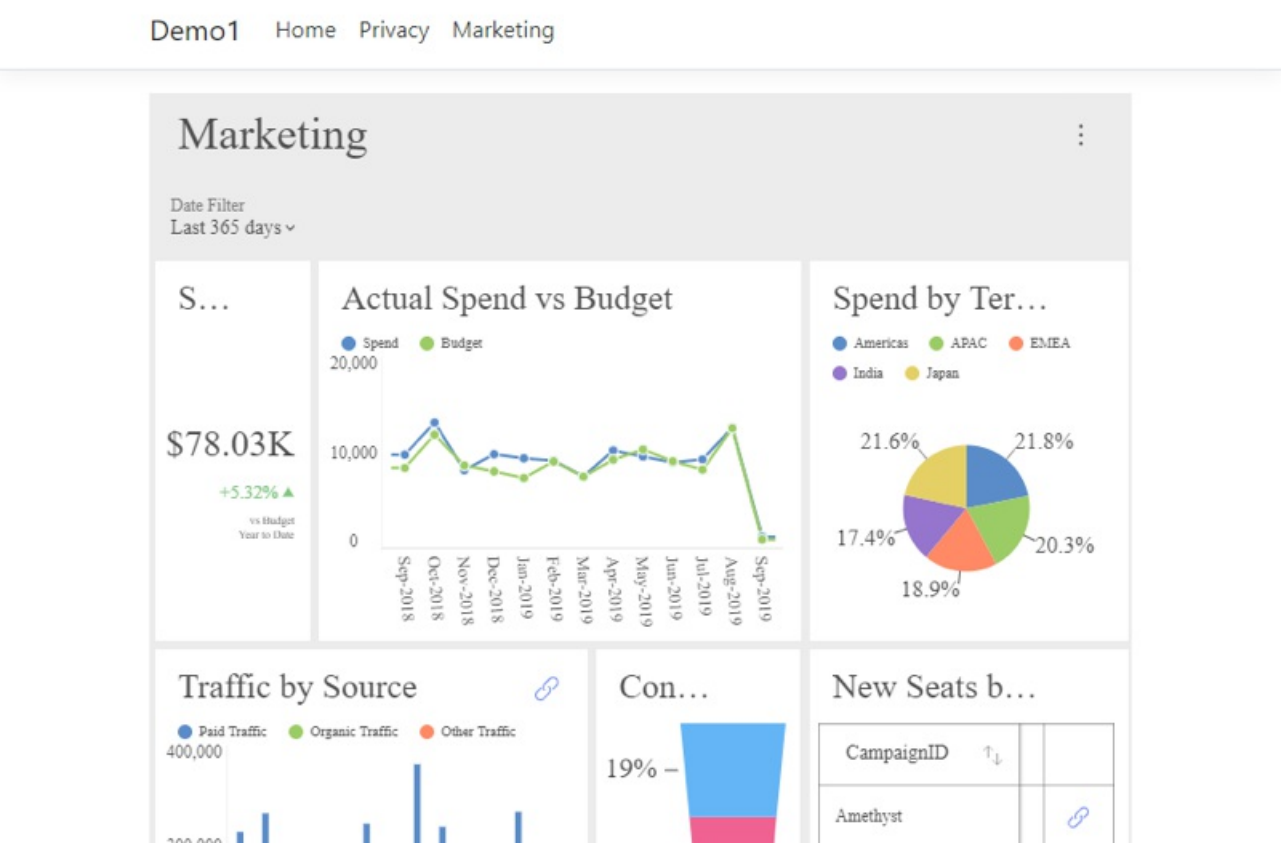
In the same file, also remove the footer section and add a link in the navigation for the new page:

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Marketing">Marketing</a>
</li>
```

Let's update the script in **Marketing.cshtml** with the logic for loading the dashboard:

```
var dashboardId = "Marketing.rdash";

$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
}, function (error) {
    //Process any error that might occur here
});
```

Finally, when running the web page, you can see the dashboard:



If you experience any issues, please refer to the **Setup and Configuration (Web)** topic.

Step 5 - Use Reveal Fonts

Reveal app uses Roboto fonts. In order to achieve the same look as in the app, download the fonts from https://fonts.google.com/specimen/Roboto and copy the following TTF files to the **wwwroot/css** folder of your project:

- Roboto-Regular.ttf
- Roboto-Bold.ttf
- Roboto-Light.ttf
- Roboto-Medium.ttf

Then, add references in the **site.css** as follows:

```
@font-face {
  font-family: "Roboto-Regular";
  src: url("Roboto-Regular.ttf");
}

@font-face {
  font-family: "Roboto-Bold";
  src: url("Roboto-Bold.ttf");
}

@font-face {
  font-family: "Roboto-Light";
  src: url("Roboto-Light.ttf");
}

@font-face {
  font-family: "Roboto-Medium";
  src: url("Roboto-Medium.ttf");
}
```

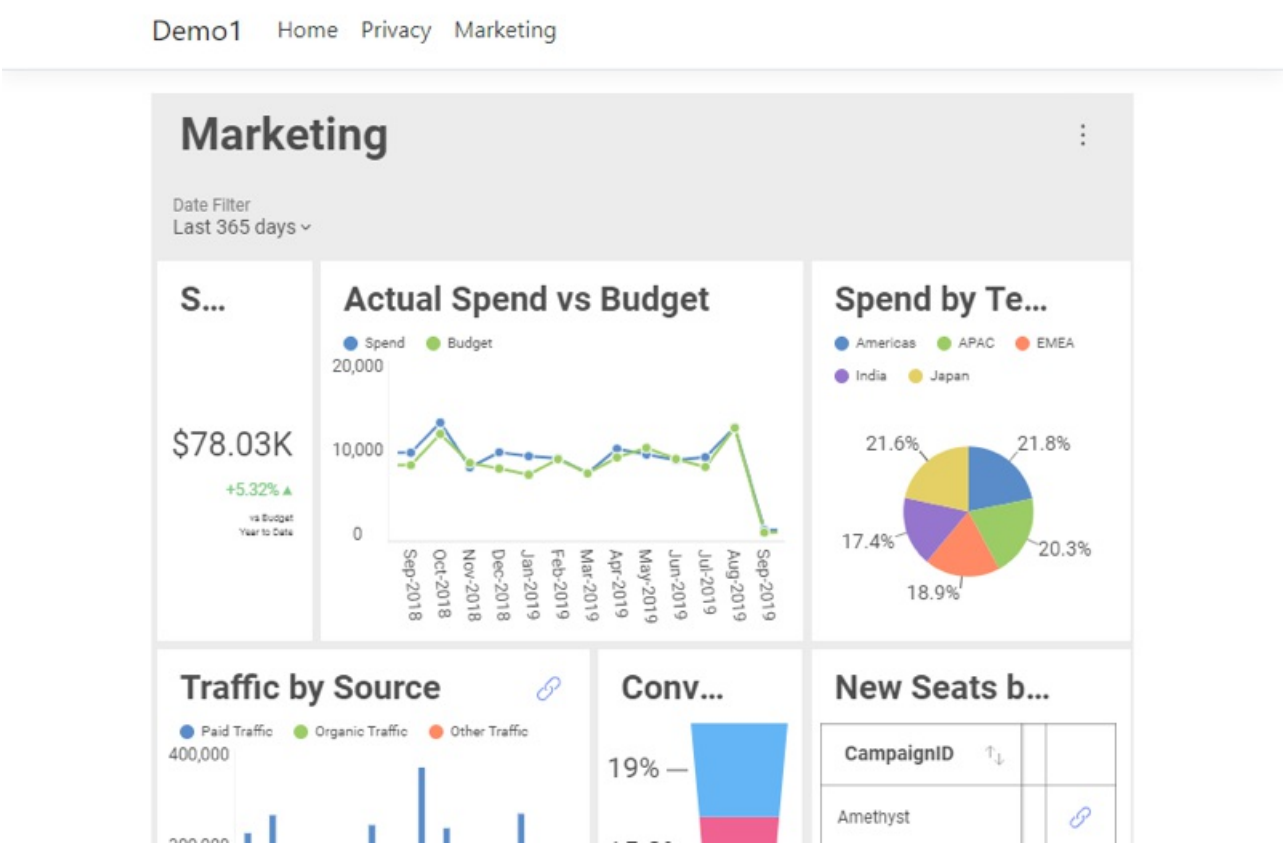
For font loading improvements add a reference to the Google Web Font Loader in **_Layout.cshtml** next to the infragistics.reveal.js reference:

```
<script src="https://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js"></script>
```

Finally, modify the script section of the **Marketing.cshtml** page to take advantage of the font loader:

```
WebFont.load({
    custom: {
        families: ['Roboto-Regular', 'Roboto-Bold', 'Roboto-Light', 'Roboto-Medium'],
        urls: ['/css/site.css']
    },
    active: function () {
        var dashboardId = "Marketing.rdash";

        $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
            var revealView = new $.ig.RevealView("#revealView");
            revealView.dashboard = dashboard;
        }, function (error) {
            //Process any error that might occur here
        });
    },
});
```

Voila!



Step 6 - Style the Client Application

Instead of using the default template, you can style the Client application.

Remove the Privacy from **HomeController.cs** and modify the Index to redirect to Marketing:

```
public IActionResult Index()
{
    return RedirectToAction("Marketing");
}
```

Also, remove the *Index.cshtml* and *Privacy.cshtml* files since they won't be used. Remove the style setting for the element in *Marketing.cshtml*.

Create a new img folder in *wwwroot* and copy there the **logo.png**, which you can download from here.

In **_Layout.cshtml** make the following changes:

- Change the title from *Demo1* to *Overview*
- Remove the div after the header
- Modify the header by adding logo, separator and title:

```
<header>
    <div class="header">
        <img class="logo" src="~/img/logo.png" alt="logo" />
        <span class="line" />
        <h1>Overview</h1>
    </div>
</header>
```

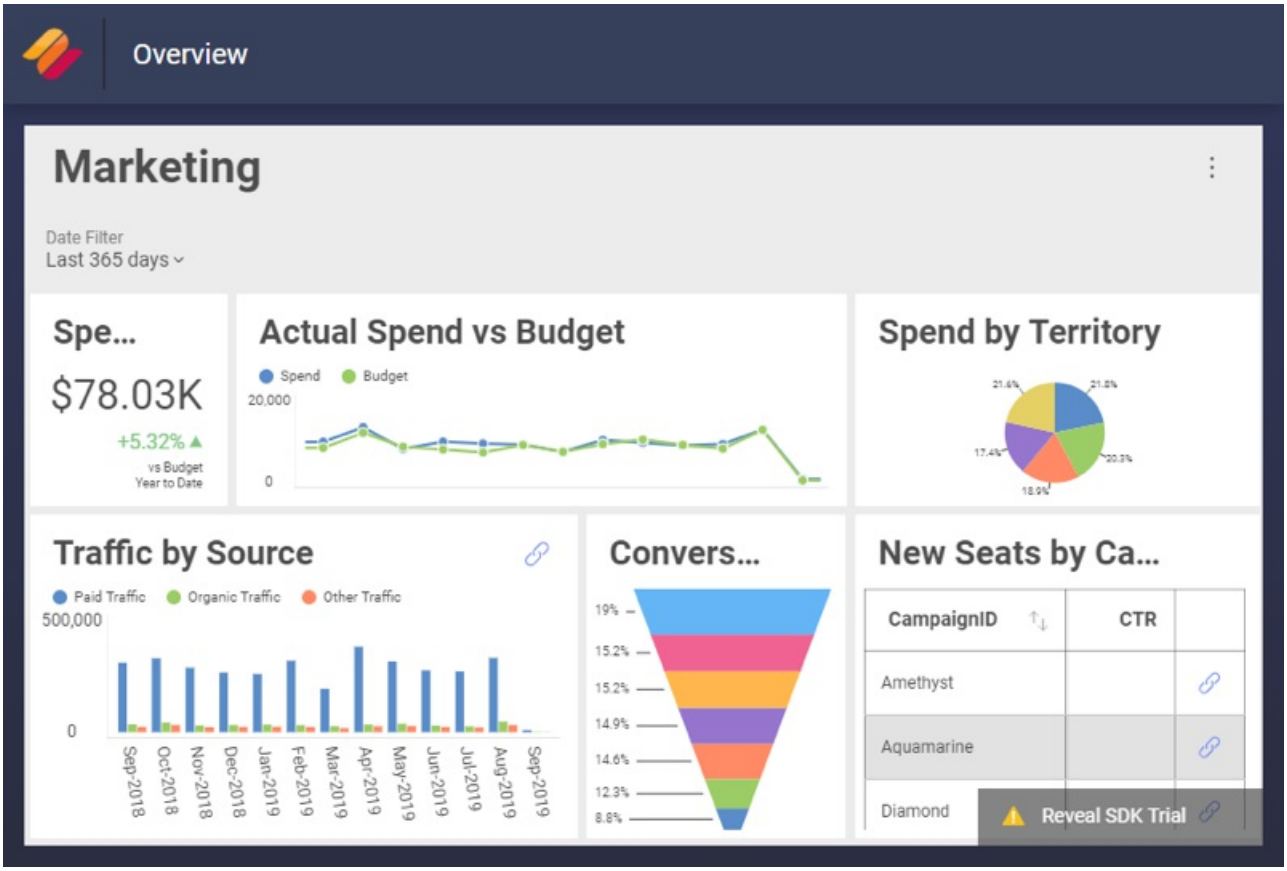
In **site.css** remove all the styles, except the ones we added for the *Roboto* fonts and add styles for the header:

```
/* Header
  ------------------------------------------------- */

  header {
     display: flex;
     width: 100%;
     height: 70px;
     box-shadow: 0 4px 12px 0 rgba(0, 0, 0, 0.2);
     background-color: #37405a;
  }

  img.logo {
     width: 50px;
     height: 50px;
     margin: 10px;
     float: left;
  }

  span.line {
     float: left;
     width: 1px;
     height: 50px;
     margin-top: 10px;
     border: solid 1px #2b2e40;
  }

  h1 {
     float: left;
     padding-top: 12px;
     padding-left: 20px;
     height: 24px;
     font-family: Roboto-Regular;
     font-size: 20px;
     font-weight: 400;
     color: #ffffff;
  }
```

And styles for the body:

```
/* Body
  ------------------------------------------------- */
  body {
     display: flex;
     flex-direction: column;
     background-image: linear-gradient(to bottom, #30365a, #2b2e40);
  }

  html, body {
     width: 100%;
     height: 100%;
  }

     body section {
        display: block;
        width: 100%;
        height: 100%;
        padding: 15px;
     }

  #revealView {
     height: 100%;
  }
```

And this should be your result:


Overview
Marketing
Date Filter
Last 365 days
Spe...
$78.03K
+5.32%
vs Budget
Year to Date
Actual Spend vs Budget
Spend
Budget
20,000
0
Spend by Territory
21.6%
21.8%
17.4%
20.3%
18.9%
Traffic by Source
Paid Traffic
Organic Traffic
Other Traffic
500,000
0
Sep-2018
Oct-2018
Nov-2018
Dec-2018
Jan-2019
Feb-2019
Mar-2019
Apr-2019
May-2019
Jun-2019
Jul-2019
Aug-2019
Sep-2019
Convers...
19%
15.2%
15.2%
14.9%
14.6%
12.3%
8.8%
New Seats by Ca...
CampaignID
CTR
Amethyst
Aquamarine
Diamond
Reveal SDK Trial

Loading Dashboard Files

Overview

There are two ways to open/save dashboards with the SDK:

- **Server-side**: First, you specify a dashboard ID in the client page. Second, on the server, using a callback method detailed below, you return the stream with the contents of the dashboard with the specified ID.

  Please note that this is the easiest approach and the one recommended when you are first evaluating the SDK.

- **Client-side**: Here you have full control and more flexibility. You provide the stream with the contents of the dashboard on the client page, getting the contents from your own server.

  Using this approach you can, for example, check user permissions, display your own user interface to select the dashboard, or allow users to upload the ".rdash" file to use. For further details about the client-side approach, follow this **Setup and Configuration(Client)**.

The Server-Side Approach

In order to visualize a dashboard, you can provide the SDK with an instance of a Dashboard class, which you could instantiate passing a stream to a rdash or json string representation of a rdash.

The code snippet below shows how to load a .rdash file that is added to the project as an embedded resource. Please note that this method is the implementation for **RevealSdkContextBase.GetDashboardAsync**.

Code

```
public override Task<Dashboard> GetDashboardAsync(string dashboardId)
{
    var dashboardFileName = dashboardId + ".rdash";
    var resourceName = $"Demo1.Dashboards.{dashboardFileName}";
    var assembly = Assembly.GetExecutingAssembly();
    var rdashStream = assembly.GetManifestResourceStream(resourceName)
    var dashboard = new Dashboard(rdashStream);

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

This code for **RevealSdkContextBase.GetDashboardAsync** will be invoked on the server when you use **RVDashboard.loadDashboard** function on the client. And you will get the *dashboardId* that was specified client-side as the first parameter.

Replacing Data Sources (MS SQL Server)

Overview

Before loading and processing the data for a dashboard (by Reveal Server SDK), you can override the configuration or data to be used for each visualization of the dashboard.

One of the properties to implement in **RevealSdkContextBase** is:

```
IRVDataSourceProvider DataSourceProvider { get;  }
```

A class implementing the interface **IRVDataSourceProvider** may replace or modify the data source used by a given visualization or dashboard filter.

Use Cases

Below you can find a list of common use cases:

- You can change the name of the database being used, depending on the current user, or any other attributes your app might get like userId, division, company, customer, etc. By doing this, you can have a single dashboard getting data from a multi-tenant database.
- You can change the name of the table being used, the path of the file to load, etc. The use case is similar to the one described above.
- You can replace a data source with an in-memory data source. As the Reveal App doesn't support in-memory data sources, you can design a dashboard using a CSV file and then use this callback to replace the CSV data source with an in-memory one. In this scenario, data is actually loaded from memory (or with your custom data loader). For further details about how to use in-memory data sources, refer to **In-Memory Data Support**.

Code

The following code snippet shows an example of how to replace the data source for visualizations in the dashboard. The method **ChangeVisualizationDataSourceItemAsync** will be invoked for every visualization, on every single dashboard being opened.

```
public class SampleDataSourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem> ChangeDashboardFilterDataSourceItemAsync(string userId, string dashboardId, RVDashboardFilter globalFilter,
RVDataSourceItem dataSourceItem)
    {
      return Task.FromResult<RVDataSourceItem>(null);
    }

    public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(
      string userId, string dashboardId, RVVisualization visualization,
      RVDataSourceItem dataSourceItem)
    {
      var sqlServerDsi = dataSourceItem as RVSqlServerDataSourceItem;
      if (sqlServerDsi != null)
      {
        // Change SQL Server host and database
        var sqlServerDS = (RVSqlServerDataSource)sqlServerDsi.DataSource;
        sqlServerDS.Host = "10.0.0.20";
        sqlServerDS.Database = "Adventure Works";

        // Change SQL Server table/view
        sqlServerDsi.Table = "Employees";
        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
      }

      // Fully replace a data source item with a new one
      if (visualization.Title == "Top Customers")
      {
        var sqlDs = new RVSqlServerDataSource();
        sqlDs.Host = "salesdb.local";
        sqlDs.Database = "Sales";

        var sqlDsi = new RVSqlServerDataSourceItem(sqlDs);
        sqlServerDsi.Table = "Customers";

        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
      }

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

In the example above, the following two replacements will be performed:

- All data sources using a MS SQL Server database will be changed to use the hardcoded server “10.0.0.20”, the “Adventure Works” database, and the “Employees” table.

  Note: This is a simplified scenario, replacing all visualizations to get data from the same table, which makes no sense as a real world scenario. In real-world applications, you’re probably going to use additional information like userId, dashboardId, or the values in the data source itself (server, database, etc.) to infer the new values to be used.

- All widgets with the title “Top Customers” will have their data source set to a new SQL Server data source, getting data from the “Sales” database in the “salesdb.local” server, using the table “Customers”.

Please note that in addition to implement **IRVDataSourceProvider** you need to modify your implementation of **RevealSdkContextBase.DataSourceProvider** to return it:

```
IRVDataSourceProvider DataSourceProvider => new SampleDataSourceProvider();
```

Replacing Excel and CSV file DataSources

Overview

When we create a dashboard in **Reveal Application** we often use Excel or CSV files stored in the cloud to populate it with data.
After exporting the dashboard and embedding it in custom application, we can move these files in a local directory and then use the **Reveal SDK** to access and set them as a local datasource.

Steps

To populate the exported dashboard using local Excel and CSV files, you need to follow these steps:

1. **Export the dashboard** file as explained in **Getting Dashboards for the SDK**
2. **Load the dashboard** in your application as described in **Creating Your First App**
3. **Download the files** you used to create the dashboard from your cloud storage and copy them to a local folder.
4. **Set the local folder name** as a value of the *LocalStoragePath*. Details about this you can find here: **Setup and Configuration(Server) - Initializing the Server SDK**
5. **Add a new *CloudToLocalDatasourceProvider* class** in the project.
6. **Copy the implementation code** from the relevant snippet in **Code** section below.
7. **Set the *DataSourceProvider* property** of the *RevealSdkContext* class to *CloudToLocalDatasourceProvider*:

```
public override IRVDataSourceProvider DataSourceProvider => new CloudToLocalDatasourceProvider();
```

Code

```
public class CloudToLocalDatasourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem> ChangeDashboardFilterDataSourceItemAsync(string userId, string dashboardId,
                RVDashboardFilter filter, RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }

    public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(string userId, string dashboardId,
                RVVisualization visualization, RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }

    protected Task<RVDataSourceItem> ProcessDataSourceItem(RVDataSourceItem dataSourceItem)
    {
        // Return data source unless it is an excel or csv file.
        if (dataSourceItem is RVExcelDataSourceItem == false &&
            dataSourceItem is RVCsvDataSourceItem == false)
        {
            return Task.FromResult(dataSourceItem);
        }

        var resourceBased = dataSourceItem as RVResourceBasedDataSourceItem;
        var resourceItem = resourceBased?.ResourceItem as RVDataSourceItem;
        var title = resourceItem?.Title;

        if (string.IsNullOrEmpty(title))
        {
            return Task.FromResult(dataSourceItem);
        }

        var localItem = new RVLocalFileDataSourceItem();

        // The SDK uses the local folder set as LocalStoragePath.
        localItem.Uri = @"local:/" + title;

        // The title assigned here is the original data source name.
        localItem.Title = title;
        resourceBased.ResourceItem = localItem;

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

Note

The *CloudToLocalDatasourceProvider* replaces automatically Excel and CSV files only. Other file types or data sources will remain unchanged. The replacement files should be the same ones used to create the dashboard or, alternatively, new files that share the **same schema** but with different data.

In-Memory Data Support

Overview

In some cases you need to use data already in memory as part of your application state. Using, for example, the result of a report requested by the user, or information from a data source not yet supported by Reveal (like a custom database or a specific file format).

In-Memory is a special type of data source that can be used only with the SDK and not in the Reveal application out of the box. Because of this, you cannot use an "in-memory data source" directly, you need to take a different approach as explained below.

Using In-Memory Data Source

The recommended approach is to **define a data file with a schema, matching your in-memory data**. Data files can be, for example, CSV or Excel files, and a schema is basically a list of fields and the data type for each field.

In the example below you find details about how to create a data file with a given schema, and then use data in memory instead of getting information from a database.

Code Example

In the following example, you want to use in-memory data with the list of Employees in the company, in order to embed a Reveal dashboard showing HR metrics in your HR system. And instead of getting the list of employees from your database, you want to use data in memory.

To achieve all that, you will need to create and export a dashboard in the Reveal application using dummy data.

About the Reveal Application

The Reveal Application is a self-service business intelligence tool that enables you to create, view and share dashboards with your teams. For further details about the Reveal app, you can access an **online demo** or browse the **Help Documentation**.

Getting the Data File and Sample Dashboard Ready

As simplified Employee has only the following properties:

- *EmployeeID*: string
- *Fullname*: string
- *Wage*: numeric

Steps:

1. Create The CSV file with the same schema:

```
EmployeeID,Fullname,Wage
23,John Smith,345.67
45,Emma Thompson,432.23
```

2. Upload the file to your preferred File Sharing System, like Dropbox or Google Drive.
3. Create a dashboard within the Reveal app using the dummy data. Please note that you are going to provide the real production data later in your application.
4. Export the dashboard from the Reveal app (Dashboard Menu → Export → Dashboard) and save a .rdash file.

Visualizing the Dashboard and Returning the Actual Data

Now you need to visualize the dashboard using your own data instead of the dummy one.

1. Implement **IRVDataSourceProvider** and return it as the **DataSourceProvider** property in **RevealSdkContextBase**, as described in **Replacing Data Sources**.

   Then, in the implementation for the method **ChangeVisualizationDataSourceItemAsync**, you need to add a code similar to this one:

```
public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(string userId, string dashboardId, RVVisualization visualization,
RVDataSourceItem dataSourceItem)
{
   var csvDsi = dataSourceItem as RVCsvDataSourceItem;
   if (csvDsi != null)
   {
      var inMemDsi = new RVInMemoryDataSourceItem("employees");
      return Task.FromResult((RVDataSourceItem)inMemDsi);
   }
   return Task.FromResult((RVDataSourceItem)null);
}
```

   This way you basically replace all references to CSV files in the dashboard with the in-memory data source identified by “employees”. This identification will be used later when returning the data.

2. Implement the method that will return the actual data, to do that implement **IRVDataProvider** as shown below:

```
public class EmbedDataProvider : IRVDataProvider
{
   public Task<IRVInMemoryData> GetData(string userId, RVInMemoryDataSourceItem dataSourceItem)
   {
      var datasetId = dataSourceItem.DatasetId;
      if (datasetId == "employees")
      {
         var data = new List<Employee>()
            {
               new Employee(){ EmployeeID = "1", Fullname="John Doe", Wage = 80325.61 },
               new Employee(){ EmployeeID = "2", Fullname="Doe John", Wage = 10325.61 },
            };
         return Task.FromResult<IRVInMemoryData>(new RVInMemoryData<Employee>(data));
      }
      else
      {
         throw new Exception("Invalid data requested");
      }
   }
}
```

   Please note that the properties in the Employee class are named exactly as the columns in the CSV file, and the data type is also the same. In case you want to alter the field name, field label and/or data type of any of the properties you can use attributes in the class declaration:

   - RVSchemaColumn attribute can be used to alter the field name and/or data type.
   - DisplayName attribute can be used to alter the field label.

```
public class Employee
{
    [RVSchemaColumn("EmployeeID", RVSchemaColumnType.Number)]
    public string EmployeeID { get; set; }

    [DisplayName("EmployeeFullname")]
    public string Fullname { get; set; }

    [RVSchemaColumn("MonthlyWage")]
    public double Wage { get; set; }
}
```

In addition, to implement **IRVDataProvider** you need to modify your implementation of **RevealSdkContextBase.DataProvider** to return it:

```
IRVDataProvider DataProvider => new EmbedDataProvider();
```

Providing Credentials to Data Sources

Overview

The Server SDK allows you to pass in a set of credentials to be used when accessing the data source.

Code

The first step is to implement **IRVAuthenticationProvider** and return it as the **AuthenticationProvider** property in **RevealSdkContextBase**, as shown below.

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
   public Task<IRVDataSourceCredential> ResolveCredentialsAsync(string userId, RVDashboardDataSource dataSource)
   {
      IRVDataSourceCredential userCredential = null;
      if (dataSource is RVPostgresDataSource)
      {
         userCredential = new RVUsernamePasswordDataSourceCredential("postgresuser", "password");
      }
      else if (dataSource is RVSqlServerDataSource)
      {
         // The "domain" parameter is not always needed and this depends on your SQL Server configuration.
         userCredential = new RVUsernamePasswordDataSourceCredential("sqlserveruser", "password", "domain");
      }
      else if (dataSource is RVGoogleDriveDataSource)
      {
         userCredential = new RVBearerTokenDataSourceCredential("fhJhbUci0mJSUzi1nIiSint....", "user@company.com");
      }
      else if (dataSource is RVRESTDataSource)
      {
         userCredential = new RVUsernamePasswordDataSourceCredential(); // Anonymous
      }
      return Task.FromResult<IRVDataSourceCredential>(userCredential);
   }
}
```

Choosing Which Class to Implement

There are two classes that can be used, both implementing the **IRVDataSourceCredential** interface. You need to choose the class depending on your data source, as detailed below.

- Class **RVBearerTokenDataSourceCredential** works with:
    - Analytics tools (Google Analytics).
    - Content Managers and Cloud Services (Box, Dropbox, Google Drive, OneDrive and SharePoint Online).
- Class **RVUsernamePasswordDataSourceCredential** works with:
    - Customer Relationship Managers (Microsoft Dynamics CRM On-Premises and Online)
    - Databases (Microsoft SQL Server, Microsoft Analysis Services Server, MySQL, PostgreSQL, Oracle, Sybase)
- **Both classes** work with:
    - Other Data Sources (OData Feed, Web Resources, REST API).

No Authentication

Sometimes you might work with an anonymous resource, without authentication. In this particular case, you can use **RVUsernamePasswordDataSourceCredential**, which has an empty constructor. You can do this for any data source that works with the class.

Code snippet to be used with the sample above:

```
else if (dataSource is RVRESTDataSource)
{
    userCredential = new RVUsernamePasswordDataSourceCredential();
}
```

Creating New Visualizations and Dashboards

Overview

As described in **Editing & Saving Dashboards**, there are two ways to handle how you save changes to dashboards: **client-side and server-side**. Those scenarios work fine when users make minor changes to existing dashboards like:

- Adding/modifying filters
- Changing the type of visualization (chart, gauge, grid, etc.)
- Changing the theme

However, to add new visualizations the user needs to **select the data source** to be used. To do that, the containing application needs to provide information to the SDK, so it can display the list of data sources available for a new visualization.

Displaying a List of Data Sources

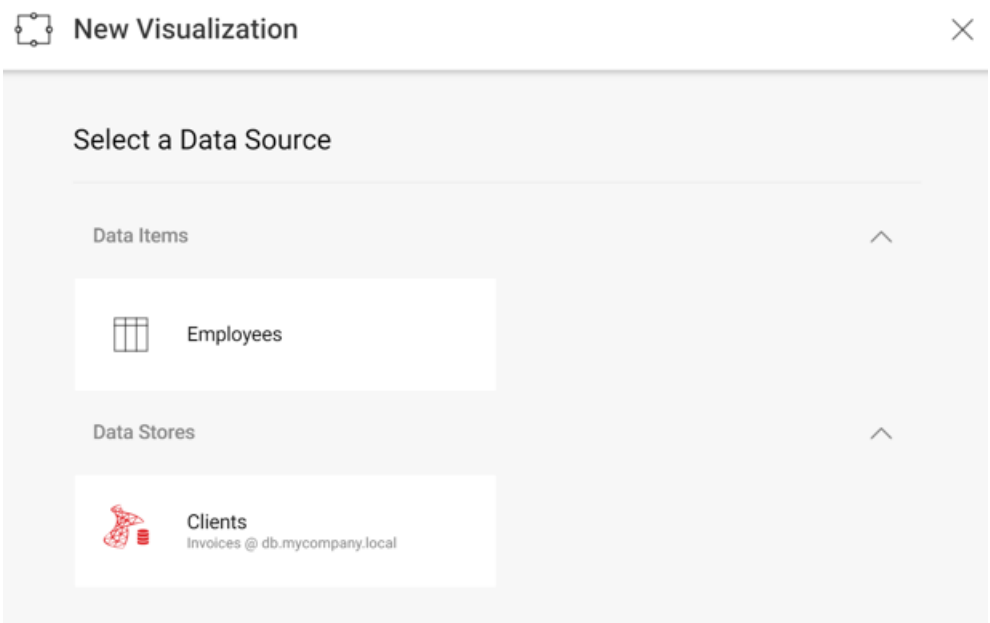
The callback you need to use to display a list of data sources is **onDataSourcesRequested**. In the case that you don't set your own function to this callback, when a new visualization is created, Reveal will display all data sources used in the dashboard (if any).

Code:

The code below shows how to configure the *data source selection* screen to show an "in-memory" item and a SQL Server data source.

```
window.revealView.onDataSourcesRequested = function (callback) {
    var inMemoryDSI = new $.ig.RVInMemoryDataSourceItem("employees");
    inMemoryDSI.title = "Employees";
    inMemoryDSI.description = "Employees";

    var sqlDs = new $.ig.RVSqlServerDataSource();
    sqlDs.title = "Clients";
    sqlDs.id = "SqlDataSource1";
    sqlDs.host = "db.mycompany.local";
    sqlDs.port = 1433;
    sqlDs.database ="Invoices";

    callback(new $.ig.RevealDataSources([sqlDs], [inMemoryDSI], false));
};
```

The "false" value in the third parameter prevents existing data sources on the dashboard from being displayed. So, when creating a new widget using the "+" button, you should get the following screen:



Please note that the “employees” parameter passed to the “RVInMemoryDataSourceItem” constructor, is the same dataset id used in **In-Memory Data Support** and identifies the dataset to be returned on the server side.

Creating New Dashboards

Creating dashboards from scratch is really simple, you just need to:

- Initialize **$.ig.RevealView** object, without setting the dashboard property to $.ig.RevealView and without using **$.ig.RVDashboard.loadDashboard**;
- Set *startInEditMode* to true, to start the dashboard in edit mode:
- Set the dashboard property to newly created instance of **$.ig.RVDashboard**

```
var revealView = new $.ig.RevealView("#revealView");
revealView.startInEditMode = true;
revealView.dashboard = new $.ig.RVDashboard();
```

You can find a working example, **CreateDashboard.cshtml**, in the *UpMedia* web application distributed with the SDK.

Editing & Saving Dashboards

Editing dashboards

The **dashboard** property (type $.ig.RVDashboard) of **revealView** is updated when the end user starts editing the dashboard. For example, when adding or removing visualizations or filters, $.ig.RVDashboard's collections get automatically updated.

In addition, the **$.ig.RVDashboard** class includes the **onHasChangesChanged** property that is very useful to check if there are unsaved changes in the dashboard.

Code Sample:

```
dashboard.onHasChangesChanged = function (hasChanges) {
    console.log("Has Changes: " + hasChanges);
};
```

After a user finishes editing a visualization, upon closing the Visualization Editor, the $.ig.RevealView's **visualizationEditorClosed** event is fired:

```
revealView.onVisualizationEditorClosed = function (args) {
    if (args.isCancelled) {
        console.log("Visualization editor cancelled " + (args.isNewVisualization ? "creating a new visualization " : "editing " + args.visualization.title));
        return;
    }
    if (args.isNewVisualization) {
        console.log("New Visualization created: " + args.visualization.title);
    } else {
        console.log("Visualization modified: " + args.visualization.title);
    }
};
```

In the case that you need to control how to add new visualizations please refer to **Creating New Visualizations and Dashboards**.

Saving Dashboards

As described in **Loading Dashboard Files**, there are two ways to handle how you save changes to dashboards:

- **Client-side**: To use this method you need to set a function in the **onSave** attribute of the **revealView** object. This is the recommended approach as it gives more flexibility to the containing app on how operations (save and save as) are performed.

  Code Sample:

  ```
  revealView.onSave = function(rv, saveEvent) {
      saveEvent.serialize(function(blobValue) {
          //TODO: save the blob value, for example using a XMLHttpRequest object
          //to POST to the server
      });
  };
  ```

  In case you don't want to handle the save action, you can turn off the option to edit dashboards by setting:

  ```
  revealView.canSaveAs = false;
  ```

  This might be useful, for example, when your users are not supposed to make changes.

- **Server-side**: When the **onSave** event is not set in the **$.ig.RevealView** object, the default server-side saving method is used. After the end user saves a modified dashboard, a HTTP POST request is invoked. As a result, the **SaveDashboardAsync** method of the currently defined SDK context is invoked. And you get the _dashboardId as string and a Stream representation of the dashboard in

dashboardStream.

With the server-side approach, you only need to implement the code client-side but you lose flexibility client-side. This means, for example, that the user cannot select the final location where the dashboard will be stored. For further details about the SDK context, please refer to **Defining the Server Context.**.

Exporting a Dashboard or a Visualization

Overview

If you want to export a dashboard or a particular visualization, you can choose between the following export options:

- as an **image**;
- as a **PDF** document;
- as a **PowerPoint** presentation;
- into **Excel** data format.

To enable a dashboard or a visualization export, you can:

- use the export setting in the $.ig.RevealView, or
- initiate export programmatically outside of the $.ig.RevealView, when exporting **as an image**.

Prerequisites for Export as an Image Option

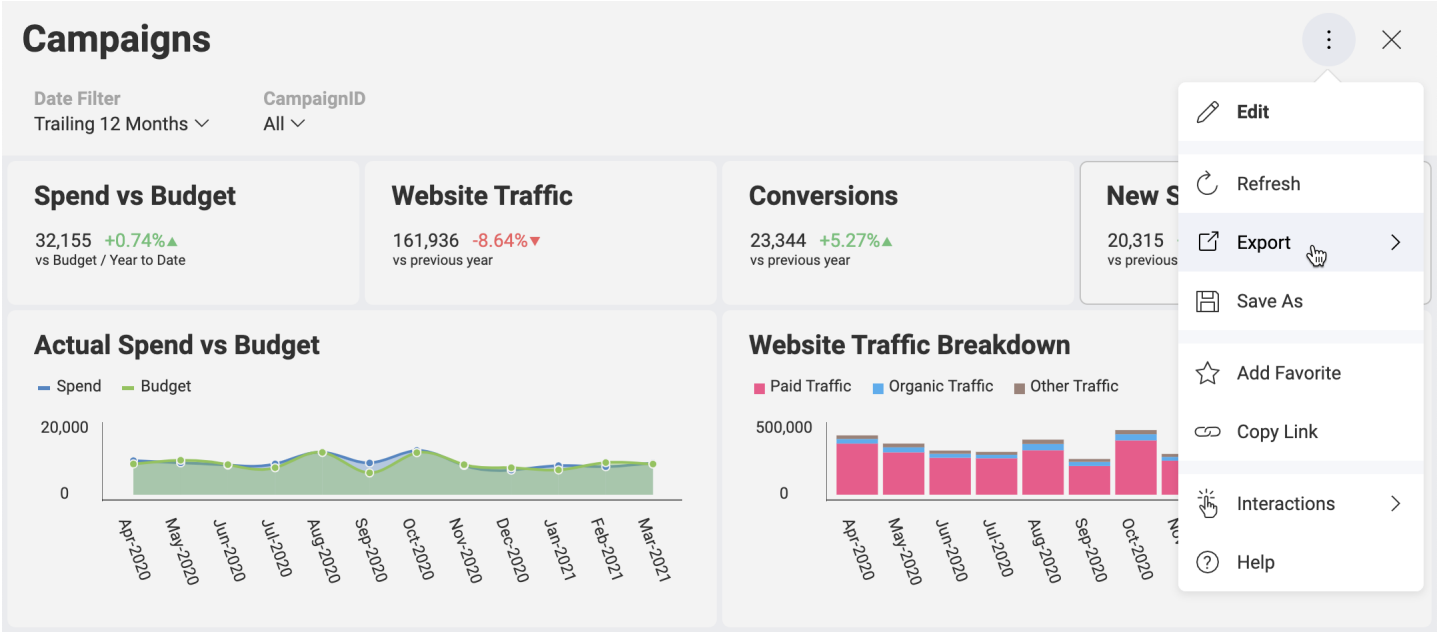
You need to enable the **export image** functionality in the server-side. To do this, please refer to Enabling server-side screenshot generation.

Using the Export Setting

To enable your end users to generate an image, document or a presentation out of a dashboard you simply need to set the relevant property to true:

- **revealView.showExportImage** - for export as an **image**;
- **revealView.showExportToPDF** - for export as a **PDF** document;
- **revealView.showExportToPowerpoint** - for export as a **PowerPoint** presentation;
- **revealView.showExportToExcel** - for export in **Excel** data format.

This will make the *Export* button available in the overflow menu when a dashboard is opened or a particular visualization is maximized.



When the user clicks the *Export* button, they can choose one of the enabled export options.

Specifics when using the image export option

If the user chooses the *Export Image* from the export options, the *Export image* dialog will open. Here, the user can choose between two options: *Copy to clipboard* and *Export Image*. If they click the *Export Image* button on the bottom right, the image will be sent to the end user.

In case the containing application needs to process the exported image in a different way it could provide **onImageExported** callback where the output image could be accessed. Here's a sample implementation of the onImageExported callback:

```
revealView.onImageExported = function (img) {
var body = window.open("about:blank").document.body;
body.appendChild(img);
}
```

Programmatically Initiated Image Export

To get an image of the $.ig.RevealView programmatically, you will need to invoke the **ToImage** method. Calling this method will not result in showing the *Export Image* dialog. This way, you can get a screenshot when the user clicks a button, which is outside of the $.ig.RevealView. This method will create a screenshot of the revealView component as it is displayed on the screen.

```
var image = revealView.toImage();
```

Keep in mind that if the end user has any dialog opened at the time of the *ToImage* method call, the dialog will appear in the screenshot together with the dashboard.

This could be useful in a scenario where the containing application would want to hide the expert button in the reveal view and trigger an export to image from another interaction that occurs outside of the reveal view.

Dashboard Linking

Overview

The Reveal application supports dashboard linking, which allows users to navigate through dashboards. By moving from dashboard to dashboard, you can go from a high level overview of the business' reality to a more detailed view with the specifics.

Common Use Cases

You can have, for example, a "Company 360" dashboard showing key performance indicators for each division (HR, Sales, Marketing). Once the user maximizes one of the visualizations, the navigation is triggered and the user is taken to another dashboard with more detailed information about that division.

Alternatively, you can also use dashboard linking to navigate to a more specific dashboard. For example, in a dashboard with a visualization showing the Top 25 Customers, by selecting one of the customers the user will navigate to a new dashboard. This new dashboard will display detailed information about the selected customer, like recent purchases, contact information, top selling products, etc.

For further details about the Dashboard Linking functionality, refer to **Dashboard Linking** from Reveal's User Guide.

Code Example

You can use dashboard linking with the SDK, but the containing application needs to be involved when the navigation is being triggered. As the SDK does not handle where dashboards are stored, it needs the containing application to provide a dashboard file for the target dashboard.

As a practical example, you can use the **Marketing.cshtml** page in the *UpMedia* sample distributed with the SDK.

Basically, you just have to do two things and the SDK will take care of the rest:

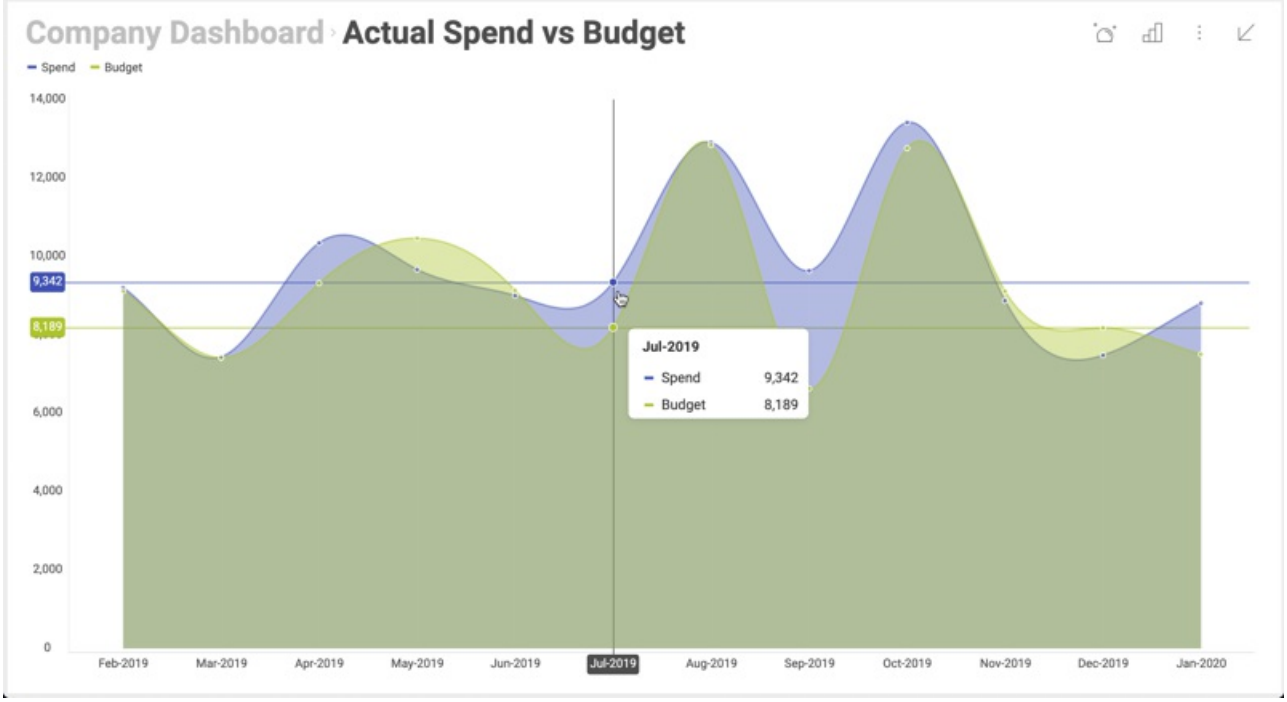
1. Handle the **onVisualizationLinkingDashboard** event.
2. Invoke the callback with the ID of the target dashboard.

```
revealView.onVisualizationLinkingDashboard = function (title, url, callback) {
    //provide the dashboard id of the target of the link
    callback("Campaigns");
};
```

Maximizing Visualizations and Single Visualization Mode

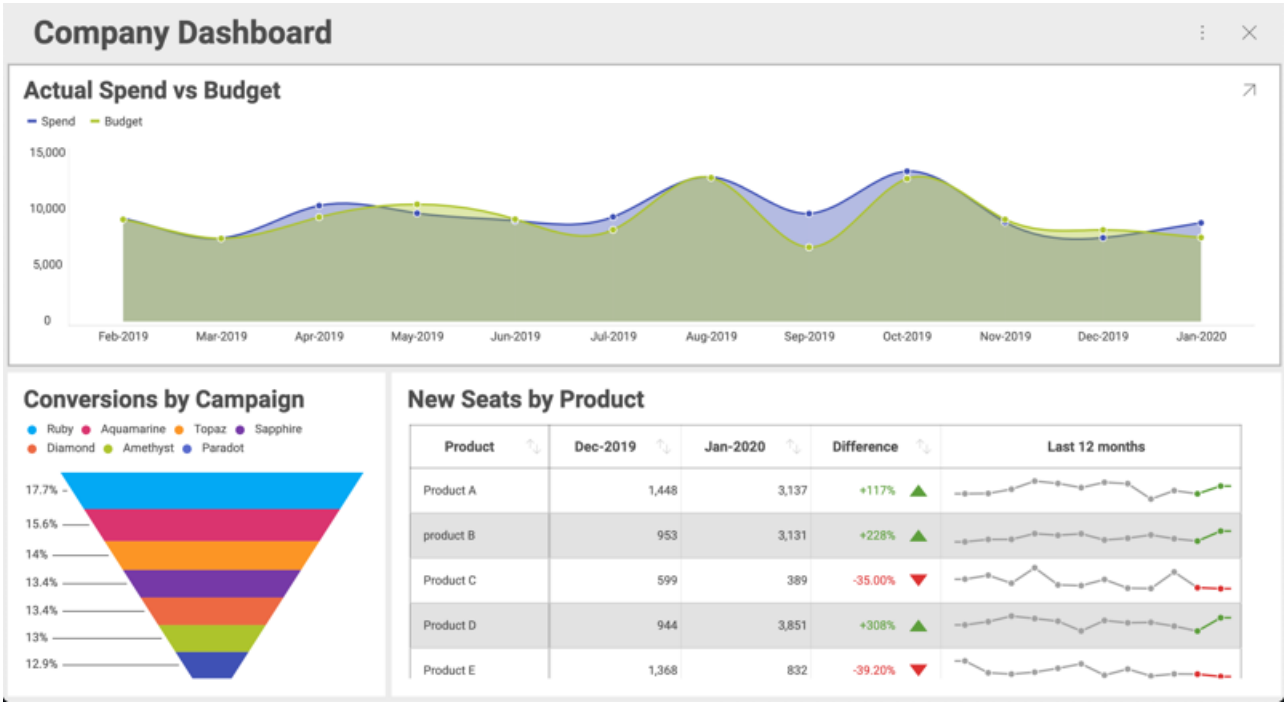
Overview

When displaying a dashboard to the user, there are some cases in which you'd like to display just one maximized visualization. In addition, you might also want to lock the initial visualization and prevent the user from accessing the whole dashboard. You can achieve both scenarios using the Web Client SDK.



Example Details

Let's assume that you have a dashboard with three visualizations, where each visualization is showing data for a different division of your company, i.e., "Marketing", "Sales" and "HR".



In this example, you'd like to showcase these visualizations in your corporate application. You want to include them as part of the information displayed on each division's home page.

Maximizing Visualizations

To open a dashboard with a maximized visualization, you need to set the dashboard property of **revealView** first. Then, set the **maximizedVisualization** property by passing the visualization you want maximized to the **$.ig.RevealView** instance. When you don't set a visualization in this attribute, the whole dashboard is displayed.

As shown in **Configuring the $.ig.RevealView object**, you can display a specific dashboard in your page. This time, you also need to set the **maximizedVisualization** property. As shown in the code snippet below with the visualization "Sales" from the dashboard with ID "AllDivisions".

```
<script type="text/javascript">
...

var dashboardId = 'AllDivisions';

$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
    revealView.maximizedVisualization = dashboard.visualizations.getByTitle('Sales');

}, function (error) {
    console.log(error);
});
</script>

<div id="revealView" style="height:500px;" />
```

Although the initial maximized visualization will be the one with title 'Sales', the end user can still return to the dashboard and see the rest of the visualizations.

Single Visualization Mode

You may also want to lock the initial visualization, making it the only one displayed at all times. This way the dashboard works like a single visualization dashboard. This is the concept behind "single visualization mode".

To turn on the "single visualization mode", just set the **singleVisualizationMode** to true as shown below.

```
revealView.singleVisualizationMode = true;
```

After adding this single line, the dashboard will work as a single visualization dashboard. You can do the same for each division's home page, just replace the title of the visualization in dashboard.visualizations.getByTitle() with the right one.

Dynamically changing a locked visualization

It is also possible for you to dynamically change the single visualization being displayed, without reloading the page. From the user's perspective, your app would be a single page application with a selector of divisions and a maximized visualization. After the user chooses one division from the list, the maximized visualization is updated.

You can achieve this scenario by using the **maximizeVisualization** method in **$.ig.RevealView**, as shown below:

```
<script type="text/javascript">
    var dashboardId = 'AllDivisions';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");
        revealView.singleVisualizationMode = true;
        revealView.dashboard = dashboard;
        revealView.maximizedVisualization = dashboard.visualizations.getByTitle('Sales');

    }, function (error) {
        console.log(error);
    });
    function maximizeVisualization(title) {
        var dashboard = window.revealView.dashboard;
        window.revealView.maximizedVisualization = dashboard.visualizations.getByTitle(title);
    }
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto;">
        <button onclick="maximizeVisualization('Sales')">Sales</button>
        <button onclick="maximizeVisualization('HR')">HR</button>
        <button onclick="maximizeVisualization('Marketing')">Marketing</button>
    </section>
    <div id="revealView" style="height:500px;" />
</section>
```

To take into account:

- The **$.ig.RevealView** object is set in _window.revealView</emphasis> in order to use it later when **maximizeVisualization** property is set.
- The buttons added to the section before the div are used just as an example. They were added as a means to switch the maximized visualization, in your case you'll have to to use a similar code in your application.
- In this example, the buttons are hardcoded to match the visualizations in the sample dashboard, but you can also generate the list of buttons dynamically by iterating the list of visualizations in the dashboard. For further details see **$.ig.RVDashboard.visualizations**.

Creating Custom Themes

Overview

When embedding analytics into your existing applications it is key that those dashboards match your app's look and feel. That's why you have full control over the Reveal dashboards through our SDK.

Key customizations you can achieve with custom themes:

- **Color palettes**: The colors used to show the series in your visualizations. You can add an unlimited number of colors. Once all colors are used in a visualization, Reveal will autogenerate new shades of these colors. This way your colors won't repeat and each value will have its own color.
- **Accent color**: The default accent color in Reveal is a shade of Blue that you can find in the **+ Dashboard** button and other interactive actions. You can change the color to match the same accent color you use in your applications.
- **Conditional formatting colors**: Change the default colors of the bounds you can set when using conditional formatting.
- **Font**: Reveal uses three types of text in the application: regular, medium and bold. You can specify the font uses for each of these text groups.
- **Visualization and dashboard background colors**: You can configure separately the background color of your dashboard and the background color of the visualizations.

Common Use Case: A New Custom Theme

Creating your own theme in Reveal is as easy as creating an instance of the new **$.ig.RevealTheme()** class. This class contains all the customizable settings listed in the overview.

When creating a new **$.ig.RevealTheme** instance, you will get the default values for each setting and you can modify them as needed.

Then, pass the theme instance to the **$.ig.RevealSdkSettings**'s class static theme property. If you have a dashboard or another Reveal component already displayed on your screen, you will need to render it again(set the dashboard property again) in order to see the applied changes.

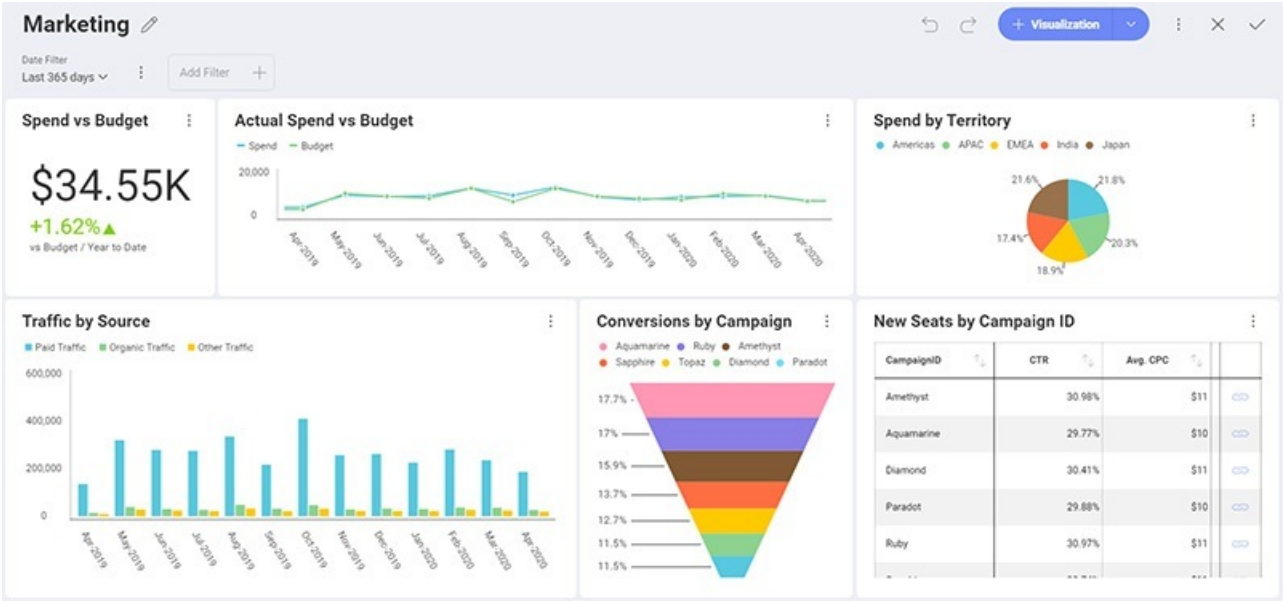
Common Use Case: Modifying a Custom Theme

You may have already applied your own theme but want to modify some of the settings without losing the changes you made to the others.

In this case, you need to get the theme static property from the **$.ig.RevealSdkSettings**. This enables you to get the last values you have set for your RevealTheme settings. Unlike the case when you create a new instance of the RevealTheme from scratch, after applying your changes and updating your theme again, you will get the current values for each setting you didn't modify instead of the default values.

Code Example

First, here's a sample dashboard before we make any changes:



In the following code snippet you can see how to create a new instance of the *revealTheme* class, apply the changes to the settings you want and update the theme in Reveal Web.

```
var revealTheme = new $.ig.RevealTheme();
revealTheme.chartColors = ["rgb(192, 80, 77)", "rgb(101, 197, 235)", "rgb(232, 77, 137)"];

revealTheme.mediumFont = "Gabriola";
revealTheme.boldFont = "Comic Sans MS";
revealTheme.fontColor = "rgb(31, 59, 84)";
revealTheme.accentColor = "rgb(192, 80, 77)";
revealTheme.dashboardBackgroundColor = "rgb(232, 235, 252)";

$.ig.RevealSdkSettings.theme = revealTheme;
```

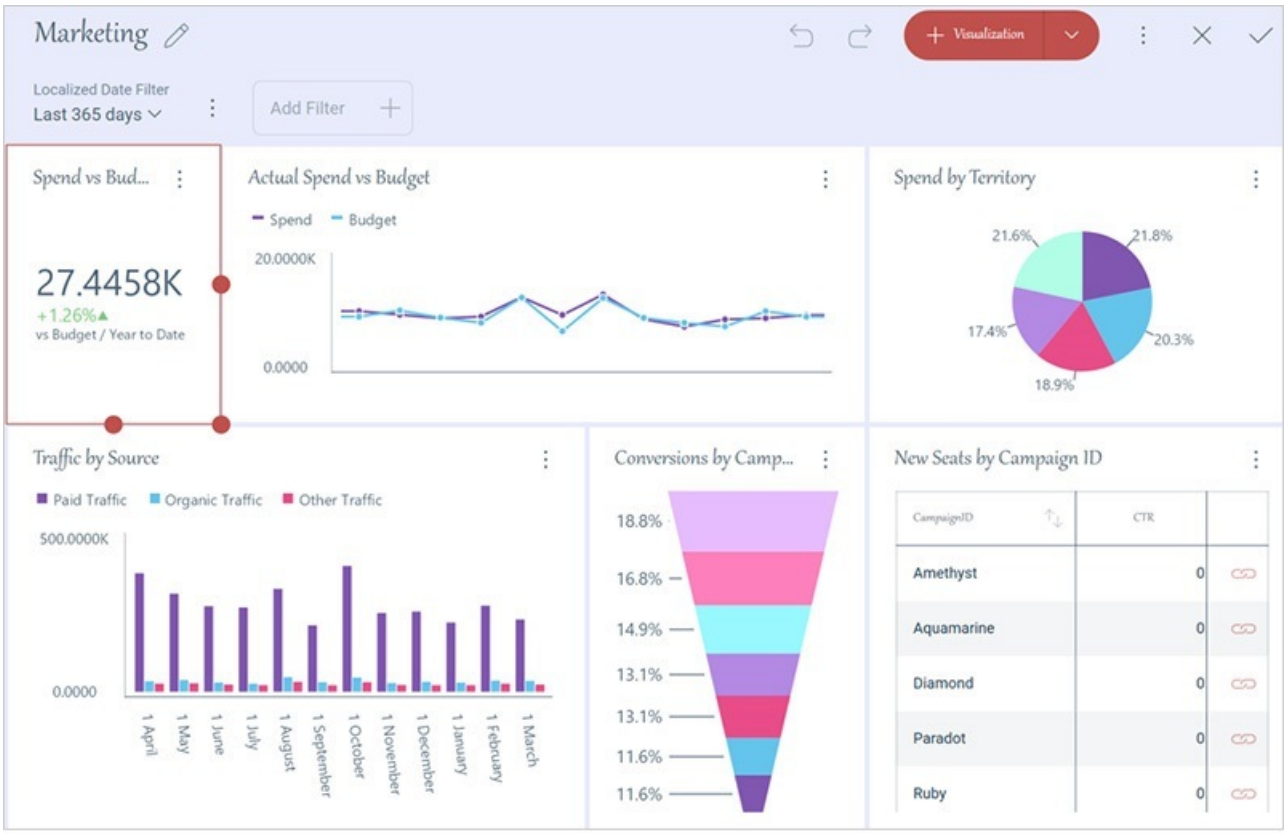
Note

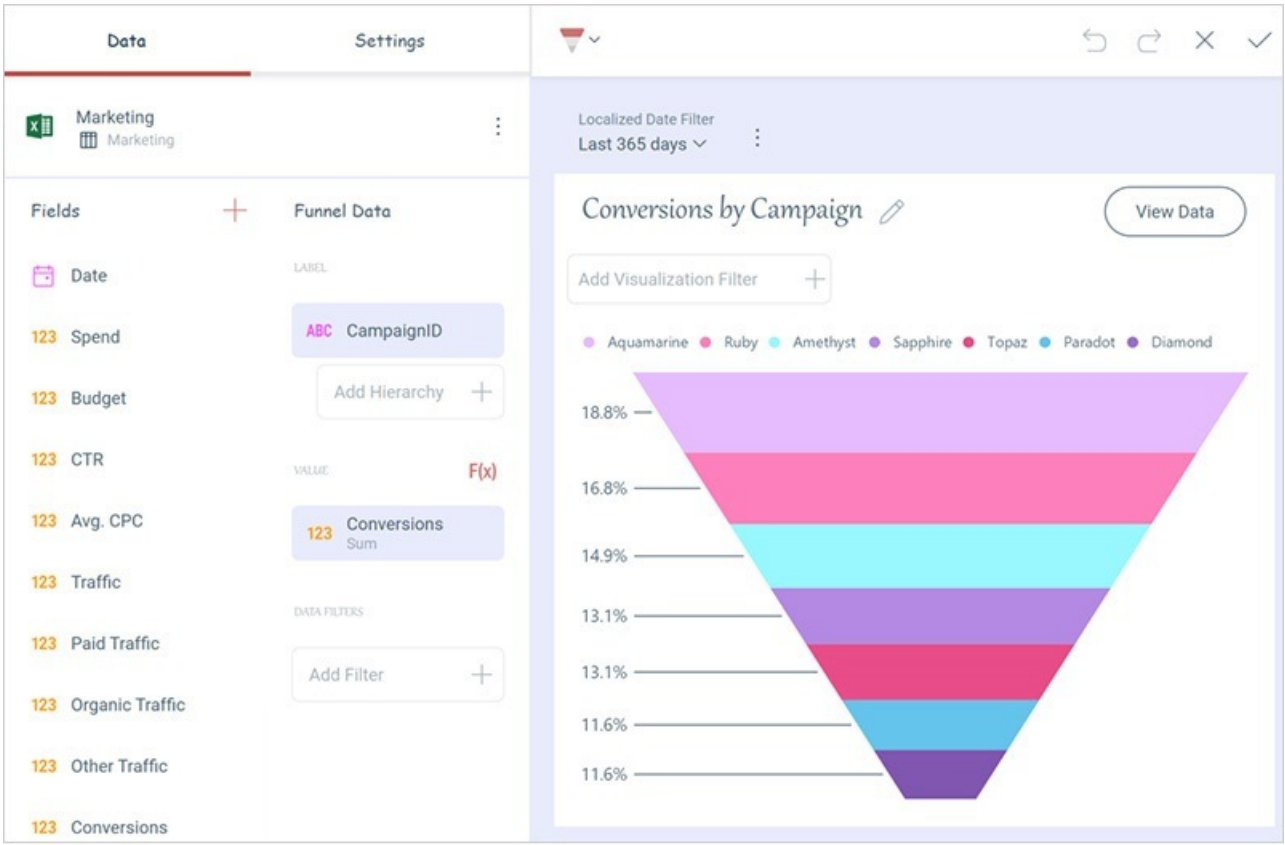
When defining the boldFont, regularFont or mediumFont settings of a reveal theme you need to pass the exact font family name. The weight is defined by the definition of the font itself and not in the name. You might need to use @font-face (CSS property) to make sure the font-face name specified in the $.ig.RevealTheme font settings are available.

In addition, for the fonts customization you need to add these lines to the CSS of the page:

```
<link href="https://fonts.googleapis.com/css?family=Righteous" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Domine" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Caveat" rel="stylesheet">
```

After implementing the theme changes, below you can see the results for both the Dashboard and Visualization Editors.





Using Color Types

You can use either RGB (red, green, blue) or HEX colors to specify the color settings.

```
revealTheme.dashboardBackgroundColor = "rgb(232, 235, 252)";
revealTheme.dashboardBackgroundColor = "#E8EBFC";
```

Built-In Themes

Reveal SDK comes with four pre-built themes: *Mountain Light*, *Mountain Dark*, *Ocean Light*, and *Ocean Dark*. You can set the one that best matches your application's design, or you can also use it as the basis for your custom theme modifications.

Apply the settings of a chosen pre-built theme by using the *UpdateCurrentTheme* method.

Mountain Light Theme

```
$.ig.RevealSdkSettings.theme = new $.ig.MountainLightTheme();
```

Note

Mountain Light contains the default values for the customizable theme settings. This means Mountain Light and the Reveal Theme look basically the same way.

Mountain Dark Theme

```
$.ig.RevealSdkSettings.theme = new $.ig.MountainDarkTheme();
```

Ocean Light Theme

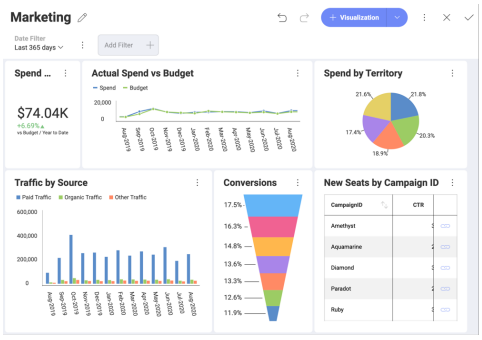
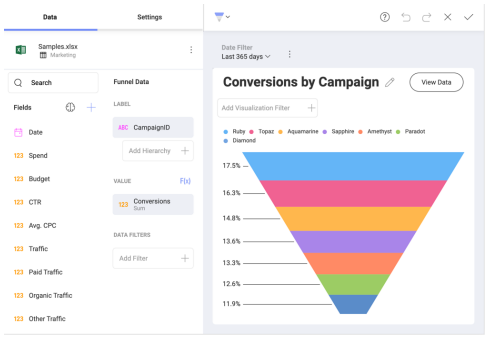
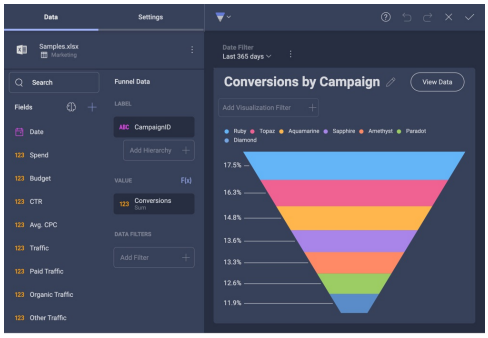
```
$.ig.RevealSdkSettings.theme = new $.ig.OceanLightTheme();
```

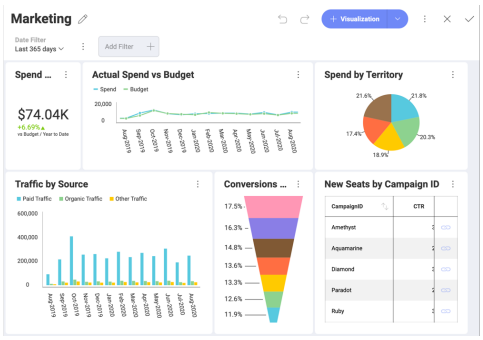
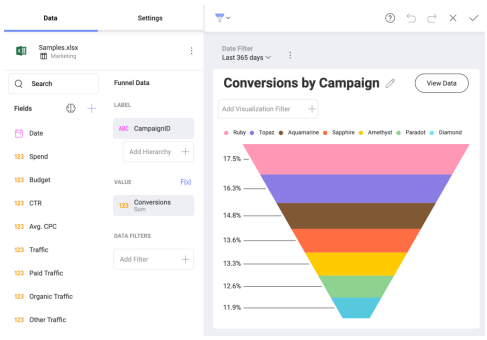
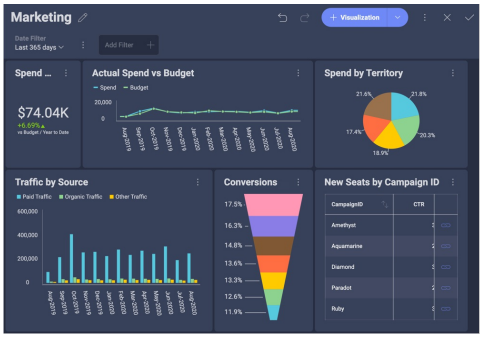
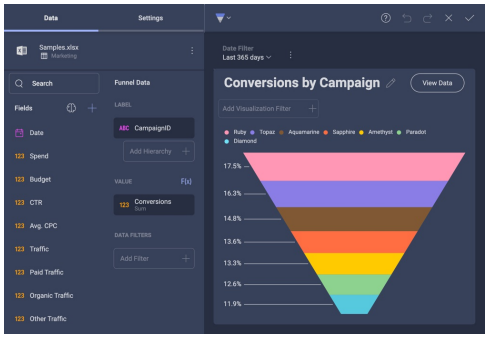
Ocean Dark Theme

```
$.ig.RevealSdkSettings.theme = new $.ig.OceanDarkTheme();
```

How the Built-In Themes Look?

Below, you will find a table showing how the *Visualization Editor* and *Dashboard Editor* look when each of the pre-built themes is applied.

THEME	DASHBOARD EDITOR	VISUALIZATION EDITOR
Mountain Light (Default)		
Mountain Dark		

THEME	DASHBOARD EDITOR	VISUALIZATION EDITOR
Ocean Light		
Ocean Dark		

Handling User Click Events

Overview

The SDK allows you to handle when the user clicks a cell with data within a visualization. This is very useful, for example, to provide your own navigation, to change existing selections in your app, among others.

Code

You can handle user click events by registering to the **onVisualizationDataPointClicked** event:

```
window.revealView.onVisualizationDataPointClicked = function (visualization, cell, row) {
    alert('Visualization clicked: ' + visualization.title() + '', cell: '' + cell.value);
};
```

In the callback function you receive information about the location of the click:

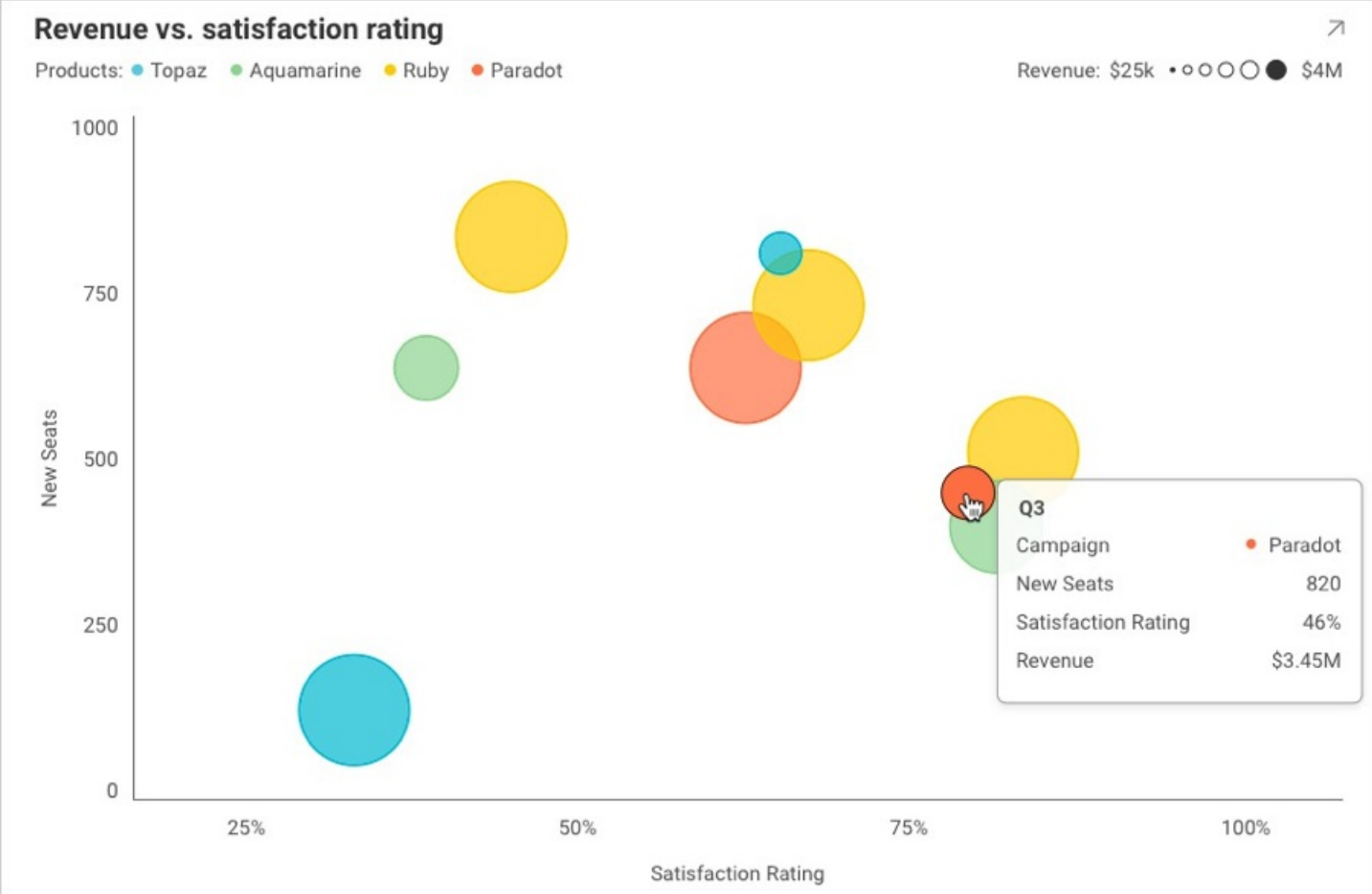
- the name of the visualization clicked;
- the values for the cell clicked (including value, formatted value, and the column’s name);
- the rest of the values in the same cell.

You can use the rest of the values in the cell to search a specific attribute, like customer ID if you want to change the selected customer in your application. Doesn’t matter that the user clicked another cell, like the sales amount for that customer, you’ll still get the information you need.

Working with Tooltips

Overview

There is an event that is triggered whenever the end-user hovers over a series in a visualization or clicks on the series (as shown in the image below). This event called **.onTooltipShowing**, gives you more flexibility regarding how you show Tooltips in your visualizations.



Common Use Cases

You can cancel the Tooltip event or modify what is shown to the user. Most common examples include:

- You want to disable tooltips altogether or only show them for specific visualizations.
- You want to display data in the tooltip that is outside of the RevealView component that might be more valuable to your viewers.

Please note that this event will not be triggered for visualizations that do not support Tooltips, such as grids, gauges, and others.

Code Example

In the following code snippet, you can see how to disable tooltips for a visualization and still get additional information from the event arguments when the end-user hovers over or clicks on this visualization.

```
revealView.onTooltipShowing = function (args) {
    if (args.visualization.title == "NoNeedForTooltips") {
            args.Cancel = true;
    }
    console.log("onTooltipShowing: visualization: " + args.visualization.title() + ",cell: " + args.cell.value + ", row:" + args.row.length);
};
```

The event arguments include information about the visualization that is being hovered over or clicked on, the exact cell of data hovered over or clicked, the whole row of this cell (in case you need information from other columns), and, of course, the Cancel boolean.

Showing/Hiding User Interface Elements

The **$.ig.RevealView** component can be used to enable or disable different features and/or UI elements towards the end user. Many of the available properties are of the Boolean type and can be very straightforward to use, but others not so much.

The *revealView* instance and the DOM element created below are assumed by all the code snippets in this topic:

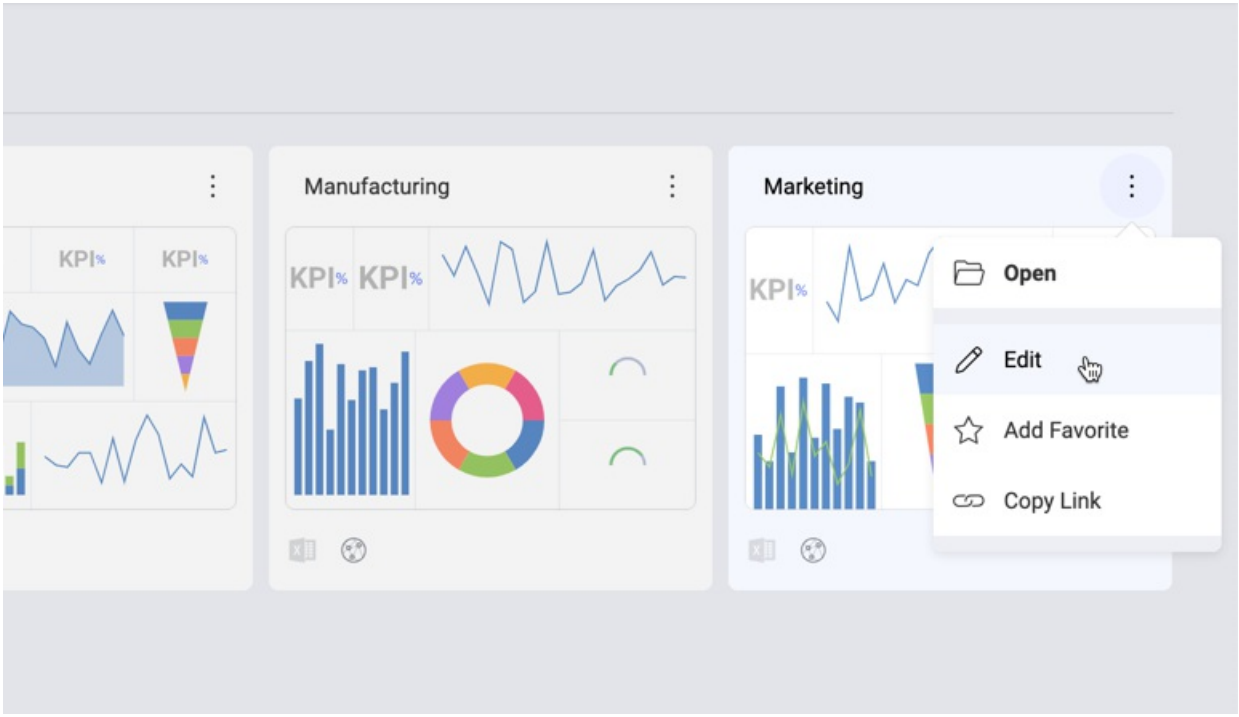
```
var revealView = new $.ig.RevealView("#revealView");
...
<div id="revealView" style="height:500px;" />
```

Note

Depending on your css layout approach you might need the element hosting the RevealView to be "positioned" by setting a position attribute that is not static (like relative or absolute).

canEdit

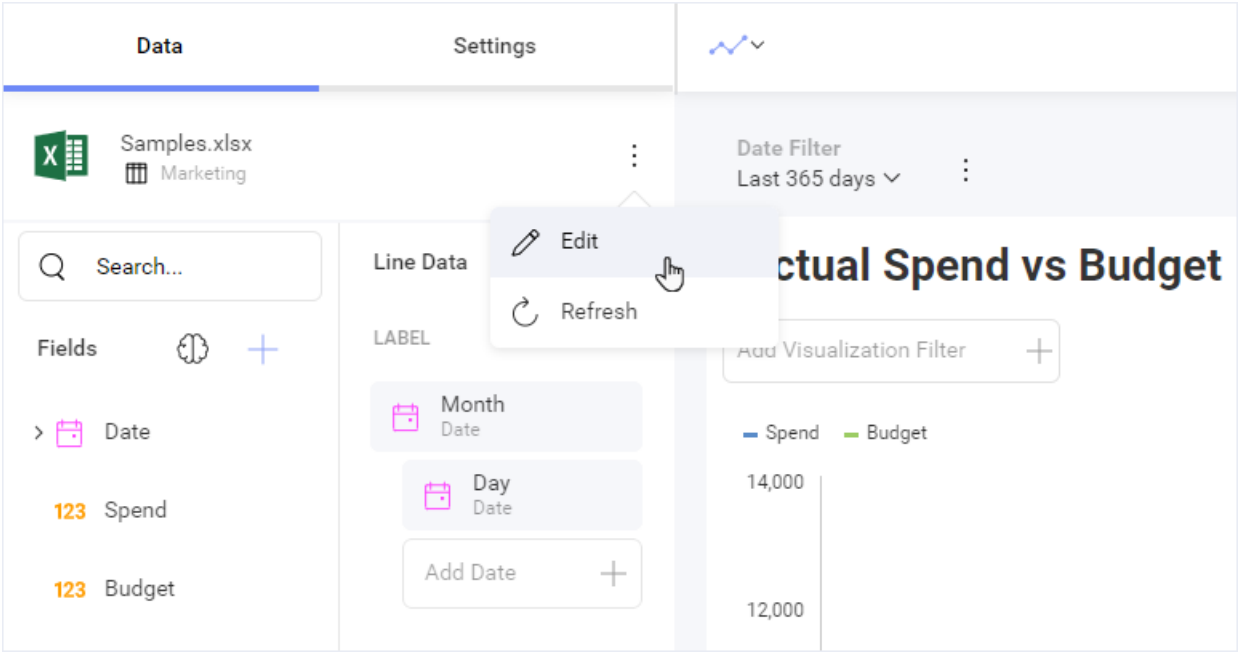
This property can be used to disable the user's ability to edit dashboards.




```
revealView.canEdit = false;
```

showEditDataSource

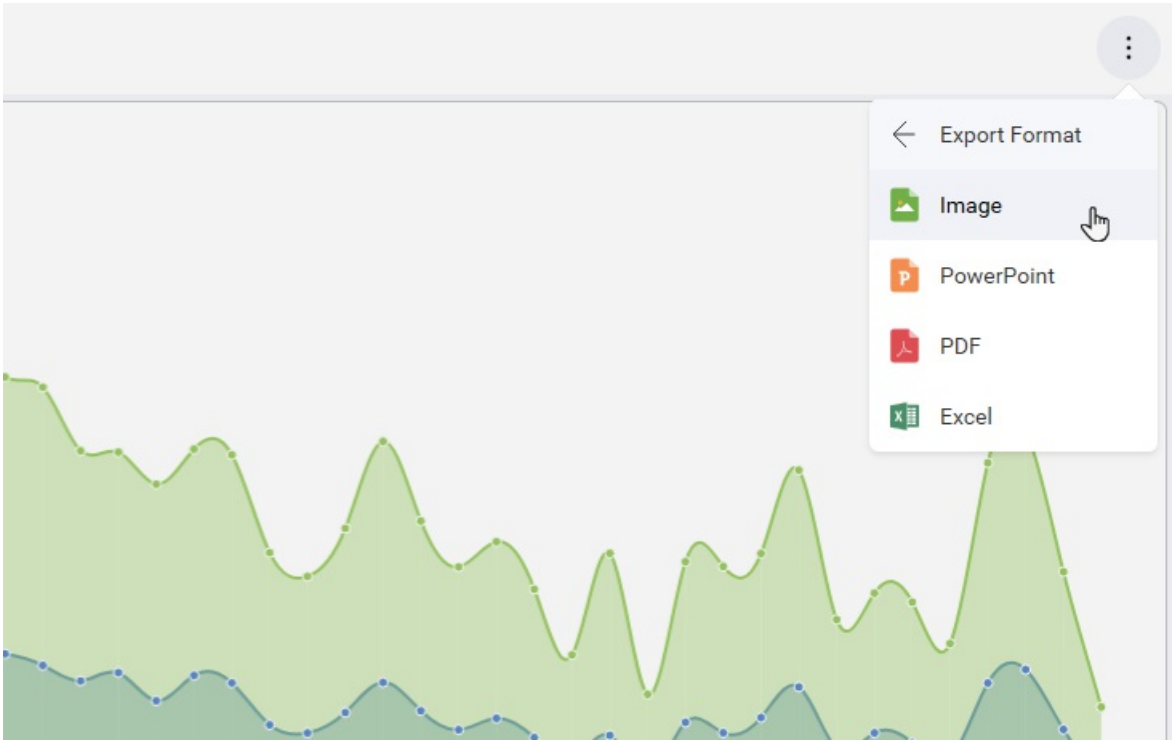
This property can be used to disable the editing of a dashboard datasource.




```
revealView.showEditDataSource = false;
```

showExportImage

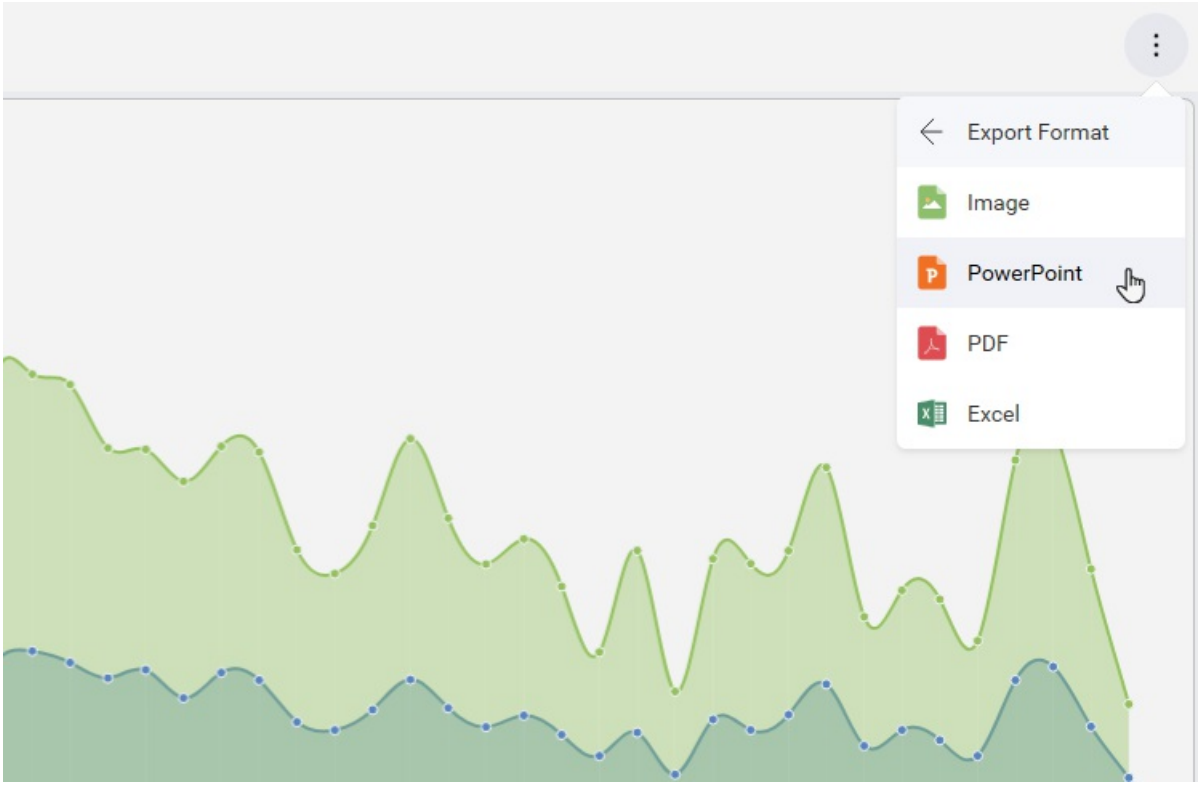
This property can be used to disable exporting the dashboad to an image.




```
revealView.showExportImage = false;
```

showExportToPowerpoint

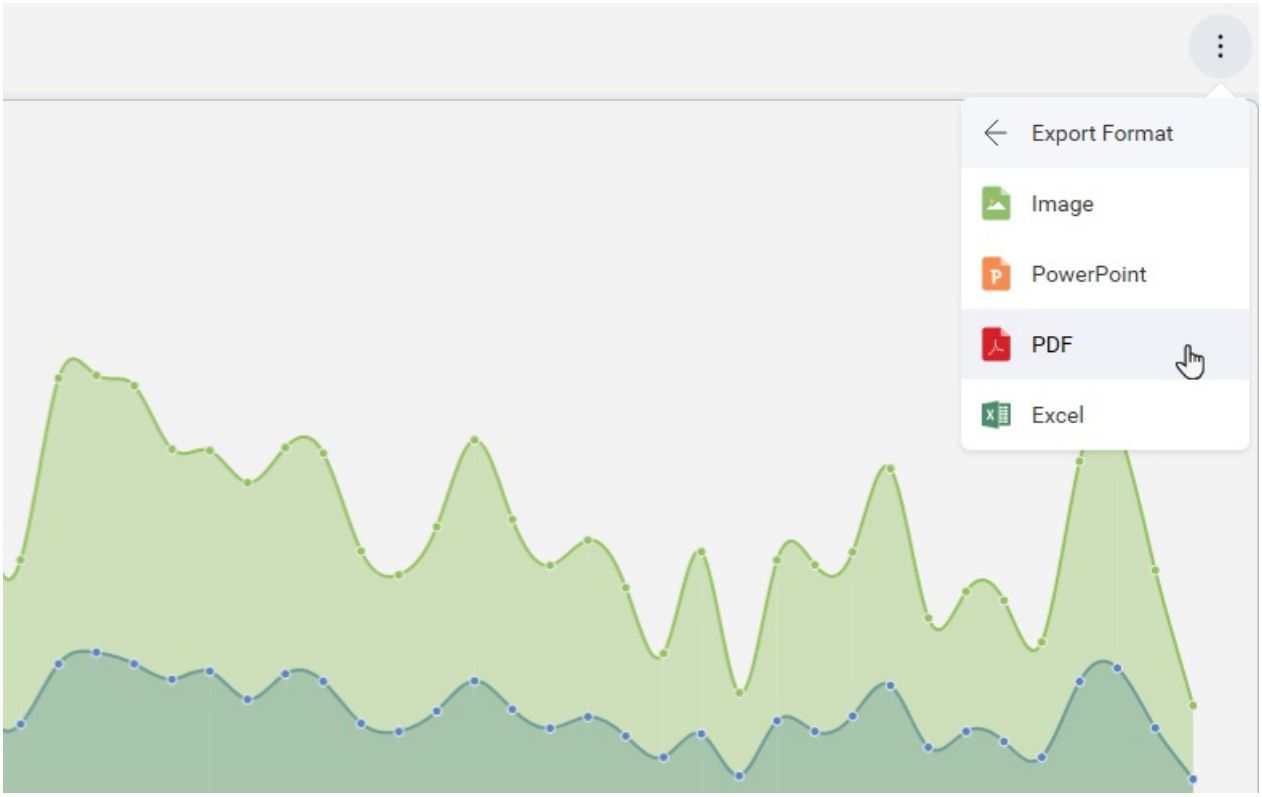
This property can be used to disable exporting the dashboad to PowerPoint.




```
revealView.showExportToPowerpoint = false;
```

showExportToPDF

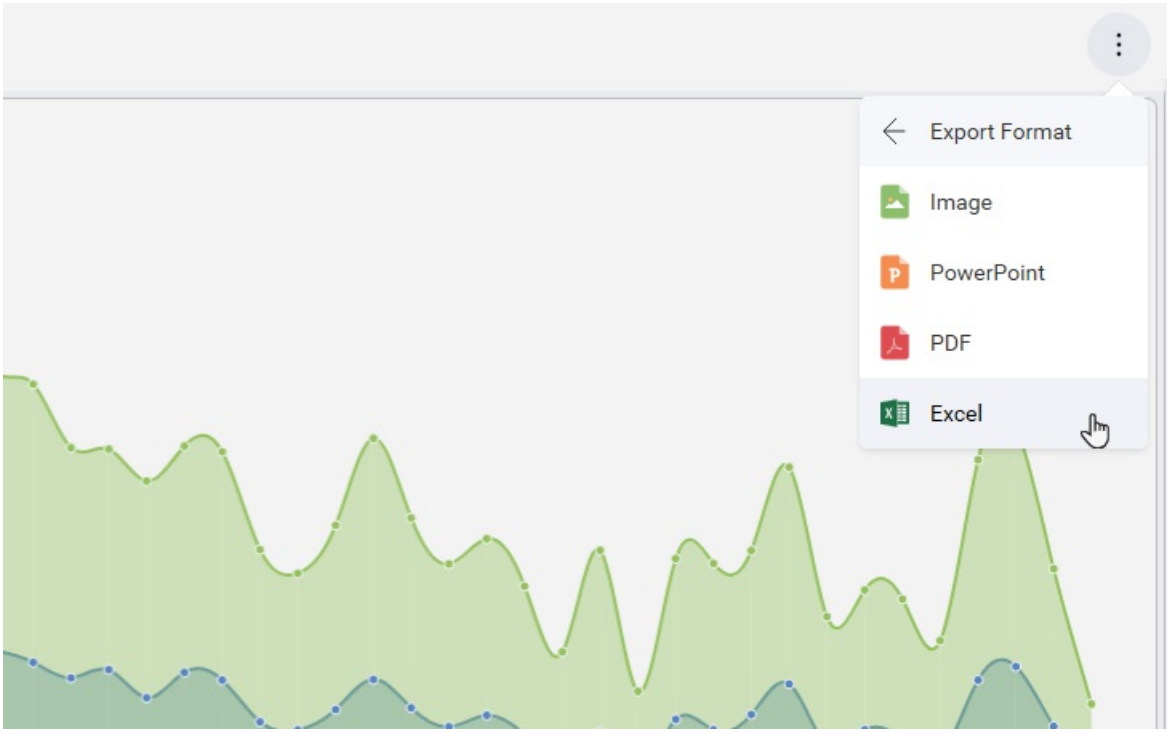
This property can be used to disable exporting the dashboad to PDF.




```
revealView.showExportToPDF = false;
```

showExportToExcel

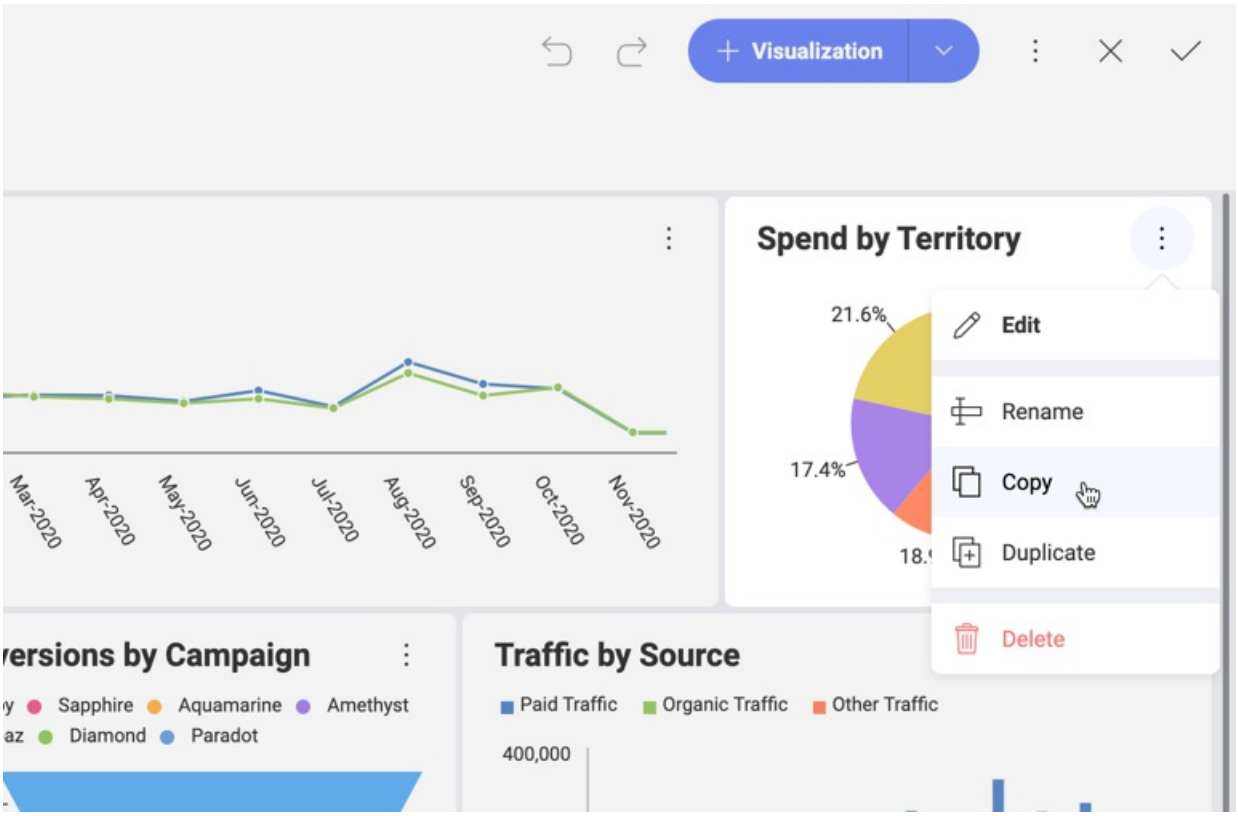
This property can be used to disable exporting the dashboad to Excel.




```
revealView.showExportToExcel = false;
```

canCopyVisualization

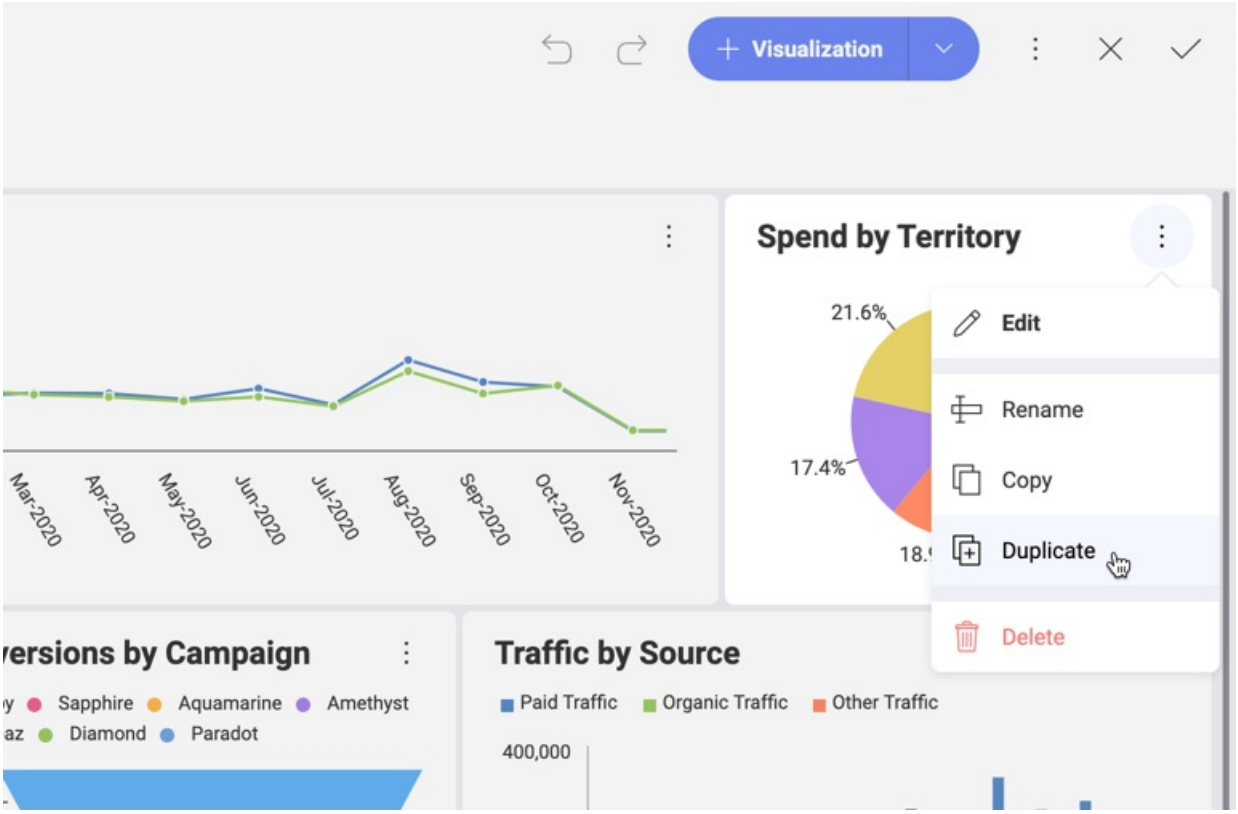
This property can be used to disable the ability to copy a visualization and later paste it in the current dashboard or a different one.




```
revealView.canCopyVisualization = false;
```

canDuplicateVisualization

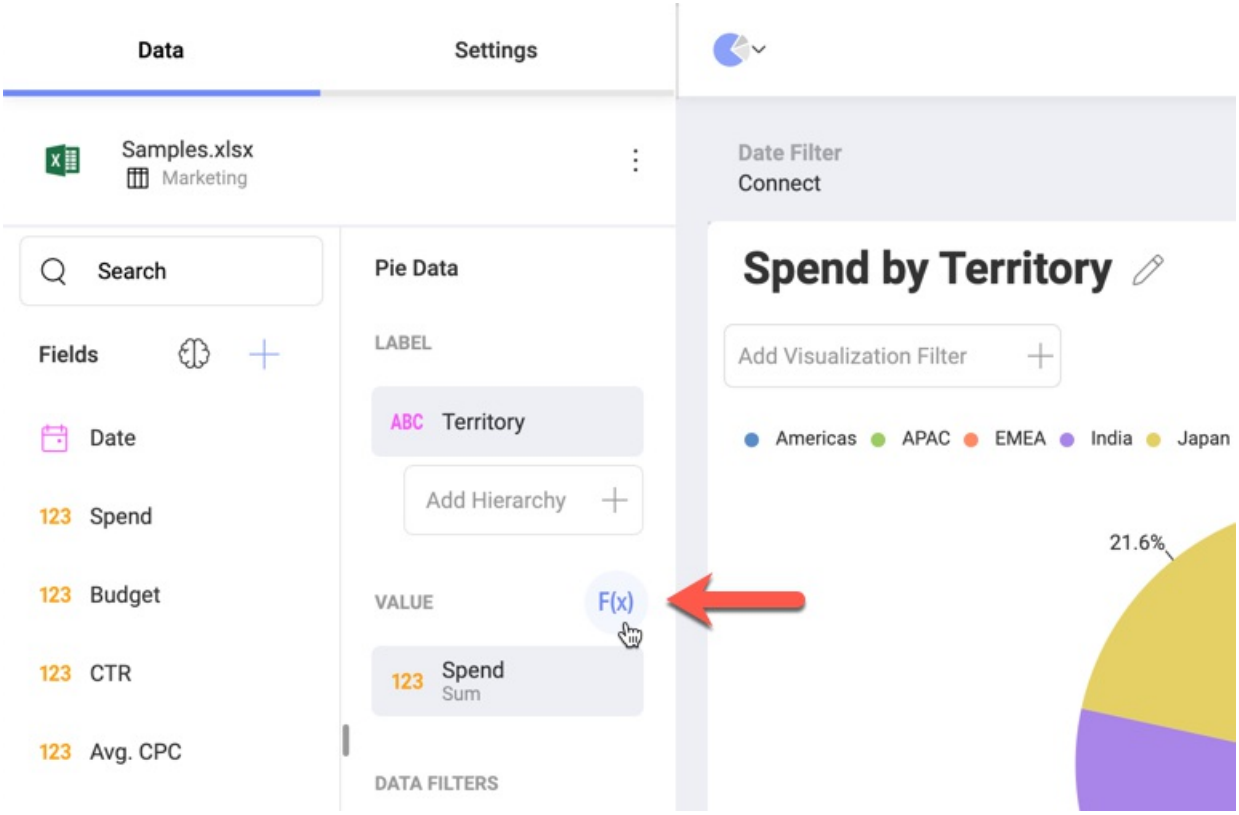
This property can be used to disable the ability to duplicate a visualization in the current dashboard.




```
revealView.canDuplicateVisualization = false;
```

canAddPostCalculatedFields

This property can be used to disable the ability to add a new post-calculated field in the current dashboard.

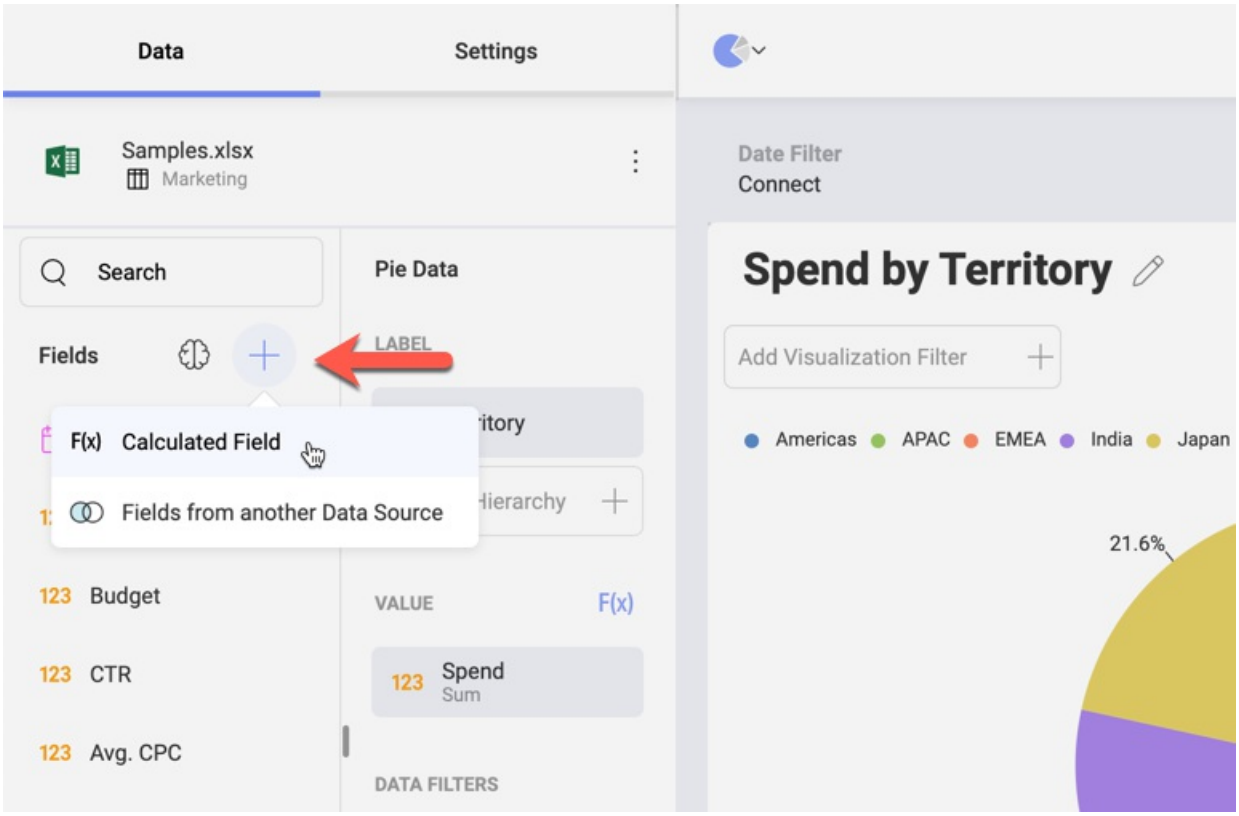


Post-calculated fields are new fields in the data set and are created by applying a formula on already summarized values.
For further details, please refer to the Reveal Help.

```
revealView.canAddPostCalculatedFields = false;
```

canAddCalculatedFields

This property can be used to disable the ability to add a new pre-calculated field in the current dashboard.

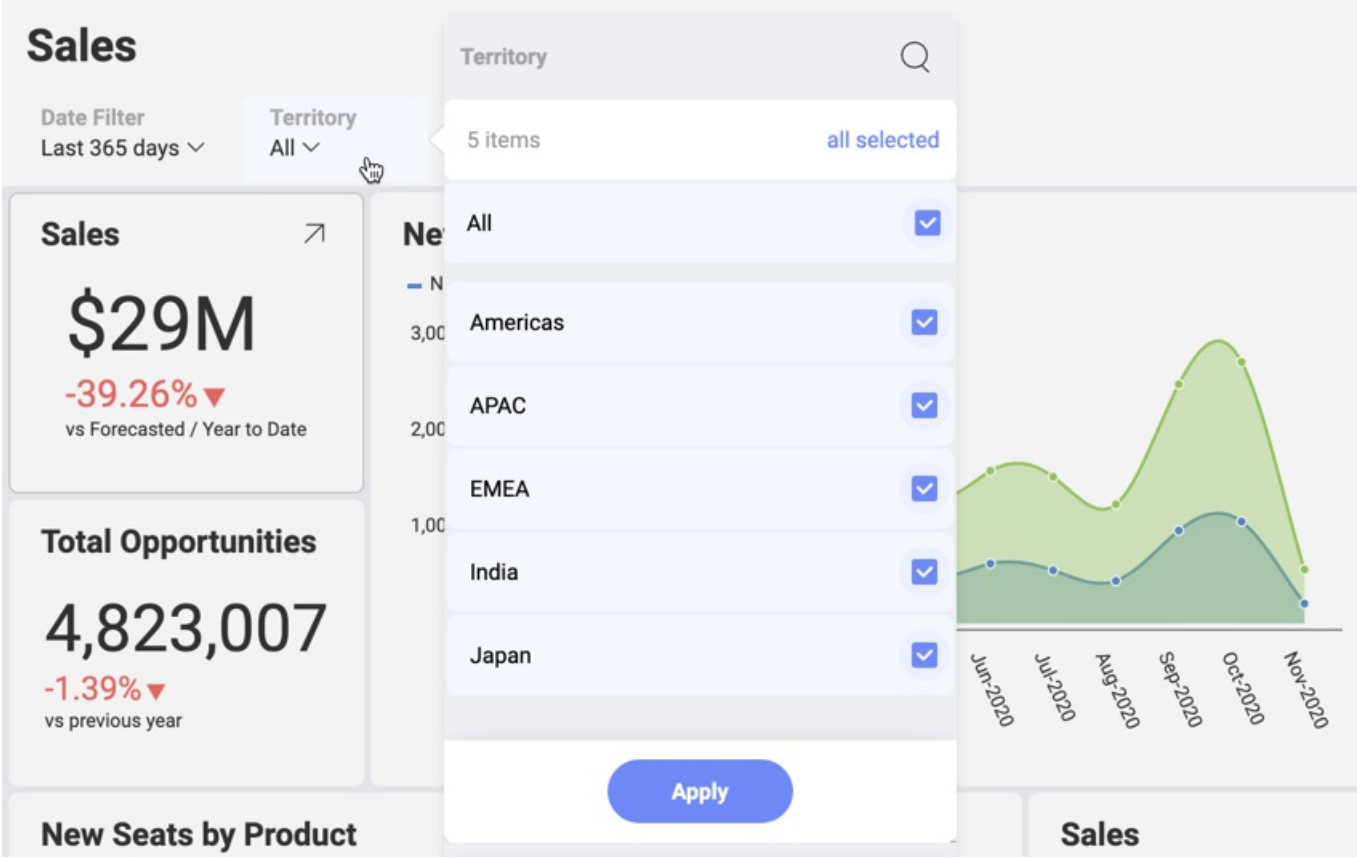


Pre-calculated fields are new fields in the data set and are evaluated before executing data editor aggregations.
For further details, please refer to the Reveal Help.

```
revealView.canAddCalculatedFields = true;
```

showFilters

This property can be used to show or hide the Dashboard Filters UI to the user.

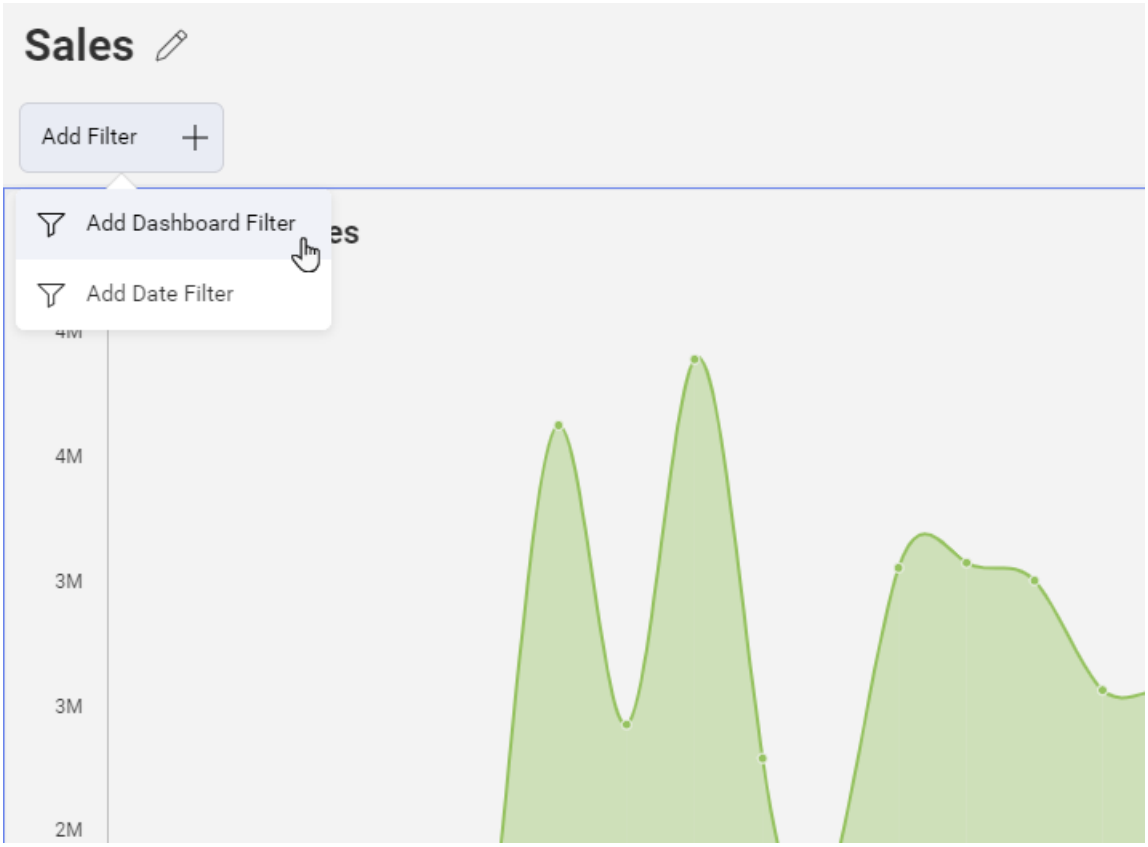


Dashboard filters allow you to slice the contents of the visualizations in a dashboard, all at once.

```
revealView.showFilters = true;
```

canAddDashboardFilter

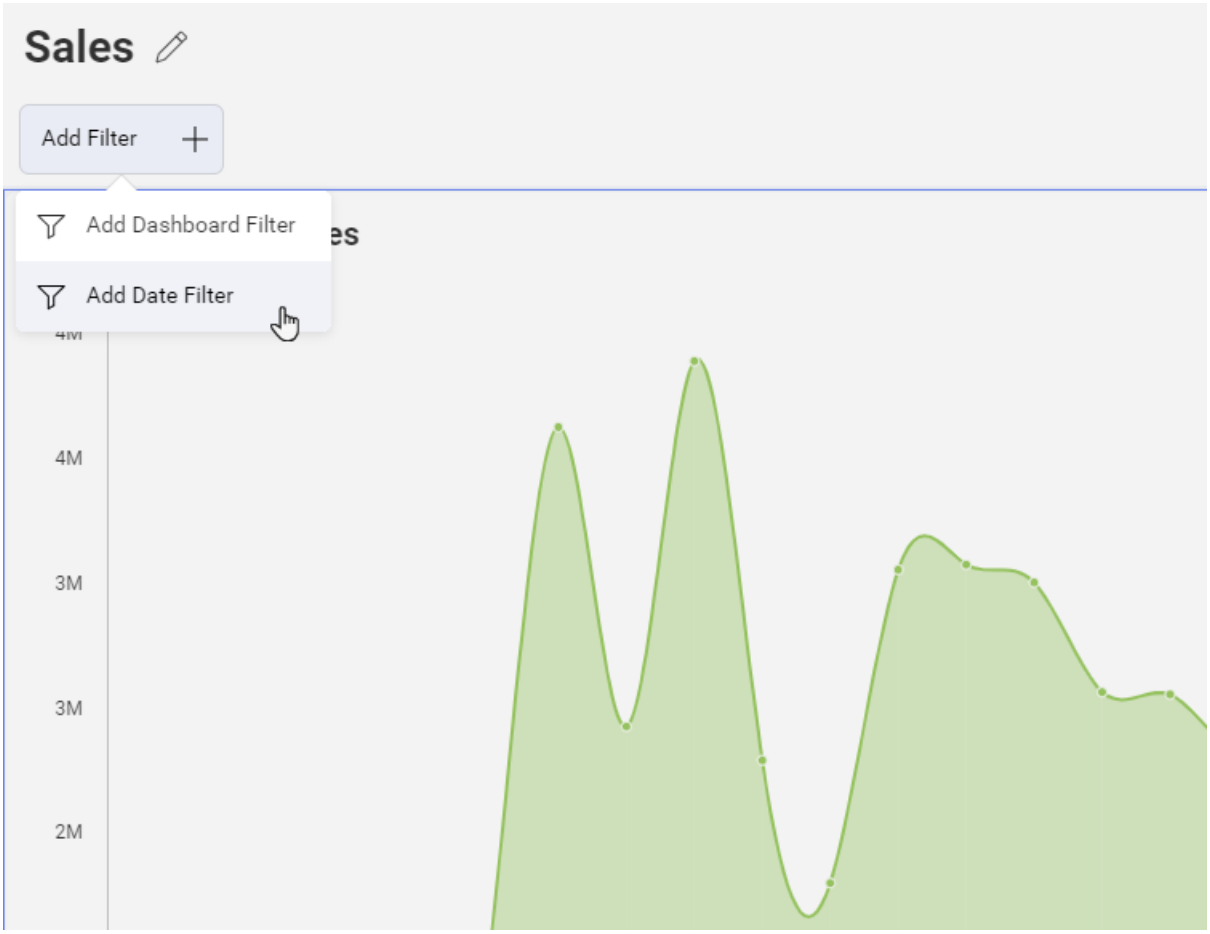
This property can be used to show or hide the Add Dashboard Filter menu item.



```
revealView.canAddDashboardFilter = false;
```

canAddDateFilter

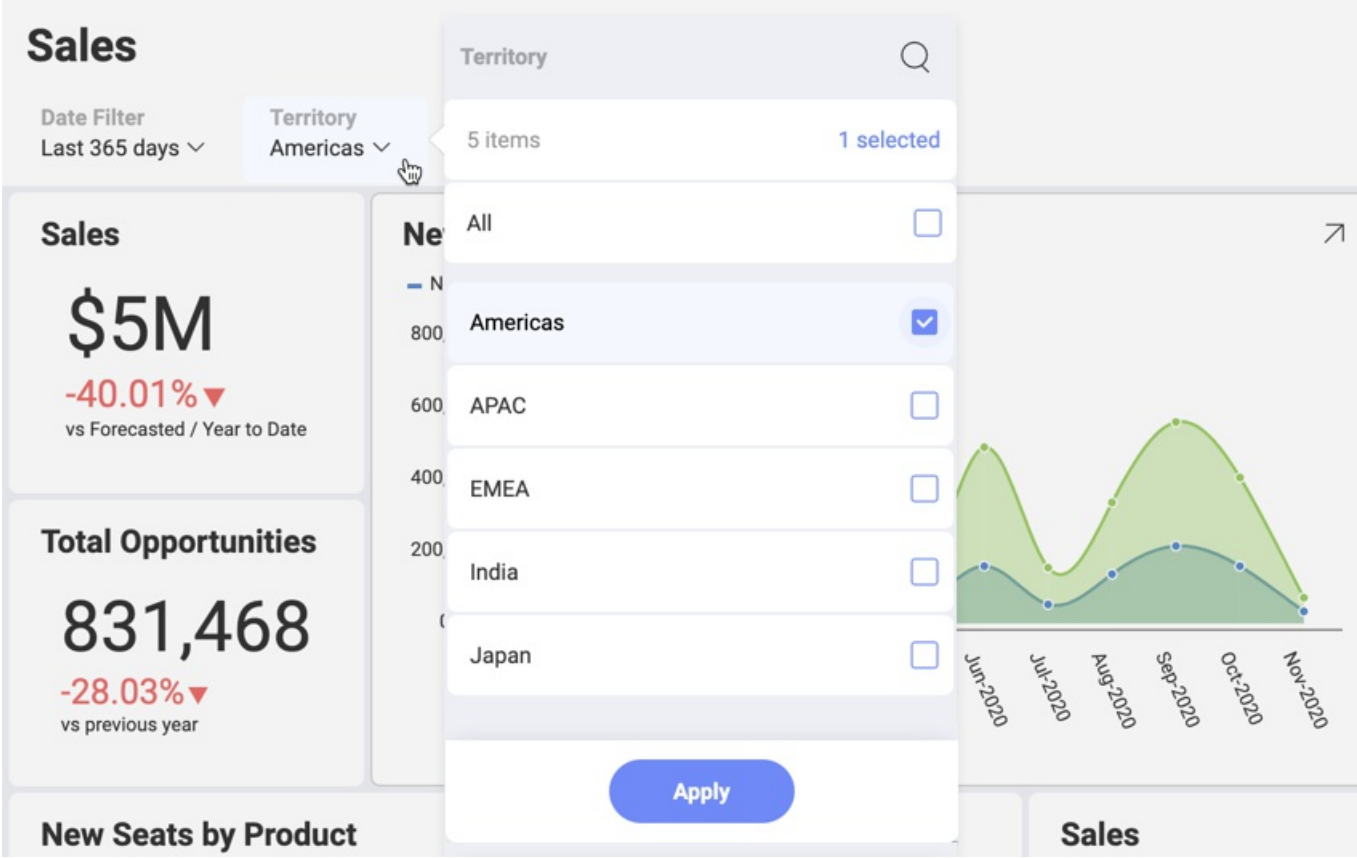
This property can be used to show or hide the Add Date Filter menu item.




```
revealView.canAddDateFilter = false;
```

Preselected Filters

You can specify which values are initially selected among existing Dashboard Filters when loading a dashboard.

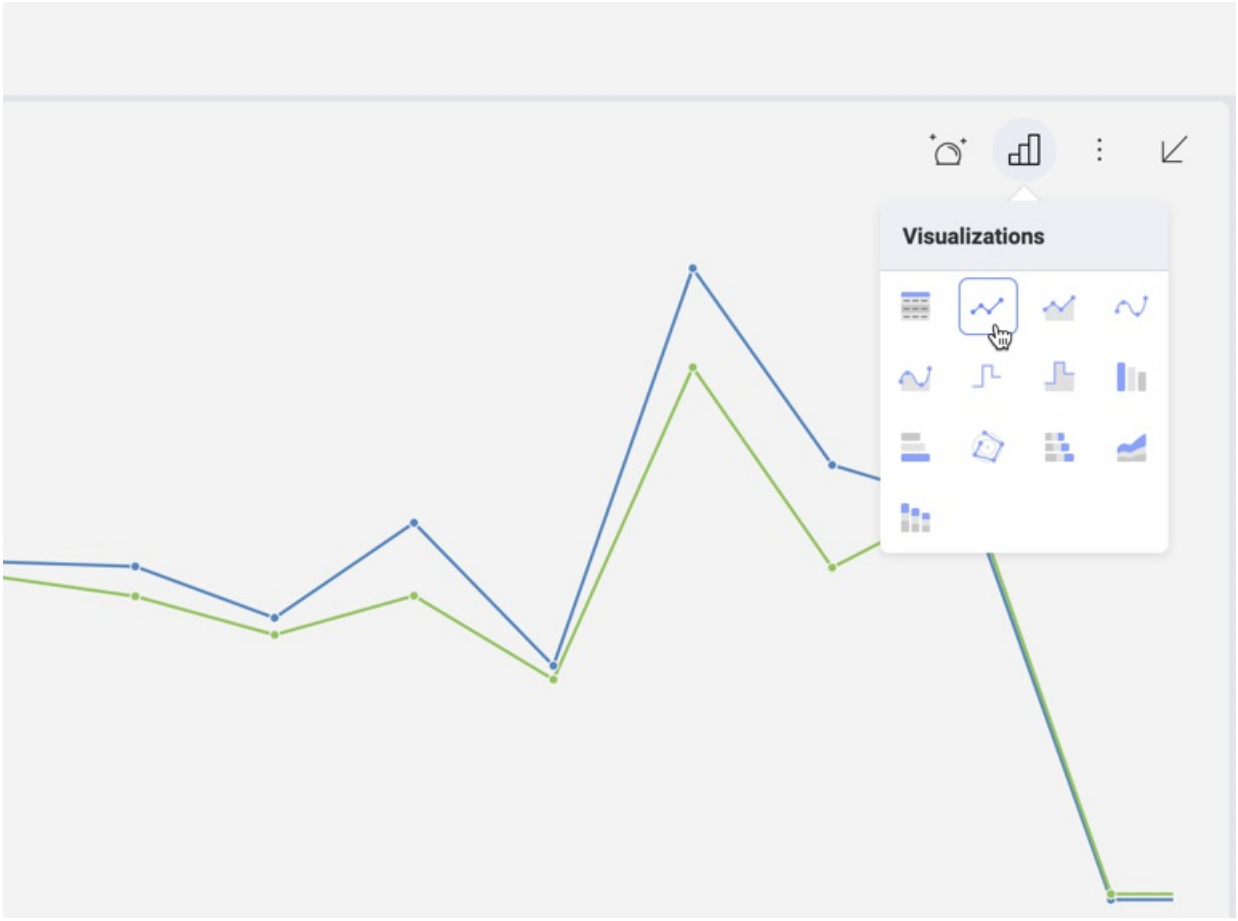


The following code snippet illustrates how to load a dashboard “AppsStats”. By setting the “Territory” dashboard filter’s selected value to be “Americas”, the dashboard will be showing data filtered by “Americas”.

```
var dashboardId = "AppsStats";

$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    dashboard.filters.getByTitle("Territory").selectedValues = ["Americas"];

    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
}, function (error) {
});
```

availableChartTypes

This property can be used to filter the visualization types available to the user.



You can, for example, add or remove visualizations as shown below:

```
revealView.availableChartTypes.add($.ig.RVChartType.bulletGraph);
revealView.availableChartTypes.remove($.ig.RVChartType.choropleth);
```

In addition, you can use a brand new Array that includes only the visualizations you want to be available:

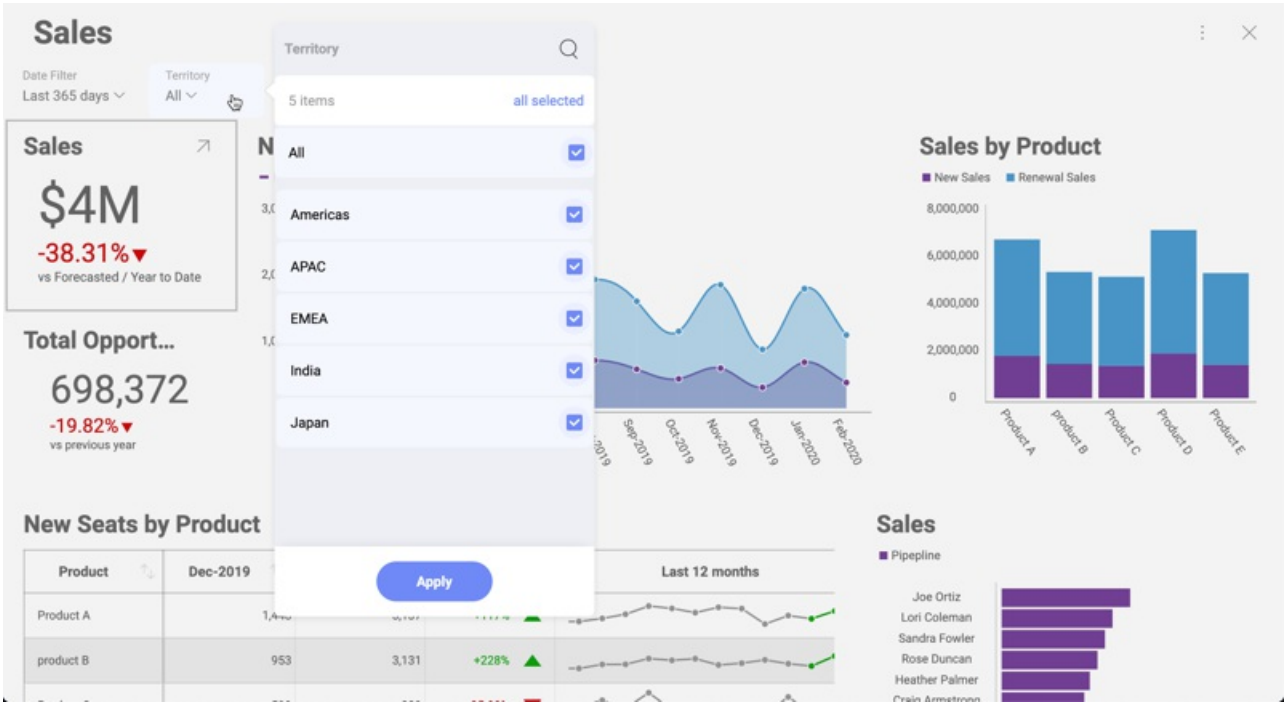
```
revealView.AvailableChartTypes = [$.ig.RVChartType.bulletGraph, $.ig.RVChartType.choropleth];
```

Setting up Dynamic Filter Selections

Overview

Sometimes your application will integrate a custom UI, to present the user with a list of values to select. And you might want that user selection to be synchronized with a filter in the dashboard.

For example, you can have a Sales dashboard that changes figures based on the current Territory and a custom UI to select the Territory. After the user selection changes, you'd want the Sales dashboard to reflect that change. Most of the times, you would hide the filter selection normally displayed in the dashboard. This way the user won't be confused with two different ways to change the Territory in the screen.



In the following code snippet, you'll find details about how to achieve the described scenario:

```
<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");
        revealView.showFilters = false;
        revealView.dashboard = dashboard;
    }, function (error) {
        console.log(error);
    });
    function setSelectedTerritory(territory) {
        window.revealView.dashboard.filters.getByTitle('Territory').selectedValues = [territory];
    }
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto auto auto;">
        <button onclick="setSelectedTerritory('Americas')">Americas</button>
        <button onclick="setSelectedTerritory('APAC')">APAC</button>
        <button onclick="setSelectedTerritory('EMEA')">EMEA</button>
        <button onclick="setSelectedTerritory('India')">India</button>
        <button onclick="setSelectedTerritory('Japan')">Japan</button>
    </section>
    <div id="revealView" style="height:500px;" />
</section>
```

As shown above, five buttons were added at the top of the dashboard, one button for each of the territories in the dashboard: Americas, APAC, EMEA, India and Japan.

Working with Dynamic Lists

Territories like Americas, APAC, India, etc. do not change over time, but other lists of values might change. In this case, if a new Territory is added to the list, a new button will not be automatically added.

You can use **$.ig.RVDashboardFilter.getFilterValues** method to get the list of values for a given filter, in this case the following call will leave an array with five **$.ig.RVFilterValue** objects in _window.territories</emphasis>:

```
var filter = window.revealView.dashboard.getByTitle('Territory');
filter.getFilterValues(function (values) {
    window.territories = values;
}, function (error) {
    console.log(error);
});
```

You can then use the *label* attribute from **$.ig.RVFilterValue** to display the name of the territory and the **values** attribute to set the selection in the filter. The following code snippet shows how to populate a ComboBox to automatically select the Territory:

```
<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashaboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");
        revealView.showFilters = false;
        revealView.dashboard = dashboard;

        var filter = revealView.dashboard.filters.getByTitle('Territory');
        filter.getFilterValues(function (values) {
            window.territories = values;
            var buttonsPanel = $('#buttonsPanel')[0];
            for (var i = 0; i < values.length; i++) {
                var button = $('<button onclick="setSelectedTerritory(window.territories[' + i + '].values)">' + values[i].label + '</button>');
                buttonsPanel.append(button[0]);
            }
        }, function (error) {
            console.log(error);
        });
    }, function (error) {
        console.log(error);
    });
    function setSelectedTerritory(territory) {
        var filter = window.revealView.dashboard.getByTitle('Territory');
        filter.selectedValues = [territory]);
    }
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto auto auto;" id="buttonsPanel">
    </section>
    <div id="revealView" style="height:500px;" />
</section>
```

As shown above, the section containing the buttons is assigned with the “buttonsPanel” id. Then, JQuery is used to dynamically create the buttons and append them to the DOM document.

The variable *window.territories* holds the list of territories, and is later used when selecting the values in the “onclick” code for each button.

Setting Up Initial Filter Selections

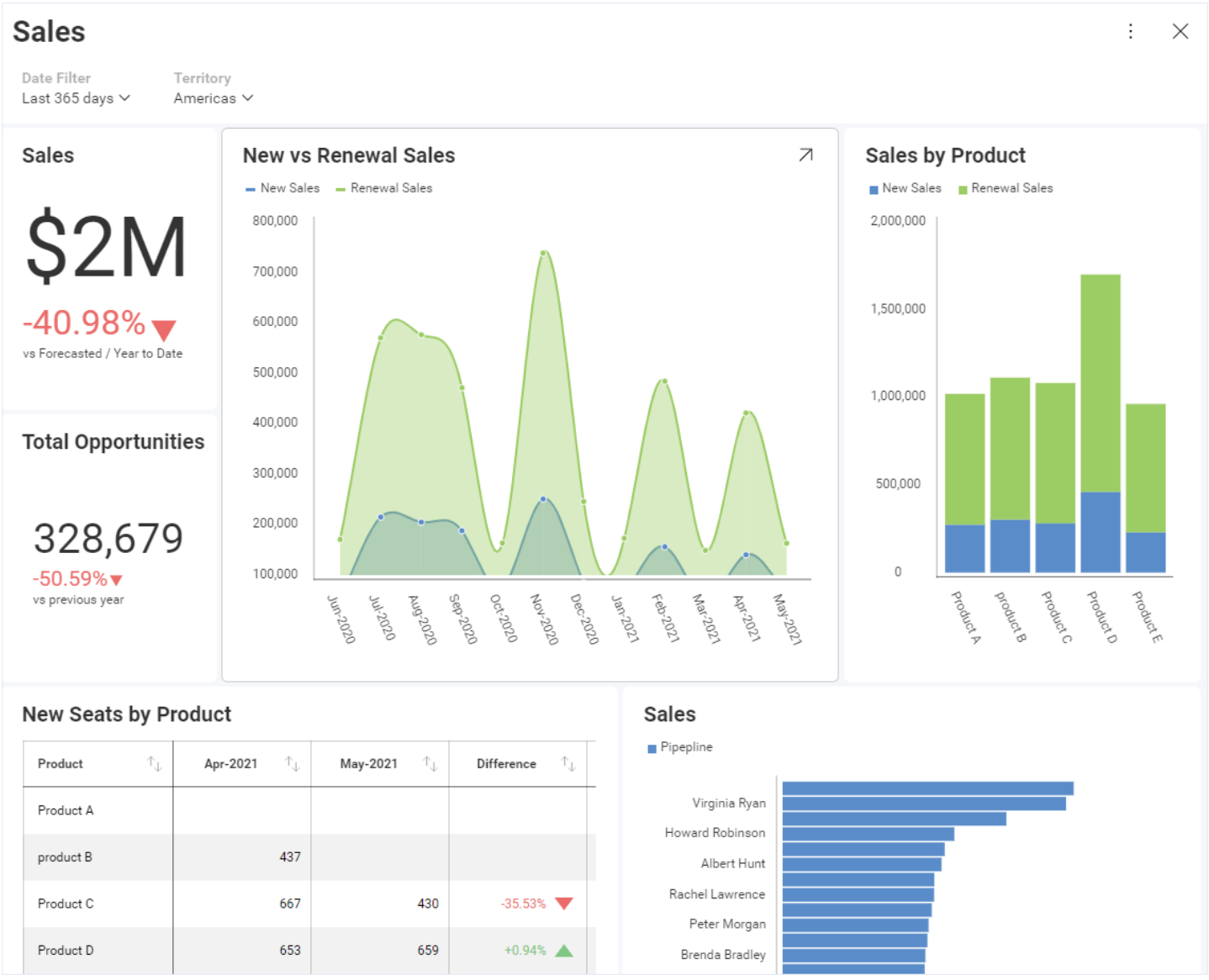
Overview

Sometimes, you want to display a dashboard with filters already applied. Dashboard filters are very useful to slice the contents of all the visualizations at once. Because of this, you can use the SDK to set up initial dashboard filter selections that remain in context for all the dashboard's visualizations.

Example Details

In this example, you have a dashboard showing Sales data with the following filters:

- A given period of time (last 365 days, Year to Date, etc.);
- Territory (Americas, Europe, Asia, etc.).



Code Example

In this case, you want to set the initial filters selection to:

- "Year to Date" (instead of "Last 365 days", the default setting for this dashboard);
- Sales associated to the Territory of the current user.

As part of the initialization process and once the dashboard is loaded, you can retrieve the list of filters in the dashboard and use these filters to set the selected values though the dashboard object and finally assign it to the revealView's dashboard property:

```
<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {

      dashboard.filters.getByTitle("Territory").selectedValues = [getCurrentUser().territory];
      dashboard.dateFilter = new $.ig.RVDateDashboardFilter($.ig.RVDateFilterType.YearToDate);

      var revealView = new $.ig.RevealView("#revealView");
      revealView.dashboard = dashboard;

    }, function (error) {
        console.log(error);
    });
</script>

<div id="revealView" style="height:500px;" ></div>
```

Note

The code above assumes that **getCurrentUser().territory** returns the territory for the current user.
Setting the initally selected value in the date filter client-side is only supported when the dashboard **already has a date filter** created (in the .rdash file that you use).

Hiding filters

It is possible that you might not want users to access data from territories different than their own. In that case, you can restrict the access to filters by configuring the **$.ig.RevealView** object to hide the panel containing the dashboard filters:

```
revealView.showFilters = false;
```

That setting will restrict users to see data only for their associated territory.

Finally, in the case that you still want users to change the date filter selection, take a look at **Setting up Dynamic Filter Selections**. There you'll find information about how to create your own UI that, allowing the user to change the date filter.

Web Server .NET API Reference

Here you will find technical information about Reveal SDK, specifically about the Web Server .NET API. For a complete reference, please follow the link

Most commonly used classes and interfaces:

Main SDK concepts and features
RevealSdkContextBase
RevealEmbedSettings
Dashboard class

Datasources
IRVDataSourceProvider

RVSqlServerDataSource
RVSqlServerDataSourceItem

RVRESTDataSource
RVRESTDataSourceItem

RVJsonDataSource
RVJsonDataSourceItem

IRVDataProvider
RVInMemoryData
RVInMemoryDataSource
RVInMemoryDataSourceItem

Authentication
IRVAuthenticationProvider
IRVDataSourceCredential
RVBearerTokenDataSourceCredential
RVUsernamePasswordDataSourceCredential

Web Client JS API Reference

Here you will find technical information about Reveal SDK, specifically about the Web Client JavaScript API. For the complete reference, please follow the link

Most commonly used classes and interfaces

Main SDK concepts and features

RevealView
RVDashboard
RevealSdkSettings
RVVisualization
RevealTheme
RevealUtility

Datasources

RVSqlServerDataSource
RVSqlServerDataSourceItem

RVRESTDataSource
RVRESTDataSourceItem

RVJsonDataSource
RVJsonDataSourceItem

RVInMemoryDataSource
RVInMemoryDataSourceItem

Filtering

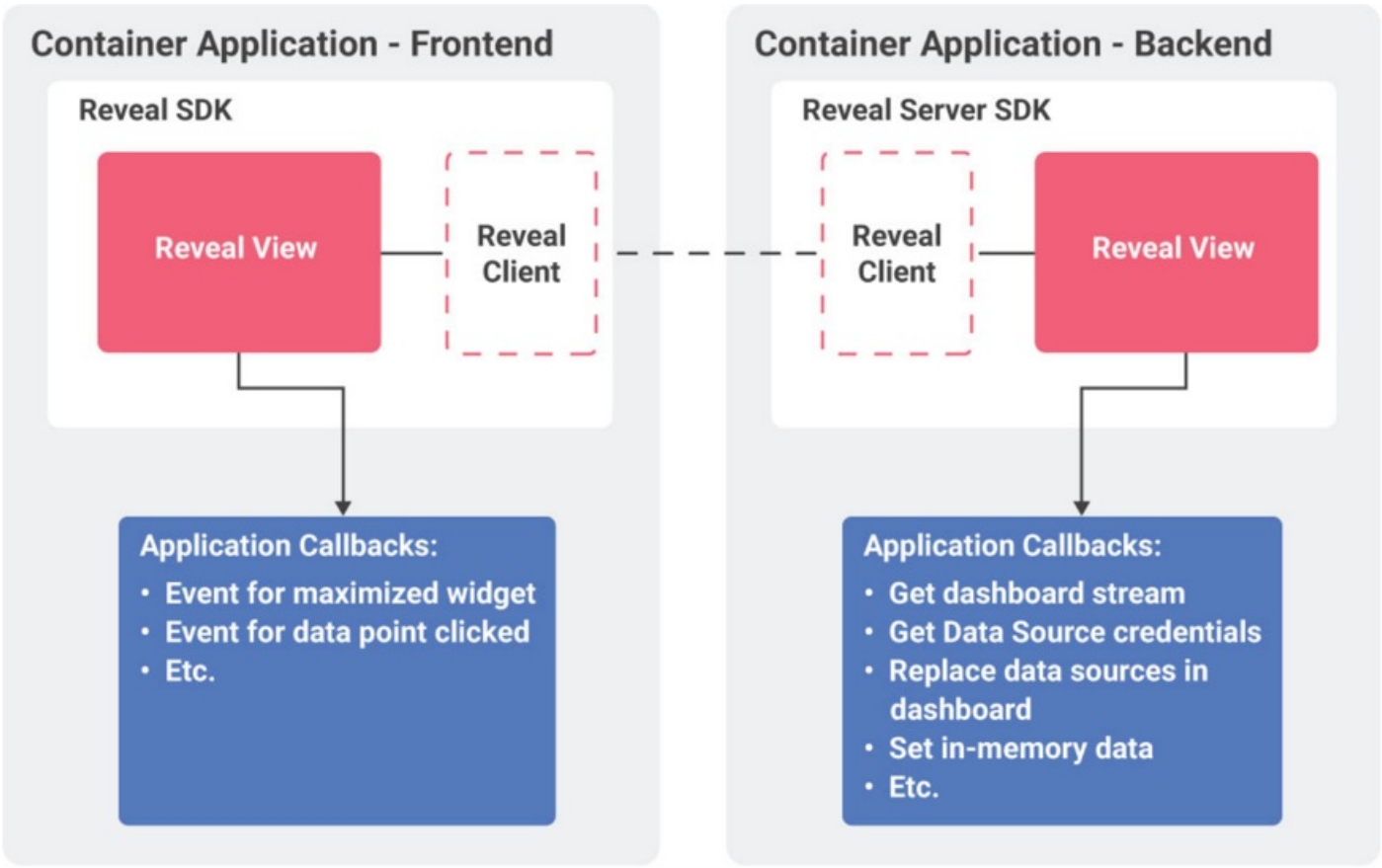
RVDateDashboardFilter
RVDashboardFilter
RVFilterValue

Overview

When embedding Reveal into web applications, the architecture is slightly more complex than with native apps, as two components are always involved:

- **Reveal Client SDK**: a set of JavaScript libraries that needs to be integrated into the web application. The frameworks supported today are: jQuery, Angular and React.
- **Reveal Server SDK**: the server-side component to be integrated into the server application, current libraries require JAVA SDK 11+ and are distributed as a set of Maven modules.

In the following diagram you visualize the architecture for a web application embedding Reveal Web SDK:



As shown above, the SDK works pretty much the same way as with native apps. The difference is that some of the callbacks are invoked in the client side (like the event sent when a data point is clicked) and others are invoked server side (like the callback to load the dashboard or to provide in-memory data).

Hosting the Client-side and Server-Side Parts on Different Servers

You can host the client-side and the server-side parts separately i.e. on different urls.

To achieve this, set a property on the window object, as shown below:

```
$.ig.RevealSdkSettings.setBaseUrl("http://localhost:8080/upmedia/reveal-api");
```

Please note that the format used is: **http:**//[server]**:**[port]/[application]**/reveal-api**. The inclusion of **/reveal-api** is required in order to set the property successfully.

Set this property *prior to the instantiation of the $.ig.RevealView*.

Running the UpMedia Samples

After getting the UpMedia samples from GitHub, below you have detailed information to help you run them.

UpMedia Sample Application in Tomcat

Requirements

- Java SDK 11.0.10 and up recommended.
- Tomcat 9.0.41 and up recommended.
- Eclipse for Enterprise Java Developers version 2020-12 and up recommended.
- Maven repository and dependency already added. For details, please refer to Setup and Configuration.

Steps

1. **Load the Project**

    1. In Eclipse, go to: *File > Import > Existing Projects into Workspace*.
    2. In *Select root directory*, choose Reveal's SDK path and also select the *UpMedia* Project.
    3. Check the *Copy projects into workspace* option.
    4. Click "Finish".

        The project is now loaded and ready to be inspected.

2. **Run the Project in Tomcat**

    1. In the UpMedia Project, right click *Run on Server*.
    2. Choose Tomcat 9 and configure the root path to Tomcat's installation path.
    3. Go through the default settings and run the Project.

3. **Visualize the Sample in a Browser**

    - Eclipse's internal browser has known issues with Windows, instead please try Google Chrome or another browser (the default URL will be http://localhost:8080/upmedia).

Alternative steps to work with a WAR file:

1. Go to *Run As -> Maven build*.
2. Use the *package* goal.
3. Take the WAR file created and manually deploy it to Tomcat.

UpMedia React Sample with Spring Backend

Requirements

- Java SDK 11.0.10 and up recommended.
- NodeJS 14.15.4 and up recommended, NPM version: 6.14.10 and up.
- Maven repository and dependency already added. For details, please refer to Setup and Configuration.

Steps

1. **Run the Spring Application**

    1. Locate the Spring Boot application in the *upmedia-backend-spring* folder.
    2. Run the sample application by executing the folllowing command:

```
mvn spring-boot:run
npm start
```

3. Verify the server by accessing http://localhost:8080/upmediabackend/reveal-api/DashboardFile/Sales. As a result you'll get the JSON document for the Sales sample dashboard. Reveal services can be found under */upmediabackend/reveal-api/*.

2. **Run the React Application**

    1. Locate the React application in the *upmedia-react* folder.

    2. Run the React application as usual:

    ```
    npm install
    npm start
    ```

    3. Access your React application at http://localhost:3000.

There you can find a few components to try:

- **Filters**: shows how to open an existing dashboard and customize the filtering experience.
- **Linking**: shows how to open an existing dashboard with a link to another dashboard and how to properly configure the linking.
- **CreateDashboard**: shows how to open the dashboard editor to create a new dashboard from scratch. In addition, it also shows how to setup the list of data sources that will be displayed to the user when creating a new visualization.

For details about Reveal's Web client SDK, please refer here.

UpMedia React sample with Tomcat backend

Requirements

- Java SDK 11.0.10 and up recommended.
- Tomcat 9.0.41 and up recommended.
- Eclipse for Enterprise Java Developers version 2020-12 and up recommended.
- Maven repository and dependency already added. For details, please refer to Setup and Configuration.

Steps

1. **Load the Project**

    1. In Eclipse, go to: *File > Import > Existing Projects into Workspace*.
    2. In *Select root directory*, choose Reveal's SDK path and also select the *upmedia-backend-tomcat* Project.
    3. Check the *Copy projects into workspace* option.

    4. Click "Finish".

        The project is now loaded and ready to be inspected.

2. **Run the Project in Tomcat**

    1. In the upmedia-backend-tomcat Project, right click *Run on Server*.
    2. Choose Tomcat 9 and configure the root path to Tomcat's installation path.
    3. Go through the default settings and run the Project.
    4. Verify the server by accessing http://localhost:8080/upmediabackend/reveal-api/DashboardFile/Sales. As a result you'll get the JSON document for the Sales sample dashboard.

3. **Run the React Application**

    1. Locate the React application in the *upmedia-react* folder.

2. Run the React application as usual:

```
npm install
npm start
```

3. Access your React application at http://localhost:3000.

There you can find a few components to try:

- **Filters**: shows how to open an existing dashboard and customize the filtering experience.
- **Linking**: shows how to open an existing dashboard with a link to another dashboard and how to properly configure the linking.
- **CreateDashboard**: shows how to open the dashboard editor to create a new dashboard from scratch. In addition, it also shows how to setup the list of data sources that will be displayed to the user when creating a new visualization.

For details about Reveal's Web client SDK, please refer here.

Setup and Configuration

Prerequisites (Maven)

Reveal Java SDK is distributed as a set of Maven modules. To work with the SDK libraries, you need to add a reference to Reveal's Maven Repository and also a dependency in your Maven pom.xml file.

Add the following repository:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

And the following dependency:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **0.9.6**.

If you are not familiar with Maven, please refer to the following link.

Note

If you work with an Oracle Database you need to add the driver to your application.

Setup and Configuration (Generic Server)

To integrate Reveal with any existing application, you need to follow these generic steps:

1. Add a dependency to the existing app implementation.
2. Add a dependency to Reveal SDK.
3. Initialize Reveal.
4. Enable server-side export

If you are interested in configuring Tomcat or Spring, follow the links below:

- Tomcat Server
- Spring Server
- Oracle Server

Step 1 - Adding a dependency to the app implementation

Add a dependency to the existing application implementation, following the steps needed for the server of your preference.

Step 2 - Adding a dependency to Reveal SDK

Add a dependency to *reveal-sdk* and specify your SDK version.

```
<dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **1.0.1821**.

Step 3 - Initializing Reveal

To initialize Reveal, you use **RevealEngineInitializer.initialize**.

It is possible to invoke the method without initial parameters:

```
RevealEngineInitializer.initialize();
```

But most of the times, you will be using parameters as shown in the example below:

```
RevealEngineInitializer.initialize(
   new InitializeParameterBuilder()
      .setAuthProvider(new RevealAuthenticationProvider())
      .setUserContextProvider(new RevealUserContextProvider())
      .setDashboardProvider(new RevealDashboardProvider())
      .setDataSourceProvider(new UpMediaDataSourceProvider())
      .setDataProvider(new UpMediaInMemoryDataProvider())
      .setMaxConcurrentImageRenderThreads(2)
      .setLicense("SERIAL_KEY_TO_BE_USED")
      .build());
```

Those parameters, are the **providers** used to customize Reveal, you'll need to create your own providers when integrating Reveal into your application.

The available parameters passed to **RevealEngineInitializer.initialize** are:

- *setAuthProvider*. Here you should include a custom class that resolves authentication, implementing IRVAuthenticationProvider.
- *setUserContextProvider*. Custom class that provides information about the user, implementing IRVUserContextProvider.
- *setDashboardProvider*. Custom class that replaces or modifies a dashboard, implementing IRVDashboardProvider.
- *setDataSourceProvider*. Custom class that replaces or modifies a data source, implementing IRVDataSourceProvider.
- *setDataProvider*. Custom class that returns in-memory data for dashboards, implementing IRVDataProvider.
- *setLicense*. Here you can configure the SDK license, by including the Serial Key.

For further details about how implement your own Dashboard providers, please check our UpMedia samples in GitHub.

Step 4 - Enabling server-side export

The Java SDK uses some native components for exporting dashboards to different formats: Image, PDF, PPT and Excel.

If you are interested in exporting server-side to one or more of those formats, please refer to Server-side Export Configuration

Setup and Configuration (Client)

To set up the Reveal Web Client SDK you need to:

1. **Check Dependencies**.
2. **Reference the Web Client SDK**.
3. **Instantiate the Web Client SDK**.

1. Checking Dependencies

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

- jQuery 2.2 or greater
- Day.js 1.8.15 or greater
- Quill RTE 1.3.6 or greater
- Marker Clusterer v3 or greater

- Google Maps v3 or greater

2. Referencing the Web Client SDK

Enabling **$.ig.RevealView** component in a web page requires several scripts to be included. These scripts will be provided as part of Reveal Web Client SDK.

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

JavaScript files can be found in "<InstallationDirectory>\SDK\Web\JS\Client".

3. Instantiating the Web Client SDK

Reveal's Dashboard presentation is handled natively through the Web Client SDK.

To get started follow these steps:

1. Define a element with "id" and invoke the **$.ig.RevealView** constructor.

   Note

   > **Hosting Client-Side and Server-Side Parts Separately** If you want to host client-side and server-side parts on different servers, please read here **before** you continue to next step.

2. Call **$.ig.RVDashboard.loadDashboard** providing the *dashboardId* and success and error handlers.

3. In the success handler instantiate the **$.ig.RevealView** component by passing a selector for the DOM element where the dashboard should be rendered into. Finally you should use the retrieved dashboard and set it to the dashboard property of the **$.ig.RevealView**

Sample Code

```
<!DOCTYPE html>
<html>
  <head>
    ⋮
    <script type="text/javascript">
      var dashboardId = "dashboardId";

      $.ig.RVDashboard.loadDashboard(
        dashboardId,
        function (dashboard) {
          var revealView = new $.ig.RevealView("#revealView");
          revealView.dashboard = dashboard;
        },
        function (error) {
          //Process any error that might occur here
        }
      );
    </script>
  </head>
  <body>
    <div id="revealView" style="height:500px;" />
  </body>
</html>
```

Tomcat Server Setup and Configuration

Prerequisites (Maven)

Reveal Java SDK is distributed as a set of Maven modules. To work with the SDK libraries, you need to add a reference to Reveal's Maven Repository and also a dependency in your Maven pom.xml file.

Add the following repository:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

And the following dependency:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **0.9.6**.

If you are not familiar with Maven, please refer to the following link.

Setup and Configuration

To set up the Reveal with an existing Tomcat application or any other JEE container, you need to:

1. Add a dependency to JAX-RS implementation.
2. Add a dependency to Reveal SDK.
3. Initialize Reveal.
4. Enable server-side export.

Step 1 - Adding a dependency to JAX-RS implementation

Add a dependency to a Jakarta RESTful Web Services (JAX-RS) implementation. You can choose between multiple options like Jersey, RESTeasy, Apache CXF, etc. Please follow the steps described by the provider of your preference.

As an example, here the dependencies you need to add for Jersey:

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>2.32</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jersey.inject</groupId>
    <artifactId>jersey-cdi2-se</artifactId>
    <version>2.32</version>
</dependency>
```

Step 2 - Adding a dependency to Reveal SDK

Add a dependency to *reveal-sdk* and specify your SDK version.

```
<dependency>
   <groupId>com.infragistics.reveal.sdk</groupId>
   <artifactId>reveal-sdk</artifactId>
   <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **1.0.1821**.

Step 3 - Initializing Reveal

Add a **ServletContextListener** class to initialize Reveal. To do this, you can copy the class **WebAppListener** from *upmedia-backend-tomcat* source code, located inside the package *com.pany.analytics.upmedia.reveal*).

To initialize Reveal, you use **RevealEngineInitializer.initialize**.

It is possible to invoke the method without initial parameters:

```
RevealEngineInitializer.initialize();
```

But most of the times, you will be using parameters as shown in the example below:

```
RevealEngineInitializer.initialize(
   new InitializeParameterBuilder()
      .setAuthProvider(new RevealAuthenticationProvider())
      .setUserContextProvider(new RevealUserContextProvider())
      .setDashboardProvider(new RevealDashboardProvider())
      .setDataSourceProvider(new UpMediaDataSourceProvider())
      .setDataProvider(new UpMediaInMemoryDataProvider())
      .setMaxConcurrentImageRenderThreads(2)
      .setLicense("SERIAL_KEY_TO_BE_USED")
      .build());
```

Those parameters, are the **providers** used to customize Reveal, you'll need to create your own providers when integrating Reveal into your application.

The available parameters passed to **RevealEngineInitializer.initialize** are:

- *setAuthProvider*. Here you should include a custom class that resolves authentication, implementing IRVAuthenticationProvider.
- *setUserContextProvider*. Custom class that provides information about the user, implementing IRVUserContextProvider.
- *setDashboardProvider*. Custom class that replaces or modifies a dashboard, implementing IRVDashboardProvider.
- *setDataSourceProvider*. Custom class that replaces or modifies a data source, implementing IRVDataSourceProvider.
- *setDataProvider*. Custom class that returns in-memory data for dashboards, implementing IRVDataProvider.
- *setLicense*. Here you can configure the SDK license, by including the Serial Key.

For further details about how implement your own Dashboards provider, please check our UpMedia samples in GitHub.

Step 4 - Enabling server-side export

The Java SDK uses some native components for exporting dashboards to different formats: Image, PDF, PPT and Excel.

If you are interested in exporting server-side to one or more of those formats, please refer to Server-side Export Configuration

Spring Server Setup and Configuration

Prerequisites (Maven)

Reveal Java SDK is distributed as a set of Maven modules. To work with the SDK libraries, you need to add a reference to Reveal's Maven Repository and also a dependency in your Maven pom.xml file.

Add the following repository:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

And the following dependency:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **0.9.6**.

If you are not familiar with Maven, please refer to the following link.

Setup and Configuration

To set up the Reveal with an existing Spring Boot application, you need to:

1. Add a dependency to spring-starter-jersey implementation.
2. Add a dependency to Reveal SDK.
3. Initialize Reveal.
4. Enable server-side export.

Step 1 - Adding a dependency to spring-starter-jersey implementation

Add a dependency to *spring-starter-jersey*, if not added yet:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
```

Step 2 - Adding a dependency to Reveal SDK

Add a dependency to *reveal-sdk* and specify your SDK version.

```
<dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
</dependency>
```

Replace version_number with a number similar to **1.0.1821**.

Step 3 - Initializing Reveal

Add a **JerseyConfig** component that will initialize a Jakarta RESTful Web Services (JAX-RS) application with Reveal resources. To do this, you can copy the class **RevealJerseyConfig** from *upmedia-backend-spring* source code, located inside the package *com.pany.analytics.upmedia.reveal*).

The *@ApplicationPath* annotation configures the path where Reveal services will be available, if you modify it then you'll also need to modify the client-side path too. For the React application this is configured in index.html:

```
$.ig.RevealSdkSettings.setBaseUrl("http://localhost:8080/upmedia-backend/reveal-api/");
```

To initialize Reveal, you use **RevealEngineInitializer.initialize**.

It is possible to invoke the method without initial parameters:

```
RevealEngineInitializer.initialize();
```

But most of the times, you will be using parameters as shown in the example below:

```
RevealEngineInitializer.initialize(
    new InitializeParameterBuilder()
        .setAuthProvider(new RevealAuthenticationProvider())
        .setUserContextProvider(new RevealUserContextProvider())
        .setDashboardProvider(new RevealDashboardProvider())
        .setDataSourceProvider(new UpMediaDataSourceProvider())
        .setDataProvider(new UpMediaInMemoryDataProvider())
        .setMaxConcurrentImageRenderThreads(2)
        .setLicense("SERIAL_KEY_TO_BE_USED")
        .build());
```

Those parameters, are the **providers** used to customize Reveal, you'll need to create your own providers when integrating Reveal into your application.

The available parameters passed to **RevealEngineInitializer.initialize** are:

- *setAuthProvider*. Here you should include a custom class that resolves authentication, implementing IRVAuthenticationProvider.
- *setUserContextProvider*. Custom class that provides information about the user, implementing IRVUserContextProvider.
- *setDashboardProvider*. Custom class that replaces or modifies a dashboard, implementing IRVDashboardProvider.
- *setDataSourceProvider*. Custom class that replaces or modifies a data source, implementing IRVDataSourceProvider.
- *setDataProvider*. Custom class that returns in-memory data for dashboards, implementing IRVDataProvider.
- *setLicense*. Here you can configure the SDK license, by including the Serial Key.

For further details about how implement your own Dashboards provider, please check our UpMedia samples in GitHub.

Step 4 - Enabling server-side export

The Java SDK uses some native components for exporting dashboards to different formats: Image, PDF, PPT and Excel.

If you are interested in exporting server-side to one or more of those formats, please refer to Server-side Export Configuration

Server-side Export Configuration

The Java SDK uses some native components for exporting dashboards to different formats: Image, PDF, PPT and Excel.

- For **exporting images** we use Playwright for Java.
- For **exporting PDF, PPT and Excel documents** we use ExportTool (our own native application).

Getting ready to export

The first time a Dashboard is opened, **both Playwright and ExportTool trigger the required downloads automatically**. But for some platforms there are some dependencies that need to be installed in advance, and also your server environment might restrict external downloads and you might need to setup these tools manually.

Playwright configuration

Playwright automatically downloads the required binaries. But if manual configuration is required or you want to understand better how it works (or how to tweak it), you can check Playwright documentation here.

macOS Dependencies

The only required library for macOS is **libgdiplus**, you can check installation information here.

Linux Dependencies

There are dependencies to multiple native libraries in Linux. The exact list of dependencies you need to install depends on the distribution used, the version, and list of packages previously installed.

Below there's a list of libraries needed for a basic Ubuntu 18.0.4 distribution:

```
sudo apt-get update

sudo apt-get install -y libgdiplus\
      libatk1.0-0\
      libatk-bridge2.0-0\
      libxkbcommon0\
      libxcomposite1\
      libxdamage1\
      libxfixes3\
      libxrandr2\
      libgbm1\
      libgtk-3-0\
      libpango-1.0-0\
      libcairo2\
      libgdk-pixbuf2.0-0\
      libatspi2.0-0

sudo apt-get install -y --no-install-recommends xvfb
```

If needed, you can get more information about the missing libraries from errors included in the log file.

For other environments, you might also have to install:

```
sudo apt-get install -y --allow-unauthenticated libc6-dev

sudo apt-get install -y --allow-unauthenticated libx11-dev
```

ExportTool Manual Setup

The instructions below are required only in the following scenarios:

- You're having issues with the automatic download mechanism
- You want to have everything pre-installed in advance.

Steps

1. Download the required binaries for your platform: Windows, Linux or macOS.
2. Unzip the file to a directory in your server, where your Web Application is running (your user should be able to access that directory).
3. After extracting the zip file, you can get the **ExportTool** at this location: <*dir*>/<*version*>/<*arch*>/ExportTool, for example:

   ```
   <dir>/1.0.0/linux-x64/ExportTool.
   ```

4. While initializing Reveal, set the directory where you extracted the zip file. Should be similar to the following code snippet:

```
String exportToolDir = "<dir>";
RevealEngineInitializer.initialize(
  new InitializeParameterBuilder()
    .setAuthProvider(new UpmediaAuthenticationProvider())
    .setUserContextProvider(new UpmediaUserContextProvider())
    .setDashboardProvider(new UpmediaDashboardProvider())
    .setLicense("SERIAL_KEY_TO_BE_USED")
    .setExportToolContainerPath(exportToolDir)
    .build());
```

Alternatively, you can specify the directory through the system property **reveal.exportToolContainerPath**, as shown below:

```
java -Dreveal.exportToolContainerPath=<dir> -jar target/upmedia-backend-spring.war
```

Loading Dashboard Files

Overview

There are two ways to open/save dashboards with the SDK:

- **Server-side**: The client-side component of Reveal will use the server-side component to get the definition of a dashboard and also to get the data for each of the visualizations and filters defined.

  Please note that this is the easiest approach and the one recommended when you are first evaluating the SDK.

- **Client-side**: Here you have full control and more flexibility. You provide the stream with the contents of the dashboard on the client page, getting the contents from your own server.

  Using this approach you can, for example, check user permissions, display your own user interface to select the dashboard, or allow users to upload the ".rdash" file to use. For further details about the client-side approach, follow this **Setup and Configuration(Client)**.

The Server-Side Approach

First, you get from the client-side both the dashboard ID and also the ID of the user requesting the dashboard. Second, on the server, you get the definition of the dashboard and get the data for each of the visualization and filters defined.

How dashboards are stored (file system, database, etc.) and who can access them is not part of the SDK and you need to handle that yourself.

You need your own dashboard provider, implementing the interface IRVDashboardProvider:

```
public interface IRVDashboardProvider {
    InputStream getDashboard(String userId, String dashboardId) throws IOException;
    void saveDashboard(String userId, String dashboardId, InputStream dashboardStream) throws IOException;
}
```

If you won't allow dashboards to be saved, you can leave the implementation for *saveDashboard* empty.

The *getDashboard* method receives the user and dashboard IDs, then you need to locate the dashboard (file system, database, etc.) where you choose to store them. Finally, it is expected that you return an InputStream with the dashboard contents, in ".rdash" format (which is basically a ZIP file containing a JSON document).

For further details, you can refer to the UpMedia Samples implementation in GitHub (*UpmediaDashboardProvider* in upmedia, upmedia-backend-tomcat and upmedia-backendspring)

Providing Credentials to Data Sources

Overview

The Server SDK allows you to pass in a set of credentials to be used when accessing the data source.

The first step is to implement **IRVAuthenticationProvider** and then you need to set your custom class when initializing Reveal with the **RevealEngineInitializer.initialize** method. For further details, refer to Initializing Reveal in Java Setup and Configuration. To look to an actual implementation, please refer to the **RevealJerseyConfig** class in the Spring sample or **WebAppListener** in Tomcat-based samples in GitHub.

Code

If you use **UpmediaAuthenticationProvider** (upmedia, upmedia-backend-tomcat and upmedia-backend-spring samples) as a reference, there you can find a single method implemented that receives the *userId* for the current user and the data source for which credentials are being requested:

```
public class UpmediaAuthenticationProvider implements IRVAuthenticationProvider {
    @Override
    public IRVDataSourceCredential resolveCredentials(String userId, RVDashboardDataSource dataSource) {
        // Returning credentials for a SqlServer data source example:
        if (dataSource instanceof RVSqlServerDataSource) {
            String host = ((RVSqlServerDataSource)dataSource).getHost();
            if (host != null && host.equals("10.10.10.10")) {
                return new RVUsernamePasswordDataSourceCredential("someuser", "somesecret", "somedomain");
            }
        }
        return null;
    }
}
```

With a code similar to the example above, you can check if the data source is a MS SQL Server data source and also check for the host name of the server to return the credentials to be used.

Similar code but for a Redshift data source:

```
if (dataSource instanceof RVRedshiftDataSource) {
    String host = ((RVRedshiftDataSource)dataSource).getHost();
    if (host != null && host.equals("1.2.3.4")) {
        return new RVUsernamePasswordDataSourceCredential("user", "password");
    }
}
```

Choosing Which Class to Implement

There are three classes that can be used, all implementing the **IRVDataSourceCredential** interface. You need to choose the class depending on your data source, as detailed below.

CLASS	EXAMPLES
RVBearerTokenDataSourceCredential Associated to OAuth authentication (usually sends the OAuth access token).	Google Analytics, Box, Dropbox, Google Drive, OneDrive, SharePoint Online, OData Feed, Web Resources, REST API.

CLASS	EXAMPLES
RVUsernamePasswordDataSourceCredential Works with user/password style authentication (with an optional domain).	Microsoft Dynamics CRM On-Premises and Online, Microsoft SQL Server, Microsoft Analysis Services Server, MySQL, PostgreSQL, Oracle, Sybase, OData Feed, Web Resources, REST API.
RVAmazonWebServicesCredentials Works with AWS (Amazon Web Services).	Athena, S3.

No Authentication

Sometimes you might work with an anonymous resource, without authentication. In this particular case, you can use **RVUsernamePasswordDataSourceCredential**, which has an empty constructor. You can do this for any data source that works with the class.

Java SDK Redistributables (OSS, Third-Party Software and Reveal Binaries)

SERVER/CLIENT	ASSEMBLY/CODE USED	LICENSE
Server-side SDK	reveal-sdk-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	reveal-rest-service-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	reveal-engine-service-api-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	reveal-sdk-api-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	reveal-engine-service-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	connectors-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	datalayer-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	core-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	requests-<*sdk_version_number*>.jar	Reveal license
Server-side SDK	ExportTool(.exe)	Reveal license
Client-side SDK	infragistics.reveal.js	Reveal license
Client-side SDK	infragistics.langpack.<*country_name*>.js	Reveal license
Client-side SDK	reveal-webComponent.js	Reveal license
Server-side SDK	driver-1.10.0.jar, driver-bundle-1.10.0.jar	Apache 2.0
Server-side SDK	Java-WebSocket-1.5.1.jar	MIT
Server-side SDK	kxml2-2.3.0.jar	MIT
Server-side SDK	ojdbc14-10.2.0.5.0.jar	OTN
Server-side SDK	postgresql-9.4.1212.jre6	PostgreSQL License
Server-side SDK	okhttp-3.8.0.jar	Apache 2.0
Server-side SDK	jtds-1.3.1.jar	LGPL-2.1
Server-side SDK	gson-2.8.6.jar	Apache 2.0
Server-side SDK	jxl-2.6.12.jar	LGPL-2.0
Server-side SDK	antlr4-runtime-4.7.jar	BSD 3-Clause
Server-side SDK	joda-time-2.10	Apache 2.0

Server-side SDK	jackson-core-2.9.6.jar	Apache 2.0
Server-side SDK	commons-text-1.1.jar	Apache 2.0
Server-side SDK	Utility class from Spring	Apache 2.0
Server-side SDK	jcifs-1.3.17.jar	LGPL-2.1
Server-side SDK	json-20171018.jar	Custom
Server-side SDK	log4j-1.2.14	Apache 2.0
Server-side SDK	mariadb-java-client-1.5.9.jar	LGPL-2.1
Server-side SDK	okio-1.13.0.jar	Apache 2.0
Server-side SDK	commons-lang3-3.5	Apache 2.0
Server-side SDK	snowflake JDBC driver	Apache 2.0
Client-side SDK	Bootstrap	MIT
Client-side SDK	jQuery	MIT
Client-side SDK	jQuery validation	MIT
Client-side SDK	jQuery validation unobtrusive	Apache 2.0
Client-side SDK	Roboto Google Font	Apache 2.0
Client-side SDK	Google Maps Marker Clusterer	Apache 2.0
Client-side SDK	Google Maps Platform	Custom
Client-side SDK	Babel JS Compiler	MIT
Client-side SDK	LitElement	BSD 3-Clause
Client-side SDK	Simple JavaScript Inheritance	MIT
Client-side SDK	Day.js	MIT
Client-side SDK	Quill Rich Text Editor	BSD 3-Clause
Client-side SDK	snowflake connector .NET	Apache 2.0

JAVA API Reference

Here you will find technical information about Reveal SDK, specifically about the JAVA Server API. For the complete reference, please follow the link.

Most commonly used classes and interfaces

Main SDK concepts and features:
RevealEngineInitializer
InitializeParameter
InitializeParameterBuilder

Datasources
IRVDashboardProvider
IRVDataSourceProvider

RVSqlServerDataSource
RVSqlServerDataSourceItem

RVExcelDataSource
RVExcelDataSource

RVRESTDataSource
RVRESTDataSourceItem

RVJsonDataSource
RVJsonDataSourceItem

RVInMemoryDataSource
RVInMemoryDataSourceItem

Authentication
IRVAuthenticationProvider
IRVDataSourceCredential
RVUsernamePasswordDataSourceCredential
IRVUserContextProvider

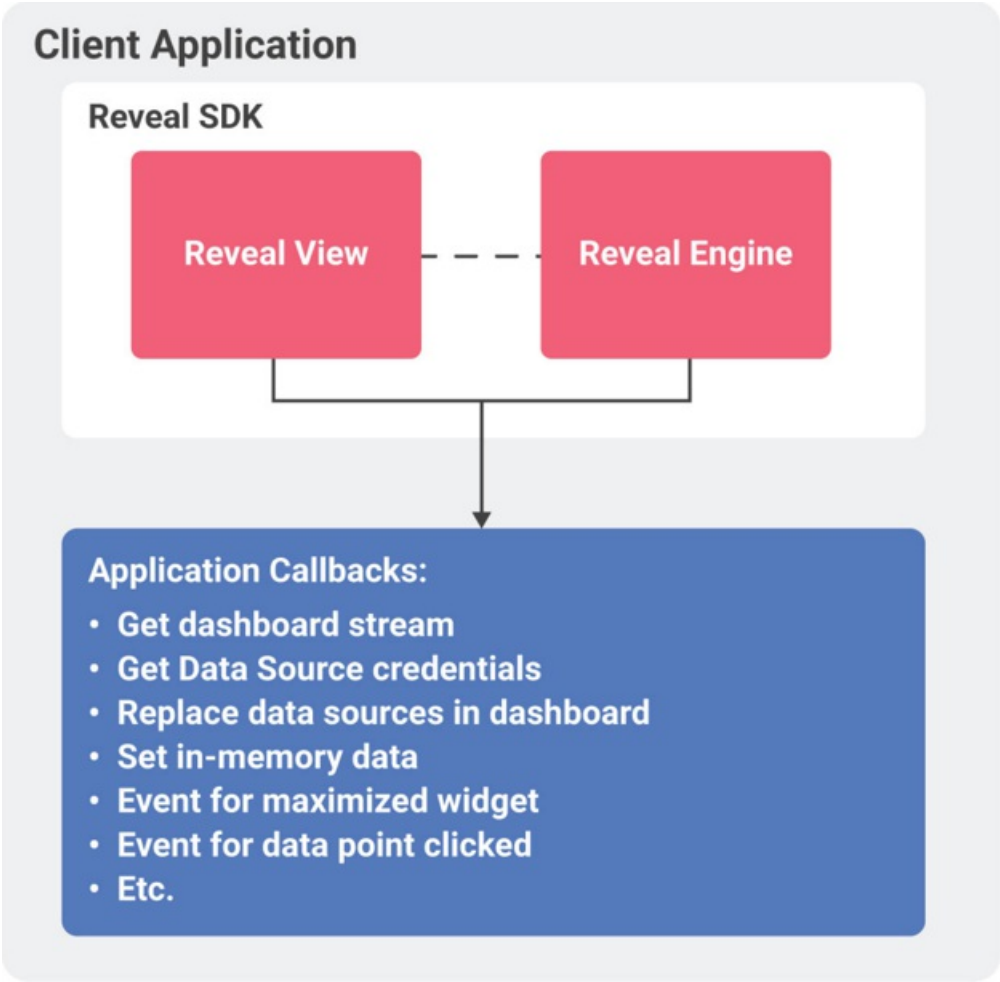
Overview

When you embed Reveal in another application (Windows WPF, Windows Forms, iOS or Android), the Reveal SDK is provided as a library or framework that is integrated into the app (the integration steps are different for each platform).

The RevealView Component

The containing app creates a RevealView object configured with the dashboard to render, this view is displayed in the containing app and then a set of callbacks can be used to customize how the dashboard is rendered and what data is used.

The RevealView component provides rendering and data transformation capabilities (automatically through the Reveal Engine), but it does not handle the storage of dashboards or credentials. The binary contents (.rdash file) holding the definition of the dashboard must be provided by the containing app. This allows the container app to handle how dashboards are used and shared by end users (for example they can be downloaded from an internal server, bundled as resources in the app's binary, stored in file system, etc.).



About Credentials

When getting data from databases (or other data sources requiring authentication), usually the containing app already handles these credentials by loading them from configuration files, or storing them in a secure storage. This means that Reveal delegates the storage and handling of these credentials to the container. The app might decide to return internal credentials or prompt the user for them, if needed.

Setup and Configuration

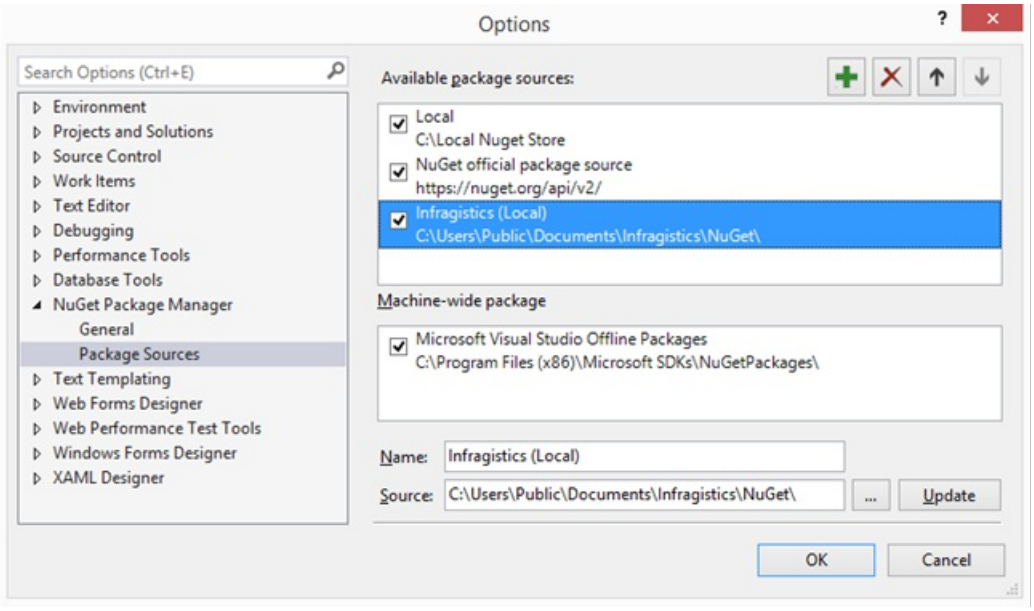
To set up the Reveal Desktop SDK you can choose between:

- Using **NuGet** package manager.
- Setting up the project **manually**.

Using NuGet (Recommended)

The easiest way to setup your WPF or WinForms application project is to install the **Reveal.Sdk.Wpf** NuGet package.

After installing the Reveal SDK, you should be able to find a new NuGet package source added to your **nuget.config** called *Infragistics (Local)* that points to "%public%\Documents\Infragistics\NuGet".



After ensuring you have the Infragistics (Local) feed properly configured by the installer, you can install the **Reveal.Sdk.Wpf** NuGet package to your application project.

By installing the NuGet package, you will also install the following dependency packages:

- CefSharp.Wpf (87.1.132+)
- SkiaSharp (1.68.0+)
- System.Data.SQLite.Core (1.0.108+)

Note

You need to manually install Microsoft.Data.SqlClient (1.1.3) package in your project to be able to visualize Microsoft Sql Server data.

Using Manual Setup

To setup your project manually you need to:

1. Add references to the assemblies dropped by the installer at "<InstallationDirectory>\SDK\WPF\Binaries".
2. Install the following NuGet packages, which *RevealView* control depends on:
    - CefSharp.Wpf (87.1.132+)
    - SkiaSharp (1.68.0+)
    - System.Data.SQLite.Core (1.0.108+)
    - Microsoft.Data.SqlClient (1.1.3+)

To handle the **CefSharp.Wpf** known issue, please continue reading below.

Handling CefSharp Dependency Package

Your build (if targeting *AnyCPU*) **will be failing** after the installation of the CefSharp dependency package.

Note

About the Error: The error description will be: *"error : CefSharp.Common will work out of the box if you specify platform (x86 / x64).* For AnyCPU Support please follow this **link**.

To fix this error, you need to add the *CefSharpAnyCpuSupport* property to your project file as explained in the error's URL. Just add the following property group to your application's **.csproj** file:

```
<PropertyGroup>

  <CefSharpAnyCpuSupport>true</CefSharpAnyCpuSupport>

</PropertyGroup>
```

This is all you need to do to fix the error. You don't need to apply the other instructions pointed out in the CefSharp's GitHub issue. Reveal component will make sure to initialize the *CefBrowser* class when needed.

At this point your project should be set up to display Reveal dashboards.

Creating New Visualizations and Dashboards

Overview

In order to add new visualizations to an existing dashboard, the user needs to **select the data source** to be used. To do that, the containing application needs to provide information to the SDK, so it can display the list of data sources available for a new visualization.

Displaying a List of Data Sources

The callback you need to use to display a list of data sources is **DataSourcesRequested**. In the case that you don't set your own method to this callback, when a new visualization is created, Reveal will display all data sources used in the dashboard (if any).

Code

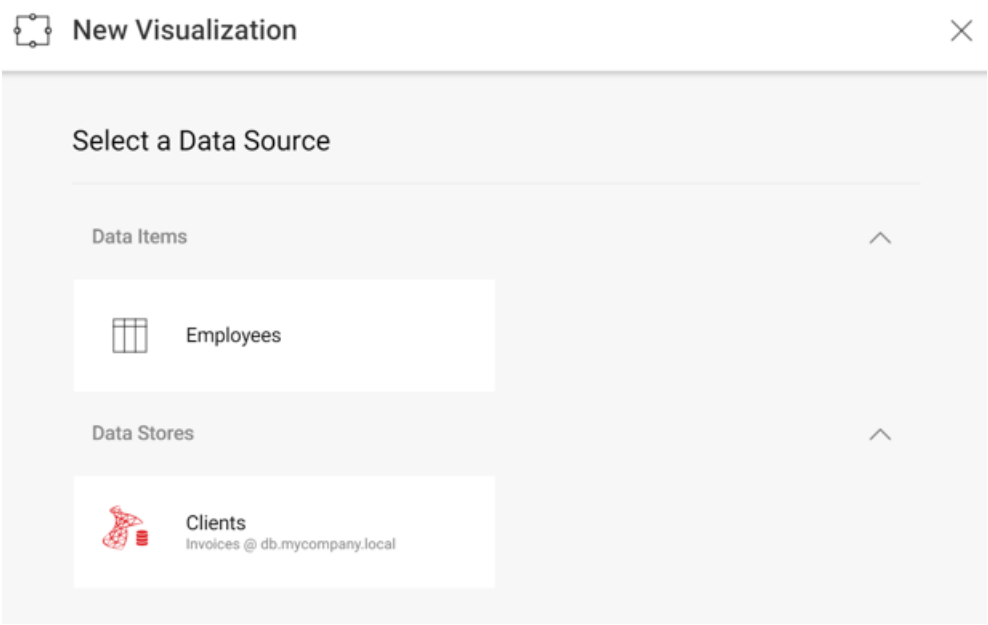
The code below shows how to configure the *data source selection* screen to show an "in-memory" item and a SQL Server data source.

```
private void RevealView_DataSourcesRequested(object sender, DataSourcesRequestedEventArgs e)
{
   var inMemoryDSI = new RVInMemoryDataSourceItem("employees");
   inMemoryDSI.Title = "Employees";
   inMemoryDSI.Description = "Employees";

   var sqlDs = new RVSqlServerDataSource();
   sqlDs.Title = "Clients";
   sqlDs.Id = "SqlDataSource1";
   sqlDs.Host = "db.mycompany.local";
   sqlDs.Port = 1433;
   sqlDs.Database = "Invoices";

   e.Callback(new RevealDataSources(
         new List<RVDashboardDataSource> { sqlDs },
         new List<RVDataSourceItem> { inMemoryDSI },
         false));
}
```

The previous code assumes that you attached the following method to handle the "DataSourcesRequested" event:

```
revealView.DataSourcesRequested += RevealView_DataSourcesRequested
```

The "false" value in the third parameter prevents existing data sources on the dashboard from being displayed. So, when creating a new widget using the "+" button, you should get the following screen:



Please note that the “employees” parameter passed to the “RVInMemoryDataSourceItem” constructor, is the same dataset id used in **In-Memory Data Support** and identifies the dataset to be returned.

Creating New Dashboards

Creating dashboards from scratch is really simple. You just need to initialize **RevealView**. Usually when providing the users the capability to create a dashboard from scratch you would want to open the empty dashboard directly in edit mode so the user could start editing it straight away.

```
revealView = new RevealView();
revealView.StartInEditMode = true;
revealView.DataSourcesRequested += RevealView_DataSourcesRequested;

revealView.Dashboard = new RVDashboard();
```

You can find a working example in the **EmptyDashboard.xaml.cs** view in the *UpMedia* WPF application distributed with the SDK.

Loading Dashboards

If you want to display an existing Reveal Dashboard in the `RevealView` control embedded within your application, you have four options to choose from.

- Load the dashboard from a file path
- Load the dashboard from a file stream
- Load the dashboard from an embedded resource
- Load the dashboard from json

Loading a dashboard into a `RevealView` consists of taking a **.rdash** file (.rdash is the file extension for dashboards created by Reveal), deserializing it as a `RVDashboard` object, and then assigning the `RevealView.Dashboard` property to the RVDashboard object instance.

You can create **.rdash** dashboard files the following ways:

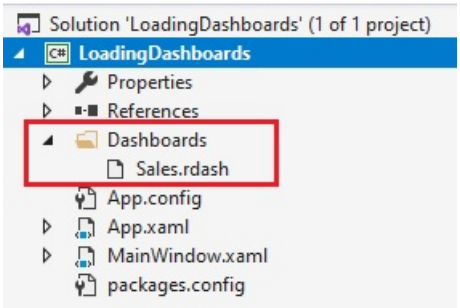
- Export the dashboard as a .rdash file from the Reveal BI website
- Export the dashboard as a .rdash file from one of the native Reveal applicatons
- Save, or Export, a dashboard that was created in an application using the Reveal SDK.

For further details, please refer to Getting Dashboards for the SDK.

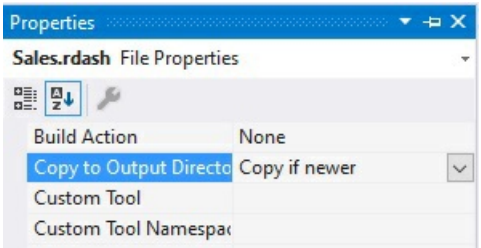
Load from File Path

It is very common to ship dashboard files with your application. These files are usually copied to the clients disk drive in a known directory so that the files can be loaded from disk during the execution of the application. In order to load these dashboards using a file path, you must know the file path to the **.rdash** file.

In this example, we have created a directory in our Visual Studio solution called **Dashboards** which will contain all the .rdash files for our application.



It's important to make sure we set the **Copy to Output Directory** value to **Copy if Newer** in the properties of each .rdash file. This will copy the dashboard files to disk when the project is built.



The first step is to get the file location of the .rdash file you wish to load. Once you have the file path to your dashboard, create a new instance of the `RVDashboard` and pass the file path to the constructor of the `RVDashboard` class.

In our example, we are using `Environment.CurrentDirectory` to get the curent executing directory of our application. We then append the location of the **Sales.rdash** dashboard, which is in our **Dashboards** directory, using the `Path.Combine` method. Once we have the correct file path to our **Sales.rdash** dashboard, we set the `RevealView.Dashboard` property to a new instance of an `RVDashboard` object using the file

path as a constructor argument.

```
var filePath = Path.Combine(Environment.CurrentDirectory, "Dashboards/Sales.rdash");
_revealView.Dashboard = new RVDashboard(filePath);
```

You can also load dashboards into the `RevealView` from a file path asynchronously using the `RVDashboard.LoadDashboardAsync` method.

```
var filePath = Path.Combine(Environment.CurrentDirectory, "Dashboards/Sales.rdash");
_revealView.Dashboard = await RVDashboard.LoadDashboardAsync(filePath);
```

Note

The source code to this sample can be found on GitHub.

Load from File Stream

Loading Reveal dashboards from a file stream is very similar to loading dashboards from a file path. In this case, once you have the file path of the dashboard file, you load it into a `FileStream` before creating the `RVDashboard` object instance.

In this example, we are using the `File.OpenRead` method to load the Sales.rdash file into a file stream. We then create a new `RVDashboard` object by passing the file stream as a constructor argument and assign the newly created `RVDashboard` instance to the `RevealView.Dashboard` property.

```
var filePath = Path.Combine(Environment.CurrentDirectory, "Dashboards/Sales.rdash");
using (var stream = File.OpenRead(filePath))
{
    _revealView.Dashboard = new RVDashboard(stream);
}
```

You can also load dashboards into the `RevealView` from a file stream asynchronously using the `RVDashboard.LoadDashboardAsync` method.

```
var filePath = Path.Combine(Environment.CurrentDirectory, "Dashboards/Sales.rdash");
using (var stream = File.OpenRead(filePath))
{
    _revealView.Dashboard = await RVDashboard.LoadDashboardAsync(stream);
}
```

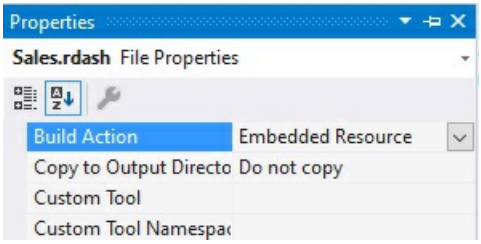
Note

The source code to this sample can be found on GitHub.

Load from Resource

Another option for distributing files in an application is to embed them into your application as a resource. This will not place any files on the client's disk drive, but rather embed the files directly into your application's assembly.

To embed a Reveal dashboard **.rdash** file as a resource in your application, open the Properties for the dashboard file in Visual Studio, and set the **Build Action** of the .rdash file to **EmbeddedResource**.



Once your dashboards have been defined as an **EmbeddedResource**, you can load the dashboard by using the `Assembly.GetManifestResourceStream` method. This metho will return a `Stream` object that you can then use to load into the `RevealView`.

It's important to note, that the `name` of the resource you will provide in the `Assembly.GetManifestResourceStream` method must include the `namespace` and file name of the .rdash file.

In this example, the name of the resource starts with the application root namespace "LoadingDashboards", plus "Dashboards" which is the directory that contains the dashboard files, followed by the name of the .rdash file "Sales.rdash". This gives us the full resource name of `LoadingDashboards.Dashboards.Sales.rdash`

```
var resource = Assembly.GetExecutingAssembly().GetManifestResourceStream($"LoadingDashboards.Dashboards.Sales.rdash");
using (resource)
{
    _revealView.Dashboard = new RVDashboard(resource);
}
```

You can also load dashboards as embedded reouscres into the `RevealView` from a resource stream asynchronously using the `RVDashboard.LoadDashboardAsync` method.

```
var resource = Assembly.GetExecutingAssembly().GetManifestResourceStream($"LoadingDashboards.Dashboards.Sales.rdash");
using (resource)
{
    _revealView.Dashboard = await RVDashboard.LoadDashboardAsync(resource);
}
```

Note

The source code to this sample can be found on GitHub.

Load From JSON

For advanced users, or users that wish to serialize Reveal dashboards into .json files instead of .rdash files, you can load these JSON based files using the `RVDashboard.LoadFromJsonAsync` method.

The first step is to serialize a Reveal dashboard into a json string. Once you have the string you can then save the JSON to disk or anothe rdata store.

To serialize a Reveal Dashboard into JSON simply call the `RVDashboard.ExportToJson` method.

```
var json = dashboard.ExportToJson();
```

Once the dashboard has been serialized into JSON format, you can now save that JSON file to disk or load it directly into the `RevealView`.

If loading a dashboard JSON file from disk, your code may look something like this:

```
var filePath = Path.Combine(Environment.CurrentDirectory, "Dashboards/Sales.json");
var json = File.ReadAllText(filePath);
```

Once you have the JSON string, you can load the dashboard by setting the `RevealView.Dashboard` property with the result of the `RVDashboard.LoadFromJsonAsync` method passing the JSON string as a method argument.

```
_revealView.Dashboard = await RVDashboard.LoadFromJsonAsync(json);
```

Warning

Manipulating or changing the contents of a Reveal dashboard after it has been serialized to JSON can break the integrety of the dashboard and cause irreversible damage to the contents of the dashboard. This could result in runtime exceptions being thrown in your application due to errors and/or a failure to load the dashboard.

Note

The source code to this sample can be found on GitHub.

Editing & Saving Dashboards

Overview

To load a dashboard, you need to provide the **RevealView** component with the stream containing the dashboard file. Later, you might want to handle the modified dashboard file, once the user made changes to the dashboard.

Editing dashboards

The **Dashboard** property (type RVDashboard) of **RevealView** is updated when the end user starts editing the dashboard. For example, when adding or removing visualizations or filters, RVDashboard's collections get automatically updated.

Attach to the **PropertyChanged** event in **RVDashboard**, to get notified of changes to properties like **HasPendingChanges**:

Code Sample:

```
dashboard.PropertyChanged += Dashboard_PropertyChanged;
```

Then, implement the event handler:

```
private void Dashboard_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)
{
    if (e.PropertyName == "HasPendingChanges")
    {
        Console.Out.WriteLine("HasPendingChanges: " + ((RVDashboard)sender).HasPendingChanges);
    }
}
```

After a user finishes editing a visualization, upon closing the Visualization Editor, the Revealview's **VisualizationEditorClosed** event is fired. You can attach to the event with this code:

```
revealView.VisualizationEditorClosed += RevealView_VisualizationEditorClosed;
```

And then you need to implement the event handler:

```
private void RevealView_VisualizationEditorClosed(object sender, VisualizationEditorClosedEventArgs e)
{
    if (e.IsCancelled)
    {
        Console.Out.WriteLine("Visualization editor cancelled " + (e.IsNewVisualization ? "creating a new visualization" : "editing " + e.Visualization.Title));
        return;
    }
    if (e.IsNewVisualization)
    {
        Console.Out.WriteLine("New visualization created: " + e.Visualization.Title);
    } else
    {
        Console.Out.WriteLine("Visualization modified: " + e.Visualization.Title);
    }
}
```

In the case that you need to control how to add new visualizations please refer to **Creating New Visualizations and Dashboards**.

Saving dashboards

You can attach to the **SaveDashboard** event with the following code:

```
_revealView.SaveDashboard += RevealView_SaveDashboard;
```

And then implement the event handler:

```
private async void RevealView_SaveDashboard(object sender, DashboardSaveEventArgs args)
{
    var data = await args.Serialize();
    using (var output = File.Open($"{args.Name}.rdash", FileMode.Create))
    {
        output.Write(data, 0, data.Length);
    }
    args.SaveFinished();
}
```

The example above shows a simplified implementation for handling the “Save” event. In a real-world scenario you might want to display a custom UI to the user, where he/she can select the location and final name of the dashboard.

In the case that the user changed the name of the dashboard, you can use the method **SerializeWithNewName**, to get the name reflected properly in the Title attribute for the dashboard.

In case you don’t want to handle the save action, you can turn off the option to edit dashboards by setting:

```
revealView.CanEdit = false;
```

This might be useful, for example, when your users are not supposed to make changes.

Exporting a Dashboard or a Visualization

Overview

If you want to export a dashboard or a particular visualization, you can choose between the following export options:

- as an **image**;
- as a **PDF** document;
- as a **PowerPoint** presentation;
- into **Excel** data format.

To enable a dashboard or a visualization export, you can:

- use the export setting in the RevealView, or
- initiate export programmatically outside of the RevealView, when exporting **as an image**.

Prerequisites for Export as an Image Option

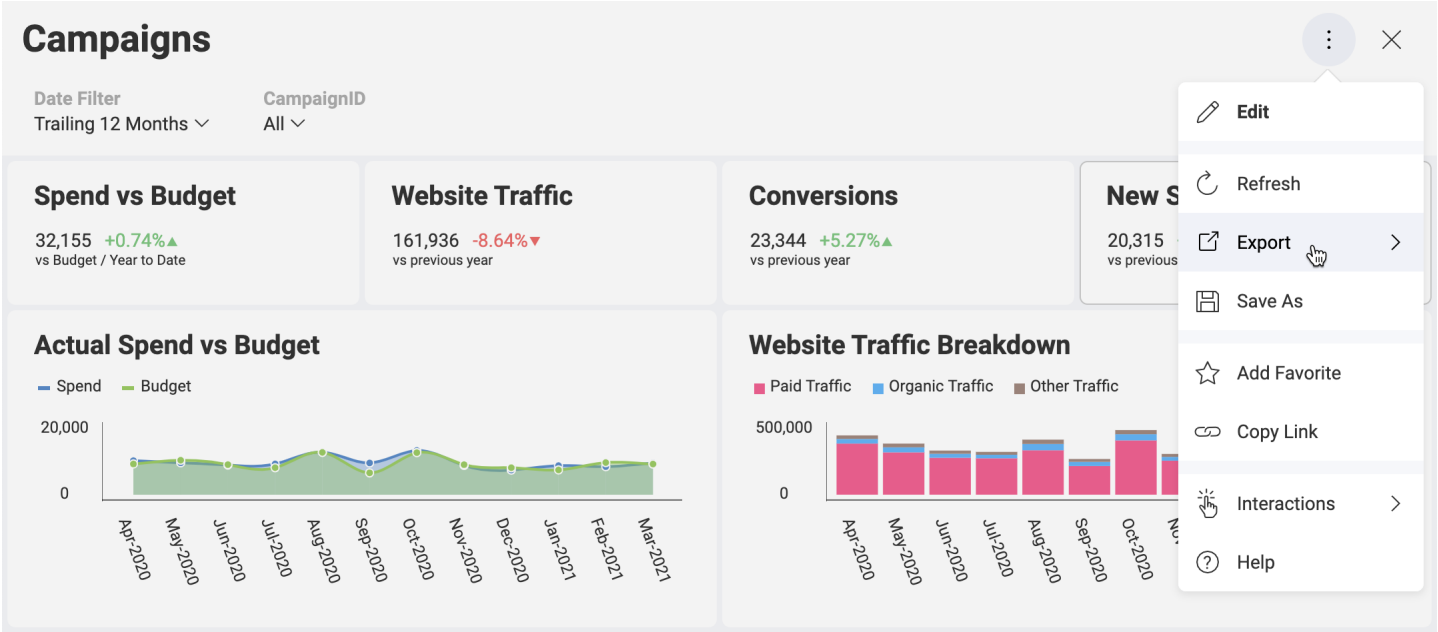
To use the Export as an image feature you will need to add a reference to CefSharp.Wpf NuGet package (>= 83.4.20) to your project.

Using the Export Setting

To enable your end users to generate an image, document or a presentation out of a dashboard you simply need to set the relevant property to true when loading the dashboard:

- **RevealView.ShowExportImage** - for export as an **image**;
- **RevealView.ShowExportToPDF](api-reference** - for export as a **PDF** document;
- **RevealView.ShowExportToPowerpoint** - for export as a **PowerPoint** presentation;
- **RevealView.ShowExportToExcel** - for export in **Excel** data format.

This will make the *Export* button available in the overflow menu when a dashboard is opened or a particular visualization is maximized.



When the user clicks the *Export* button, they can choose one of the enabled export options.

Specifics when using the image export option

If the user chooses the *Export Image* from the export options, the *Export image* dialog will open. Here, the user can choose between two options: *Copy to clipboard* and *Export Image*.

If they click the *Export Image* button on the bottom right, the RevealView raises the **ImageExported** event. To access the image via the **Image Property** of the **ImageExportedEventArgs**, you need to have already subscribed to this event through the event handler. If you haven't subscribed to the event the save file dialog will be opened for the user to specify a location to save the image to.

Here's a sample implementation of the *ImageExported* event handler:

```
private void RevealView_ImageExported(object sender, ImageExportedEventArgs e)
{
  var image = e.Image;
  if (image == null) return;
  // save to disk just to open it with some app
  var imageFile = Path.GetTempFileName() + ".png";
  using (var fileStream = new FileStream(imageFile, FileMode.Create))

  {
    BitmapEncoder encoder = new PngBitmapEncoder();
    encoder.Frames.Add(BitmapFrame.Create(image));
    encoder.Save(fileStream);
  }

  System.Diagnostics.Process.Start(imageFile);
}
```

The other property of the *ImageExported* event arguments is the **CloseExportDialog**. Its default value is set to true. If you set it to false, the *Export Image* dialog won't be closed after finishing of the event handler invocation.

It might be useful to set the *CloseExportDialog* to false in a scenario, in which you show a save dialog to the end user so they can choose where to save the image. If the user does not pick a location and filename and closes the save dialog, you might want to keep the ExportImage dialog opened.

Programmatically Initiated Image Export

To get an image of the RevealView programmatically, you will need to invoke the **ToImage** method. Calling this method will not result in showing the *Export Image* dialog. This way, you can get a screenshot when the user clicks a button, which is outside of the RevealView. This method will create a screenshot of the RevealView component as it is displayed on the screen.

Keep in mind that if the end user has any dialog opened at the time of the *ToImage* method call, the dialog will appear in be screenshot together with the dashboard.

Dashboard Linking

Overview

The Reveal application supports dashboard linking, which allows users to navigate through dashboards. By moving from dashboard to dashboard, you can go from a high level overview of the business' reality to a more detailed view with the specifics.

Common Use Cases

You can have, for example, a "Company 360" dashboard showing key performance indicators for each division (HR, Sales, Marketing). Once the user maximizes one of the visualizations, the navigation is triggered and the user is taken to another dashboard with more detailed information about that division.

Alternatively, you can also use dashboard linking to navigate to a more specific dashboard. For example, in a dashboard with a visualization showing the Top 25 Customers, by selecting one of the customers the user will navigate to a new dashboard. This new dashboard will display detailed information about the selected customer, like recent purchases, contact information, top selling products, etc.

For further details about the Dashboard Linking functionality, refer to **Dashboard Linking** from the Reveal's User Guide.

Code Example

You can use dashboard linking with the SDK, but the containing application needs to be involved when the navigation is being triggered. As the SDK does not handle where dashboards are stored, it needs the containing application to provide a dashboard file for the target dashboard.

As a practical example, you can use the **Marketing.xaml.cs** file in the *UpMedia* sample distributed with the SDK.

Basically, you just have to do two things and the SDK will take care of the rest:

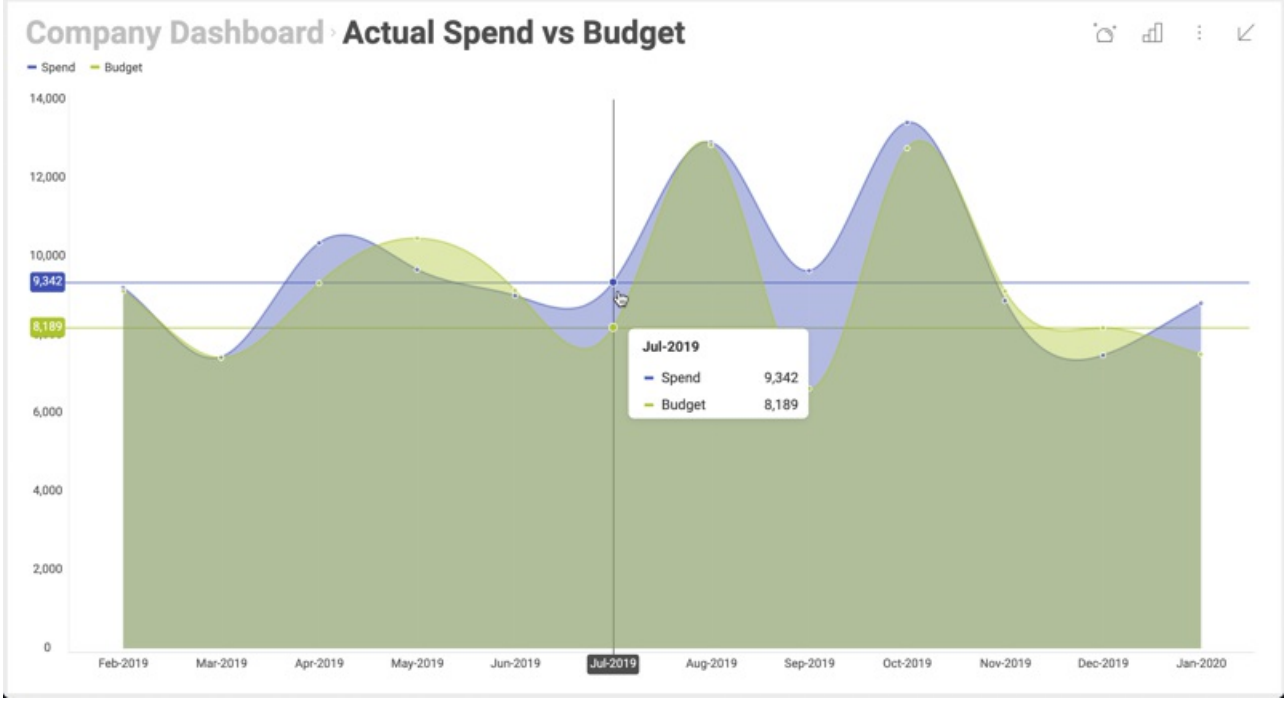
1. Handle the **VisualizationLinkingDashboard** event.
2. Invoke the callback with the ID of the target dashboard.

```
//attach to VisualizationLinkingDashboard event
revealView.VisualizationLinkingDashboard += RevealView_VisualizationLinking;

//Implement the event handler
private void RevealView_VisualizationLinking(object sender,
    VisualizationLinkingDashboardEventArgs e)
    {
        e.Callback("Campaigns", GetDashboardStream("Campaigns"));
    }
```

Maximizing Visualizations and Single Visualization Mode

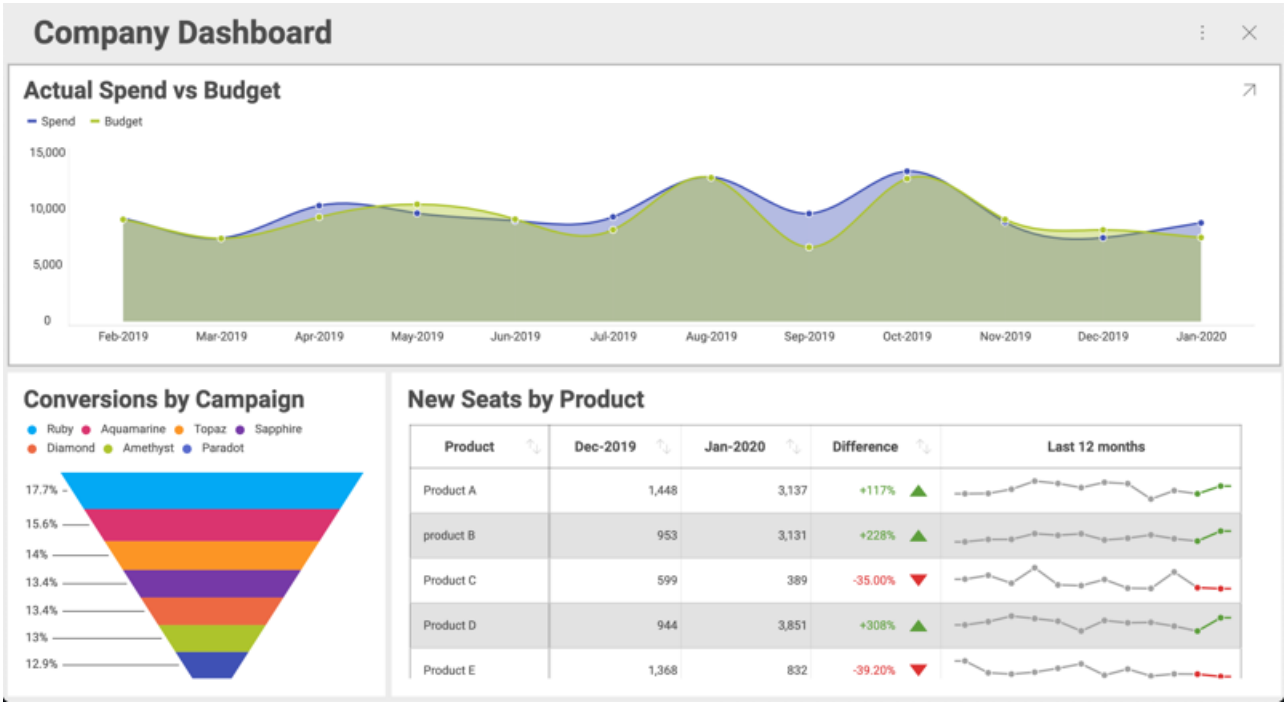
Overview

When displaying a dashboard to the user, there are some cases in which you'd like to display just one maximized visualization. In addition, you might also want to lock the initial visualization and prevent the user from accessing the whole dashboard. You can achieve both scenarios using the Desktop SDK.



Example Details

Let's assume that you have a dashboard with three visualizations, where each visualization is showing data for a different division of your company, i.e., "Marketing", "Sales" and "HR".



In this example, you'd like to showcase these visualizations in your corporate application. You want to include them as part of the information displayed on each division's home page.

Maximizing Visualizations

To open a dashboard with a maximized visualization, you need to use the **MaximizedVisualization** property of **RevealView** after you have assigned the revealView.Dashboard property. When you don't set a visualization in this attribute, the whole dashboard is displayed.

As shown in **Configuring the RevealViewobject**, you can display a specific dashboard in your page. This time, you also need to set the **MaximizedVisualization** attribute. As shown in the code snippet below with the visualization "Sales".

```
var revealView = new RevealView();
using (var fileStream = File.OpenRead(path))
{
   var dashboard = new RVDashboard(fileStream);
   revealView.Dashboard = dashboard;
   revealView.MaximizedVisualization = dashboard.Visualizations.GetByTitle("Sales");
}
```

Although the initial maximized visualization will be the one with title 'Sales', the end user can still return to the dashboard and see the rest of the visualizations.

Single Visualization Mode

You may also want to lock the initial visualization, making it the only one displayed at all times. This way the dashboard works like a single visualization dashboard. This is the concept behind "single visualization mode".

To turn on the "single visualization mode", just set the **SingleVisualizationMode** property to true, as shown below.

```
revealView.SingleVisualizationMode = true;
```

After adding this single line, the dashboard will work as a single visualization dashboard. You can do the same for each division's home page, just replace the title of the visualization in **dashboard.Visualizations.GetByTitle()** with the right one.

Dynamically changing a locked visualization

It is also possible for you to dynamically change the single visualization being displayed, without reloading the page. From the user's perspective, your app would be a single page application with a selector of divisions and a maximized visualization. After the user chooses one division from the list, the maximized visualization is updated.

You can achieve this scenario by using the **MaximizeVisualization** method in **RevealView** or set the **MaximizedVisualization** property, as shown below:

```
private void MaximizeVisualization(string title)
    {
       revealView.MaximizeVisualization(revealView.Dashboard.Visualizations.GetTitle(title));
       //or set the property
       revealView.MaximizedVisualization = revealView.Dashboard.Visualizations.GetTitle(title);
    }
```

Finally, you have to connect your custom control with the method above. That way the visualization will be maximized when the selection in your application changes.

To take into account:

- You can generate the list of buttons dynamically by iterating the list of visualizations in the dashboard. For further details see **RVDashboard.Visualizations**.
- There is a working example in the **Manufacturing.xaml.cs** view, in the *UpMedia* WPF application distributed with the SDK. That sample view shows all visualizations as a list of toggle buttons, at the bottom of the screen.

In-Memory Data Support

Overview

In some cases you need to use data already in memory as part of your application state. Using, for example, the result of a report requested by the user, or information from a data source not yet supported by Reveal (like a custom database or a specific file format).

In-Memory is a special type of data source that can be used only with the SDK and not in the Reveal application out of the box. Because of this, you cannot use an "in-memory data source" directly, you need to to take a different approach as explained below.

Using In-Memory Data Source

The recommended approach is to **define a data file with a schema, matching your in-memory data**. Data files can be, for example, CSV or Excel files, and a schema is basically a list of fields and the data type for each field.

In the example below you find details about how to create a data file with a given schema, and then use data in memory instead of getting information from a database.

Code Example

In the following example, you want to use in-memory data with the list of Employees in the company, in order to embed a dashboard showing HR metrics in your HR system. And instead of getting the list of employees from your database, you want to use data in memory.

To achieve all that, you will need to create and export a dashboard in the Reveal application using dummy data.

About the Reveal Application

The Reveal Application is a self-service business intelligence tool that enables you to create, view and share dashboards with your teams. For further details about the Reveal app, you can access an **online demo** or browse the **Help Documentation**.

Getting the Data File and Sample Dashboard Ready

As simplified Employee has only the following properties:

- *EmployeeID*: string
- *Fullname*: string
- *Wage*: numeric

Steps

1. Create The CSV file with the same schema:

   ```
   EmployeeID,Fullname,Wage
   23,John Smith,345.67
   45,Emma Thompson,432.23
   ```

2. Upload the file to your preferred File Sharing System, like Dropbox or Google Drive.

3. Create a dashboard within the Reveal app using the dummy data. Please note that you are going to provide the real production data later in your application.

4. Export the dashboard from the Reveal app (Dashboard Menu → Export → Dashboard) and save a .rdash file.

Visualizing the Dashboard and Returning the Actual Data

Now you need to visualize the dashboard using your own data instead of the dummy one.

1. Implement **IRVDataSourceProvider** and set it to the **DataSourceProvider** property in **RevealSdkSettings**, as described in

Replacing Data Sources.

Then, in the implementation for the method **ChangeVisualizationDataSourceItemAsync**, you need to add a code similar to this one:

```
public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(RVVisualization visualization, RVDataSourceItem dataSourceItem)
{
   var csvDsi = dataSourceItem as RVCsvDataSourceItem;
   if (csvDsi != null)
   {
      var inMemDsi = new RVInMemoryDataSourceItem("employees");
      return Task.FromResult((RVDataSourceItem)inMemDsi);
   }
   return Task.FromResult((RVDataSourceItem)null);
}
```

This way you basically replace all references to CSV files in the dashboard with the in-memory data source identified by “employees”. This identification will be used later when returning the data.

2. Implement the method that will return the actual data, to do that implement **IRVDataProvider** as shown below:

```
public class EmbedDataProvider : IRVDataProvider
{
   public Task<IRVInMemoryData> GetData(RVInMemoryDataSourceItem dataSourceItem)
   {
      var datasetId = dataSourceItem.DatasetId;
      if (datasetId == "employees")
      {
         var data = new List<Employee>()
            {
               new Employee(){ EmployeeID = "1", Fullname="John Doe", Wage = 80325.61 },
               new Employee(){ EmployeeID = "2", Fullname="Doe John", Wage = 10325.61 },
            };
         return Task.FromResult<IRVInMemoryData>(new RVInMemoryData<Employee>(data));
      }
      else
      {
         throw new Exception("Invalid data requested");
      }
   }
```

Please note that the properties in the Employee class are named exactly as the columns in the CSV file, and the data type is also the same. In case you want to alter the field name, field label and/or data type of any of the properties you can use the following attributes in the class declaration:

- RVSchemaColumn attribute can be used to alter the field name and/or data type
- DisplayName attribute can be used to alter the field label

```
public class Employee
{
 [RVSchemaColumn("EmployeeID", RVSchemaColumnType.Number)]
 public string EmployeeID { get; set; }

 [DisplayName("EmployeeFullName")]
 public string Fullname { get; set; }

 [RVSchemaColumn("MonthlyWage")]
 public double Wage { get; set; }
}
```

Finally, remember to set your implementation of **IRVDataProvider** to **RevealView.DataProvider** property:

```
revealView.DataProvider = new SampleDataProvider();
```

Providing Credentials to Data Sources

Overview

The Server SDK allows you to pass in a set of credentials to be used when accessing the data source.

Code

The first step is to implement **IRVAuthenticationProvider** and set it to the **AuthenticationProvider** property in **RevealSdkSettings**, as shown below.

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
   public Task<IRVDataSourceCredential> ResolveCredentialsAsync(RVDashboardDataSource dataSource)
      {
         IRVDataSourceCredential userCredential = null;
         if (dataSource is RVPostgresDataSource)
         {
            userCredential = new RVUsernamePasswordDataSourceCredential("postgresuser", "password");
         }
         else if (dataSource is RVSqlServerDataSource)
         {
            userCredential = new RVUsernamePasswordDataSourceCredential("sqlserveruser", "password", "domain");
         }
         else if (dataSource is RVGoogleDriveDataSource)
         {
            userCredential = new RVBearerTokenDataSourceCredential("fhJhbUci0mJSUzi1nIiSint....", "user@company.com");
         }
         else if (dataSource is RVRestDataSource)
         {
            userCredential = new RVUsernamePasswordDataSourceCredential(); // Anonymous
         }
         return Task.FromResult<IRVDataSourceCredential>(userCredential);
   }
}
```

Choosing Which Class to Implement

There are two classes that can be used, both implementing the **IRVDataSourceCredential** interface. You need to choose the class depending on your data source, as detailed below.

- Class **RVBearerTokenDataSourceCredential** works with:
    - Analytics tools (Google Analytics).
    - Content Managers and Cloud Services (Box, Dropbox, Google Drive, OneDrive and SharePoint Online).
- Class **RVUsernamePasswordDataSourceCredential** works with:
    - Customer Relationship Managers (Microsoft Dynamics CRM On-Premises and Online)
    - Databases (Microsoft SQL Server, Microsoft Analysis Services Server, MySQL, PostgreSQL, Oracle, Sybase)
- **Both classes** work with:
    - Other Data Sources (OData Feed, Web Resources, REST API).

No Authentication

Sometimes you might work with an anonymous resource, without authentication. In this particular case, you can use

RVUsernamePasswordDataSourceCredential, which has an empty constructor. You can do this for any data source that works with the class.

Code snippet to be used with the sample above:

```
else if (dataSource is RVRESTDataSource)
{
    userCredential = new RVUsernamePasswordDataSourceCredential();
}
```

Replacing Data Sources (MS SQL Server)

Overview

Before loading and processing the data for a dashboard (by Reveal SDK), you can override the configuration or data to be used for each visualization of the dashboard.

You would need to set the DataSourceProvider property of **RevealSdkSettings**:

```
public IRVDataSourceProvider DataSourceProvider { get; set; }
```

A class implementing the interface **IRVDataSourceProvider** may replace or modify the data source used by a given visualization or dashboard filter.

Use Cases

Below you can find a list of common use cases:

- You can change the name of the database being used, depending on the current user, or any other attributes your app might get like userId, division, company, customer, etc. By doing this, you can have a single dashboard getting data from a multi-tenant database.
- You can change the name of the table being used, the path of the file to load, etc. The use case is similar to the one described above.
- You can replace a data source with an in-memory data source. As the Reveal App doesn't support in-memory data sources, you can design a dashboard using a CSV file and then use this callback to replace the CSV data source with an in-memory one. In this scenario, data is actually loaded from memory (or with your custom data loader). For further details about how to use in-memory data sources, refer to **In-Memory Data Support**.

Code

The following code snippet shows an example of how to replace the data source for visualizations in the dashboard. The method **ChangeVisualizationDataSourceItemAsync** will be invoked for every visualization, on every single dashboard being opened.

```
public class SampleDataSourceProvider : IRVDataSourceProvider
    {
        public Task<RVDataSourceItem> ChangeDashboardFilterDataSourceItemAsync(
            RVDashboardFilter globalFilter, RVDataSourceItem dataSourceItem)
        {
            return Task.FromResult<RVDataSourceItem>(null);
        }

    public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(
        RVVisualization visualization, RVDataSourceItem dataSourceItem)
    {
        var sqlServerDsi = dataSourceItem as RVSqlServerDataSourceItem;
        if (sqlServerDsi != null)
        {
            // Change SQL Server host and database
            var sqlServerDS = (RVSqlServerDataSource)sqlServerDsi.DataSource;
            sqlServerDS.Host = "10.0.0.20";
            sqlServerDS.Database = "Adventure Works";

            // Change SQL Server table/view
            sqlServerDsi.Table = "Employees";
                return Task.FromResult((RVDataSourceItem)sqlServerDsi);
        }

        // Fully replace a data source item with a new one
        if (visualization.Title == "Top Customers")
        {
            var sqlDs = new RVSqlServerDataSource();
            sqlDs.Host = "salesdb.local";
            sqlDs.Database = "Sales";

            var sqlDsi = new RVSqlServerDataSourceItem(sqlDs);
            sqlServerDsi.Table = "Customers";

            return Task.FromResult((RVDataSourceItem)sqlServerDsi);
        }

        return Task.FromResult((RVDataSourceItem)dataSourceItem);
    }
}
```

In the example above, the following two replacements will be performed:

- All data sources using a MS SQL Server database will be changed to use the hardcoded server “10.0.0.20”, the “Adventure Works” database, and the “Employees” table.

  Note

  This is a simplified scenario, replacing all visualizations to get data from the same table makes no sense as a real world scenario. In real-world applications, you’re probably going to use additional information like userId, dashboardId, or the values in the data source itself (server, database, etc.) to infer the new values to be used.

- All widgets with the title “Top Customers” will have their data source set to a new SQL Server data source, getting data from the “Sales” database in the “salesdb.local” server, using the table “Customers”.

Please note that in addition to implement **IRVDataSourceProvider**, you need to set your implementation in the **DataSourceProvider** property of **RevealSdkSettings**:

```
RevealSdkSettings.DataSourceProvider = new SampleDataSourceProvider();
```

Replacing Excel and CSV file DataSources

Overview

When we create a dashboard in **Reveal Application** we often use Excel or CSV files stored in the cloud to populate it with data.
After exporting the dashboard and embedding it in custom application, we can move these files in a local directory and then use the **Reveal SDK** to access and set them as a local datasource.

Steps

To populate the exported dashboard using local Excel and CSV files, you need to follow these steps:

1. **Export the dashboard** file as explained in **Getting Dashboards for the SDK**
2. **Load the dashboard** in your application as described in: **Loading Dashboard Files**
3. **Download the files** you used to create the dashboard from your cloud storage and copy them to a local folder.
4. **Set the local folder name** as a value of the *Reveal.SdkSettings.LocalFilesRootFolder* property.
   You can use the *Reveal.Sdk.Samples.UpMedia.Wpf* sample application as reference for setting *Datasources* as local folder. The sample application comes with **Reveal SDK** installation.
5. **Add a new *CloudToLocalDatasourceProvider* class** in the project.
6. **Copy the implementation code** from the relevant snippet in **Code** section below.
7. **Set the *DataSourceProvider* property** of the *RevealSdkContext* class to *CloudToLocalDatasourceProvider*:

```
public override IRVDataSourceProvider DataSourceProvider => new CloudToLocalDatasourceProvider();
```

Code

```
public class CloudToLocalDatasourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem> ChangeDashboardFilterDataSourceItemAsync(RVDashboardFilter filter, RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }
    public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(RVVisualization visualization, RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }
    protected Task<RVDataSourceItem> ProcessDataSourceItem(RVDataSourceItem dataSourceItem)
    {
        // Return data source unless it is an excel or csv file.
        if (dataSourceItem is RVExcelDataSourceItem == false &&
            dataSourceItem is RVCsvDataSourceItem == false)
        {
            return Task.FromResult(dataSourceItem);
        }

        var resourceBased = dataSourceItem as RVResourceBasedDataSourceItem;
        var resourceItem = resourceBased?.ResourceItem as RVDataSourceItem;
        var title = resourceItem?.Title;

        if (string.IsNullOrEmpty(title))
        {
            return Task.FromResult(dataSourceItem);
        }

        var localItem = new RVLocalFileDataSourceItem();

        // The SDK uses the local folder set as Reveal.SdkSettings.LocalFilesRootFolder
        localItem.Uri = @"local:/" + title;

        // The title assigned here is the original data source name.
        localItem.Title = title;
        resourceBased.ResourceItem = localItem;

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

Note

The *CloudToLocalDatasourceProvider* replaces automatically Excel and CSV files only. Other file types or data sources will remain unchanged. The replacement files should be the same ones used to create the dashboard or, alternatively, new files that share the **same schema** but with different data.

Working with the Localization Service

The Localization service allows you to localize different dashboard elements based on your custom logic. It also provides you with the ability to set custom formatting settings for fields.

Supported Elements for Localization

Dashboard elements you can localize:

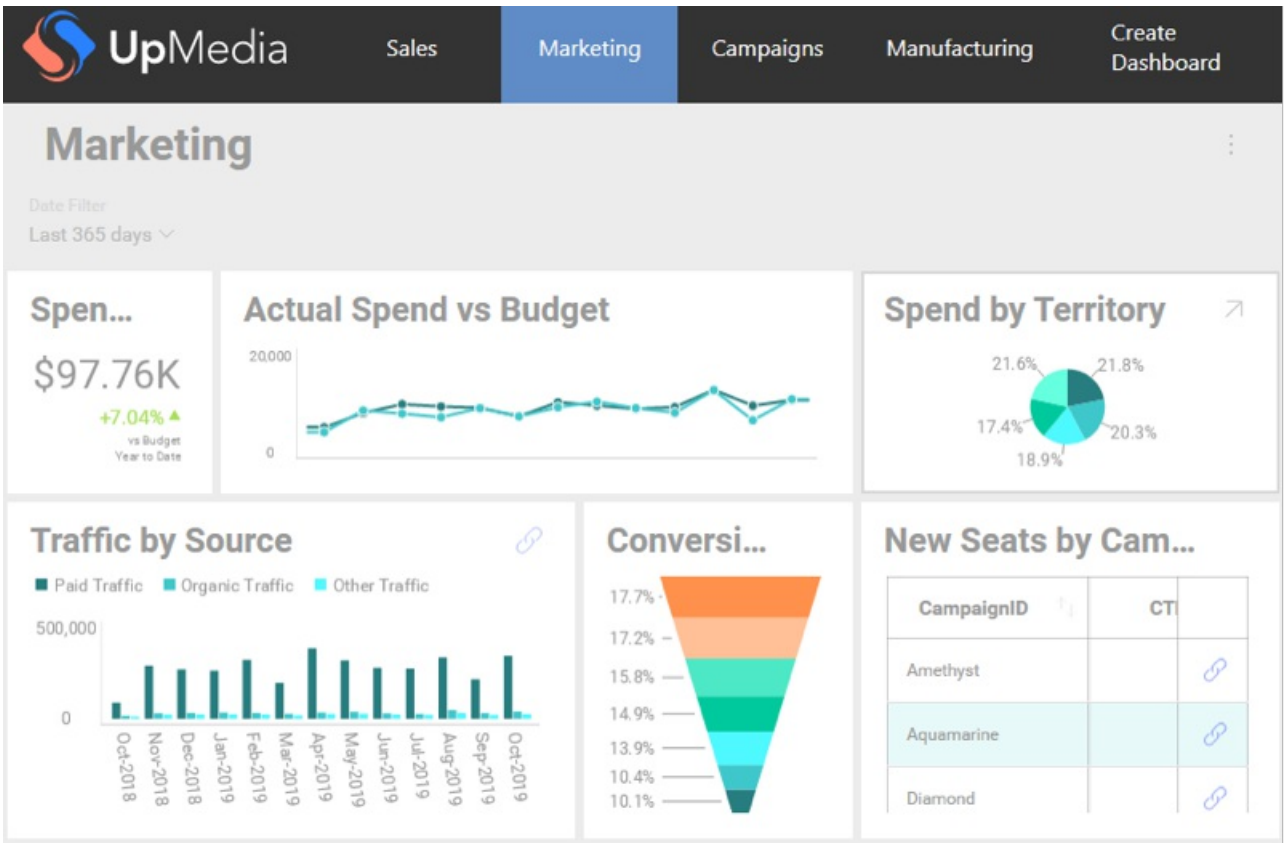
- Dashboard Title
- Filter Title
- Visualization Title
- Field Label
- Summarization Field Label

Using the Localization service

Below you will find two examples of how to **localize the dashboard title** and how to **localize a field label** in the same dashboard by adding custom logic. You will also find examples of how to **change the formatting settings of a numeric field and of a non-aggregated date field**. The dashboard used for the example is the *Marketing* sample dashboard.

Localizing a dashboard title - example

The initial state of the *Marketing* sample:



Follow the steps below to localize the *Marketing* dashboard title to *Localized Marketing*.

1. To be allowed to localize the dashboard, you should set the **LocalizationProvider** property to your custom implementation:

```
RevealSdkSettings.LocalizationProvider = new UpMediaLocalizationProvider()
```

2. Implement the **IRVLocalizationProvider**:

```
public class UpMediaLocalizationProvider : IRVLocalizationProvider
{
   public IRVLocalizationService GetLocalizationService()
   {
     return new UpMediaLocalizationService();
   }
}
```

3. Implement the **GetLocalizedString** method in the **IRVLocalizationService** as shown below to localize the dashboard title:

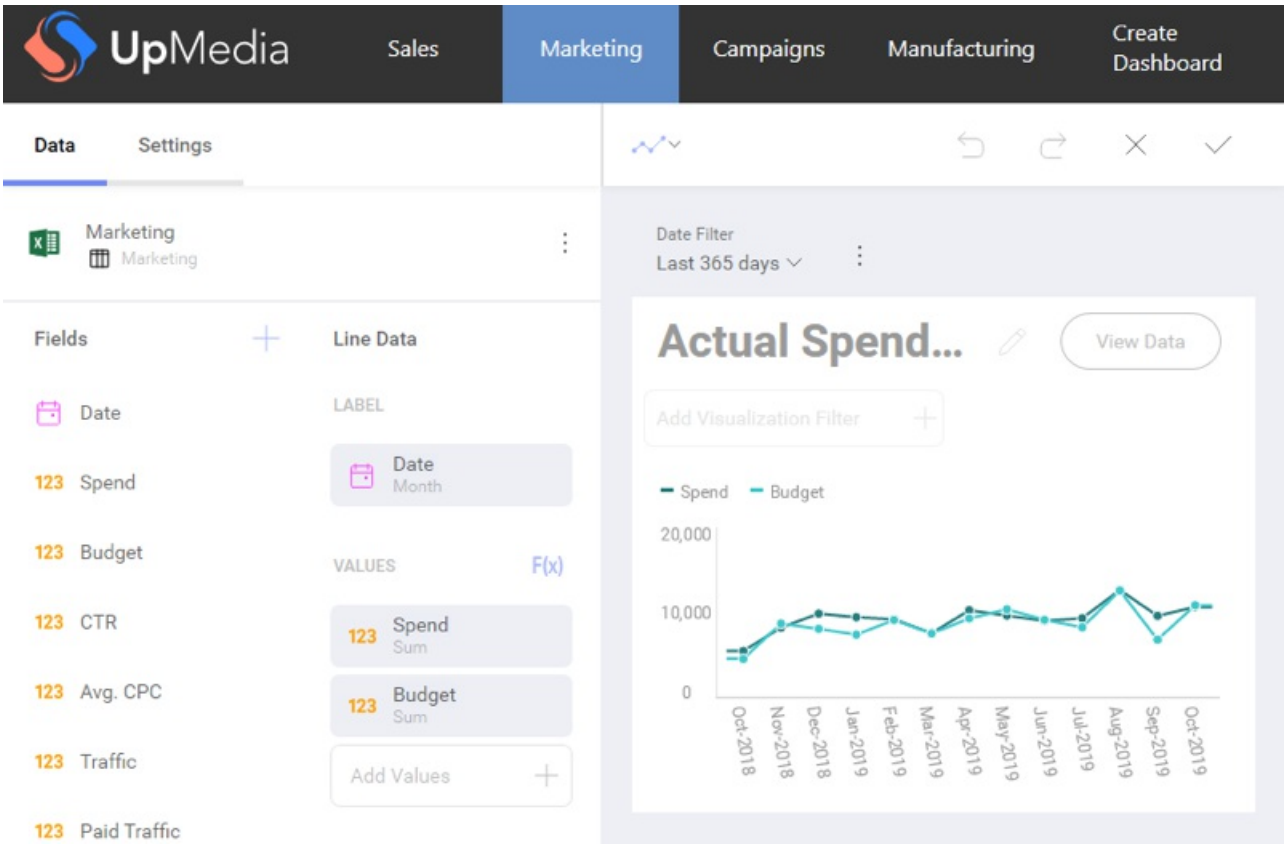
```
public class UpMediaLocalizationService : IRVLocalizationService
{
   public RVFormattingSpec GetFormattingSettingsForField(string fieldName, RVDashboardDataType dataType, RVFormattingSpec
currentSettings, bool isAggregated)
   {
      return null;
   }

   public string GetLocalizedString(string originalValue, RVLocalizationElementType type)
   {
      if (type == RVLocalizationElementType.DashboardTitle && originalValue == "Marketing")
      {
         return "Localized Marketing";
      }

      return originalValue;
   }
}
```

When running the app again you can see the localized dashboard title - *Localized Marketing*:

Localizing a field label - example

Below you will see an example of how to localize more than one element of the same dashboard.

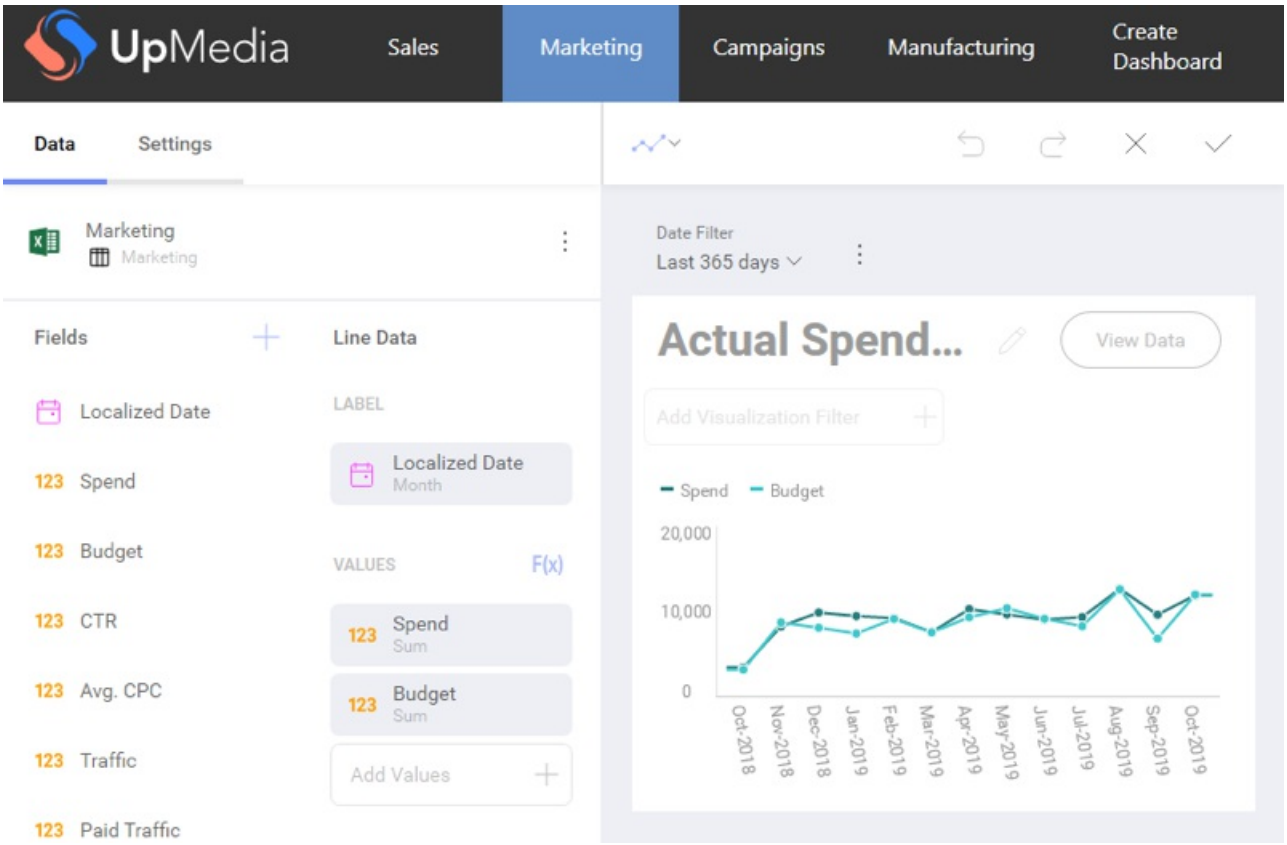
Here is the initial state of one of the *Marketing*'s sample visualizations - *Actual Spend vs Budget*:



To localize the *Date* field label, you need to add some logic to the **UpMediaLocalizationService** that will handle the localization of the *Date* field:

```
public class UpMediaLocalizationService : IRVLocalizationService
{
   public RVFormattingSpec GetFormattingSettingsForField(string fieldName, RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool
isAggregated)
   {
      return null;
   }

   public string GetLocalizedString(string originalValue, RVLocalizationElementType type)
   {
      if (type == RVLocalizationElementType.DashboardTitle && originalValue == "Marketing")
      {
         return "Localized Marketing";
      }
      else if (type == RVLocalizationElementType.FieldLabel && originalValue == "Date")
      {
         return "Localized Date";
      }

      return originalValue;
   }
}
```

The *Date* field label in *Actual Spend vs Budget* is now changed to *Localized Date*:



You can use the steps in the examples to localize any other dashboard element.

Using the Localization Service to Change Formatting Settings

Currently you can use the Localization service to change the formatting settings of numeric fields and non-aggregated date fields.

Changing the formatting settings of a numeric field - example

The initial state of the *Spend vs Budget* visualization below shows the numeric field formatted in the US Dollars ($) currency:

To change the currency format, you will need to create new formatting settings and return them in the **GetFormattingSettingsForField** method in the implementation of **IRVLocalizationService**.

The code snippet illustrates how to change the number formatting to the Japanese Yen (¥) and display it with no decimal digits:

```
public class UpMediaLocalizationService : IRVLocalizationService
{
   public RVFormattingSpec GetFormattingSettingsForField(string fieldName, RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool
isAggregated)
   {
      if (fieldName == "Spend" && dataType == RVDashboardDataType.Number && isAggregated == true)
      {
         var newSettings = new RVNumberFormattingSpec()
         {
            ApplyMkFormat = false,
            CurrencySymbol = "¥",
            DecimalDigits = 0,
            FormatType = RVDashboardNumberFormattingType.Currency,
            NegativeFormat = RVDashboardNegativeFormatType.MinusSign,
            ShowGroupingSeparator = true
         };

         return newSettings;
      }

      return null;
   }

   public string GetLocalizedString(string originalValue, RVLocalizationElementType type)
   {
     ...
   }
}
```

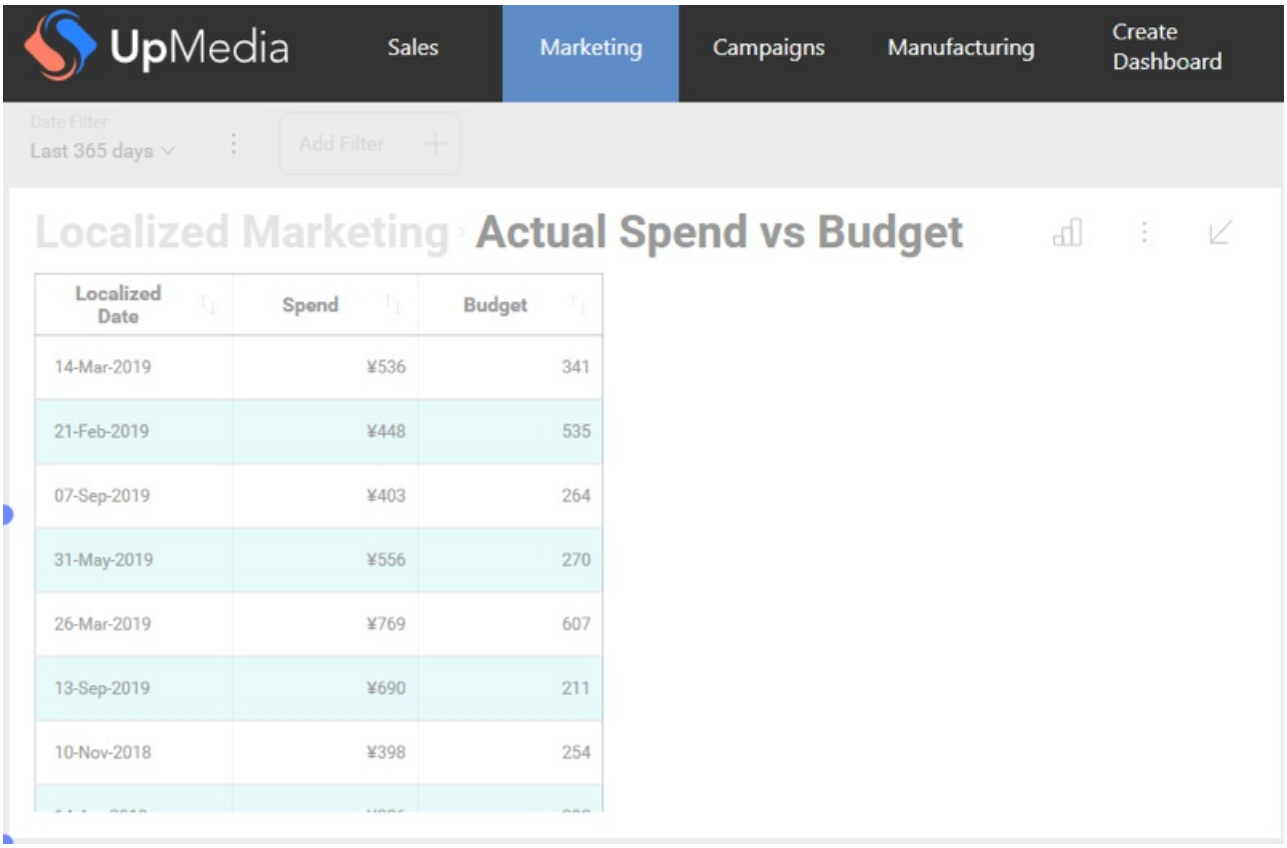
Now, the amount is displayed in a different currency:

Changing the formatting settings of a date field - example

Currently changing the formatting settings for an aggregated date field cannot be configured with the Localization service and won't affect the values in pivot. In order to achieve this you have to use the Formatting service.

The Localization service allows you to change the formatting settings of **non-aggregated date fields**.

First, change the *Actual Spend vs Budget* visualization to **Grid**, in order to exclude any aggregated data:



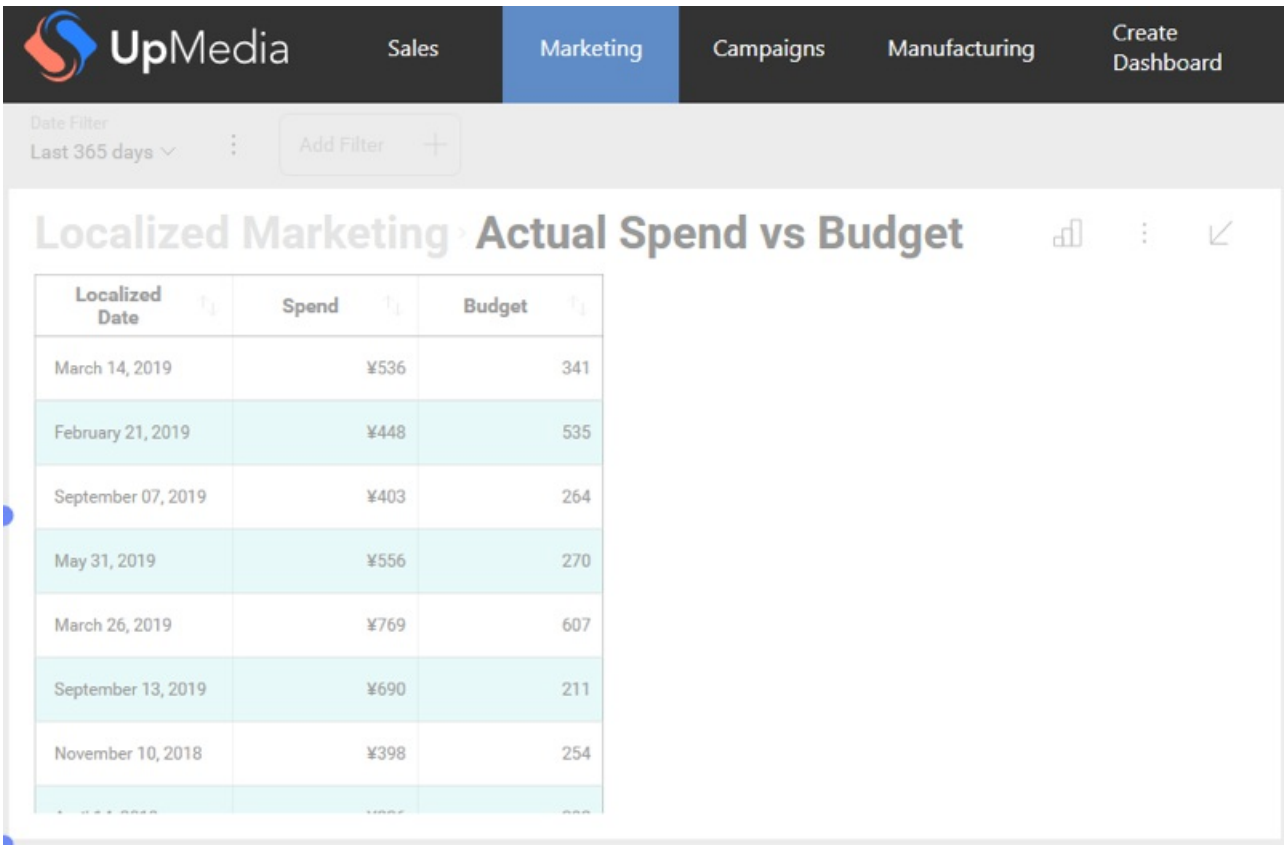
To change the formatting settings of the *Date* field, you need to add your preferences to the logic of the **GetFormattingSettingsForField** method. The code snippet below shows how to change the date format to display the full name of the month, like: "January 01, 2001".

```
public RVFormattingSpec GetFormattingSettingsForField(string fieldName, RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool
isAggregated)
{
   if (fieldName == "Spend" && dataType == RVDashboardDataType.Number && isAggregated == true)
   {
      var newSettings = new RVNumberFormattingSpec()
      {
         ApplyMkFormat = false,
         CurrencySymbol = "¥",
         DecimalDigits = 0,
         FormatType = RVDashboardNumberFormattingType.Currency,
         NegativeFormat = RVDashboardNegativeFormatType.MinusSign,
         ShowGroupingSeparator = true
      };

      return newSettings;
   }
   else if (fieldName == "Date" && dataType == RVDashboardDataType.Date && isAggregated == false)
   {
      var newSettings = new RVDateFormattingSpec()
      {
         DateFormat = "MMMM dd,yyyy"
      };

      return newSettings;
   }

   return null;
}
```

Note

Note that you need to check if the field name is *Date*, not *Localized Date*. The reason is the formatting settings are applied based on the name of the field. In this case, *Date* is the name of the field and *Localized Date* is just the label displayed. When editing a dashboard, you can modify the field labels, but the field names keep their original names.

After you run the app again and change the visualization to Grid, you will see the updated date format:



Working with the Formatting Service

The Formatting service allows you to format your dashboard data to your preferences, ignoring a field formatting.

Supported Elements for Formatting

Dashboard elements you can format:

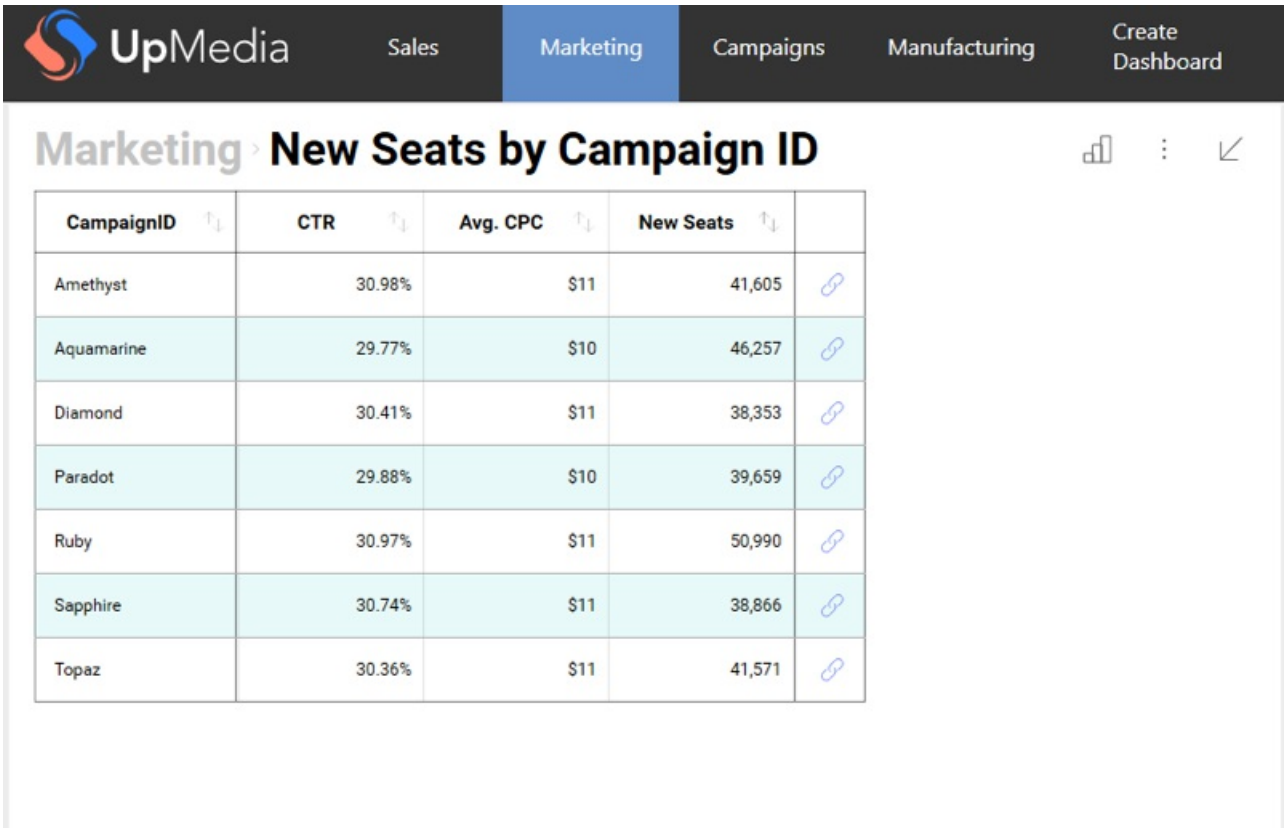
- Numeric data
- Date, Time or DateTime data
- Aggregated Date, Time or DateTime data

Using the Formatting Service

Below you will find three examples of how to format: numeric data, aggregated DateTime data and (non-aggregated) DateTime data. The dashboard used for the examples is the **Marketing sample** dashboard.

Formatting numeric data - example

Below you see the initial state of the **New Seats by Campaign ID** visualization in the **Marketing sample** dashboard:



Follow the steps below to format the numeric data to display 5 decimal digits:

1. To be allowed to format the dashboard data, you should set the **FormattingProvider** property to your custom implementation:

```
RevealSdkSettings.FormattingProvider = new UpMediaFormattingProvider();
```

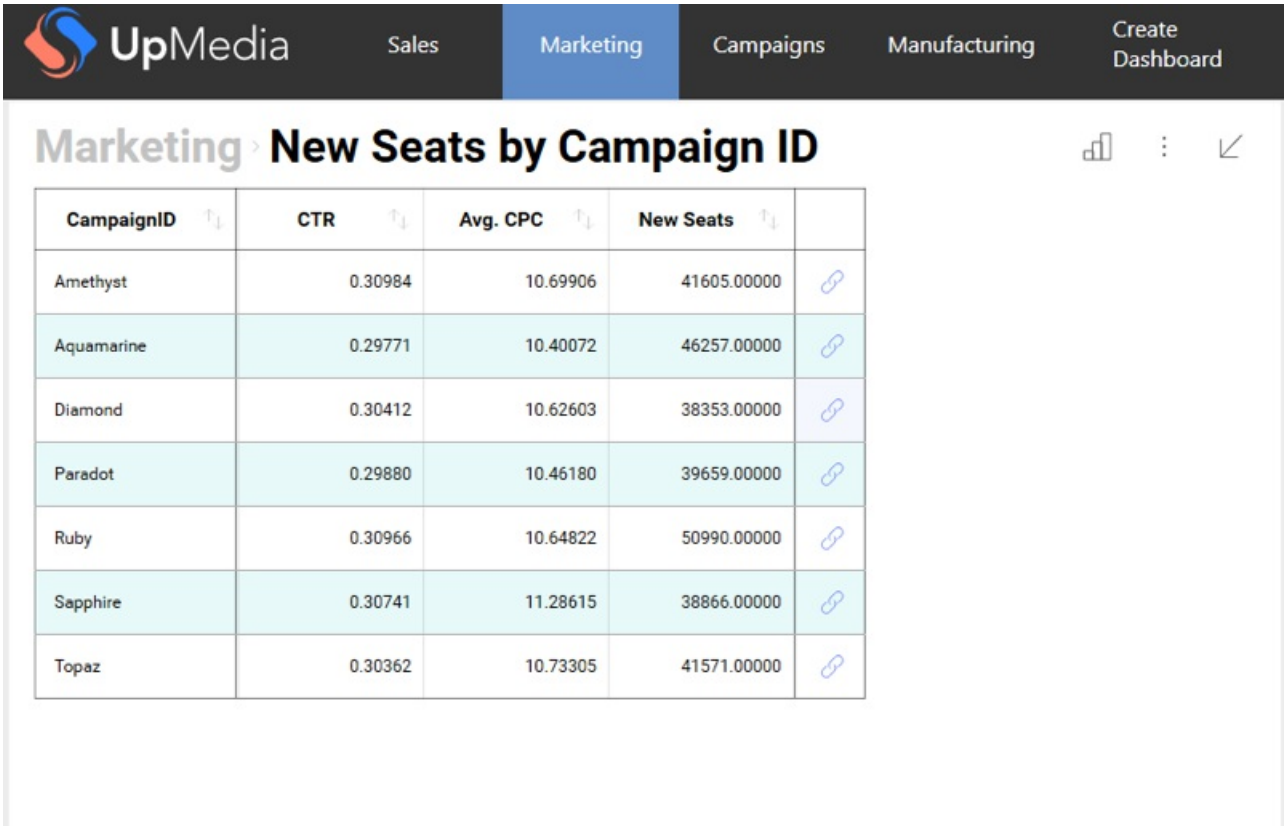
2. Implement the **IRVFormattingProvider**:

```
public class UpMediaFormattingProvider : IRVFormattingProvider
{
    public RVBaseFormattingService GetFormattingService()
    {
        return new UpMediaFormattingService();
    }
}
```

3. Override the **FormatNumber** method in the implementation of the **RVBaseFormattingService**:

```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatNumber(double value, RVNumberFormattingSpec formatting, bool ignoreMkFormatting)
    {
        return string.Format("{0:0.00000}", value);
    }
}
```

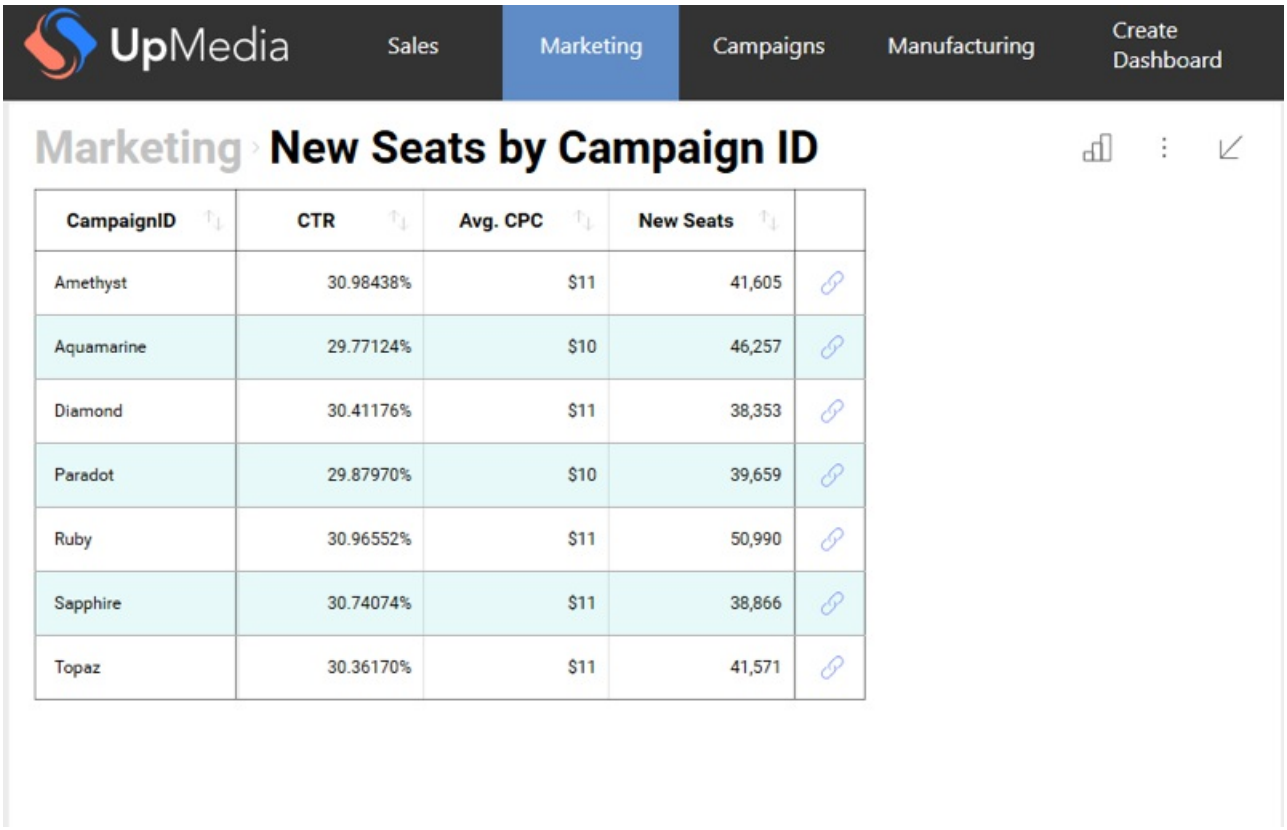
When running the app again you will see that all the numeric data is formatted to display numbers with 5 decimal digits and all other formatting settings (such as whether the field represents currency or percentage) are ignored:



If you want to change the formatting of only the fields displaying percentage, for example, you need to add a check for the type of the numeric field:

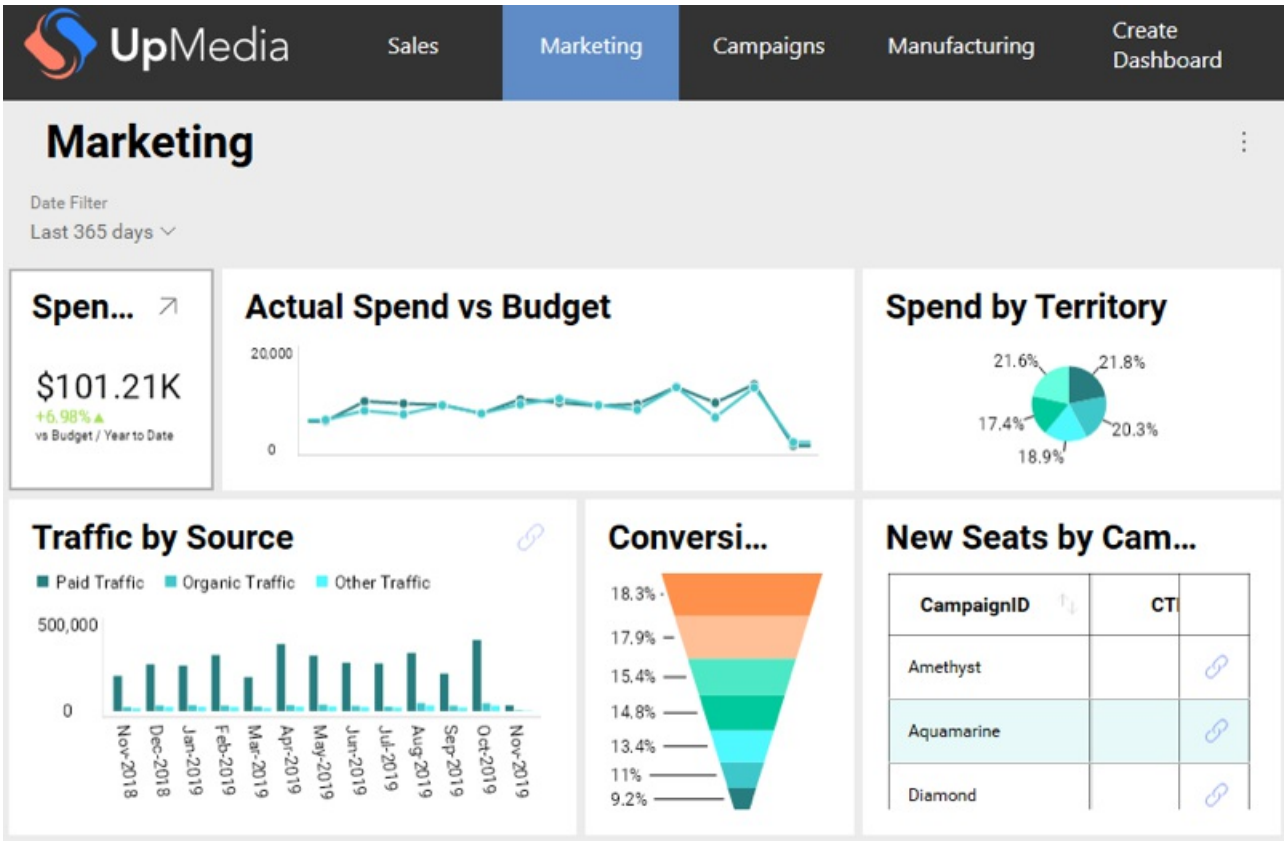
```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatNumber(double value, RVNumberFormattingSpec formatting, bool ignoreMkFormatting)
    {
        if (formatting.FormatType == RVDashboardNumberFormattingType.Percent)
        {
            return string.Format("{0:0.00000%}", value);
        }

        return base.FormatNumber(value, formatting, ignoreMkFormatting);
    }
}
```

Now you have only the percentage fields formatted:



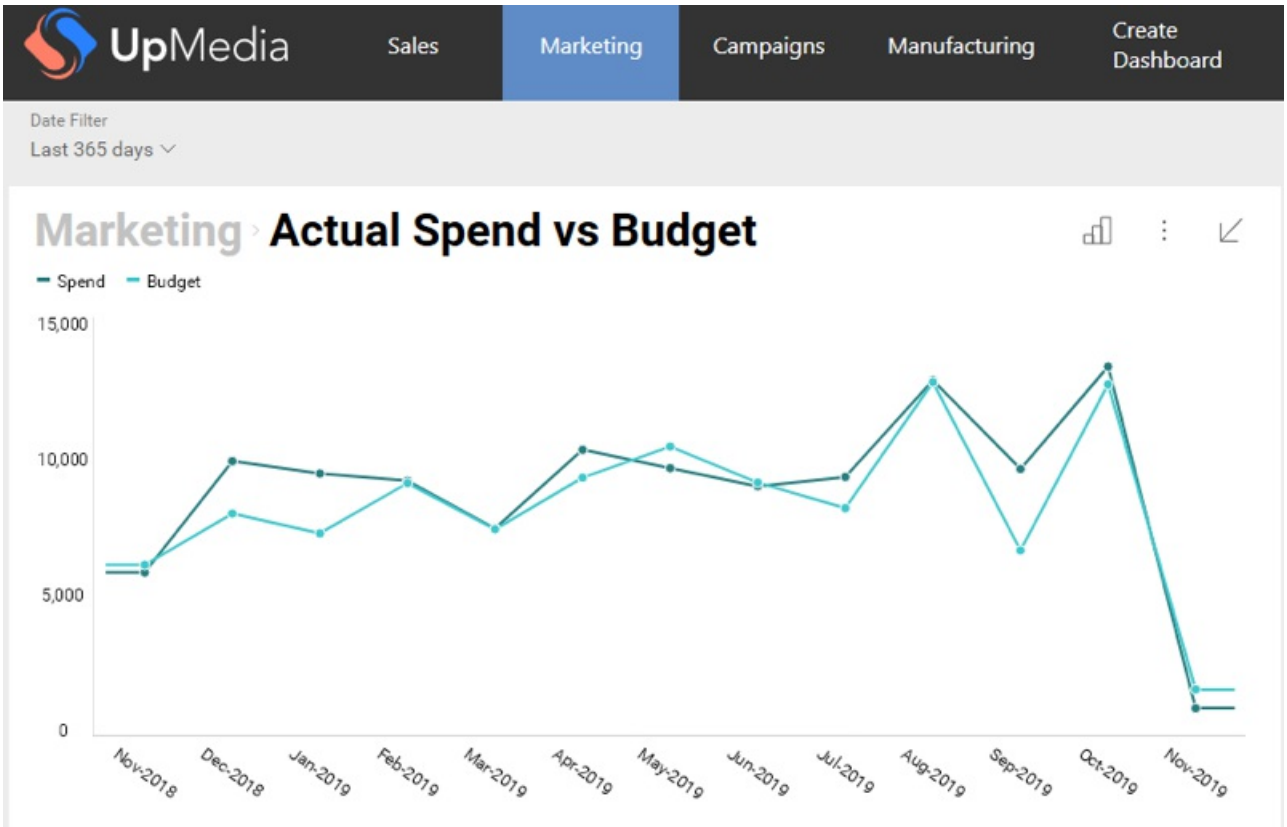
CampaignID	CTR	Avg. CPC	New Seats
Amethyst	30.98438%	$11	41,605
Aquamarine	29.77124%	$10	46,257
Diamond	30.41176%	$11	38,353
Paradot	29.87970%	$10	39,659
Ruby	30.96552%	$11	50,990
Sapphire	30.74074%	$11	38,866
Topaz	30.36170%	$11	41,571

If you take a look at the other visualizations in the dashboard, you will notice that the formatting of the numeric data in them is not changed. In these cases, the formatting is controlled by the chart, so you will have to modify the formatting settings of the fields in order to modify it. You can do this by using the Localization Service.



Formatting aggregated DateTime data - example

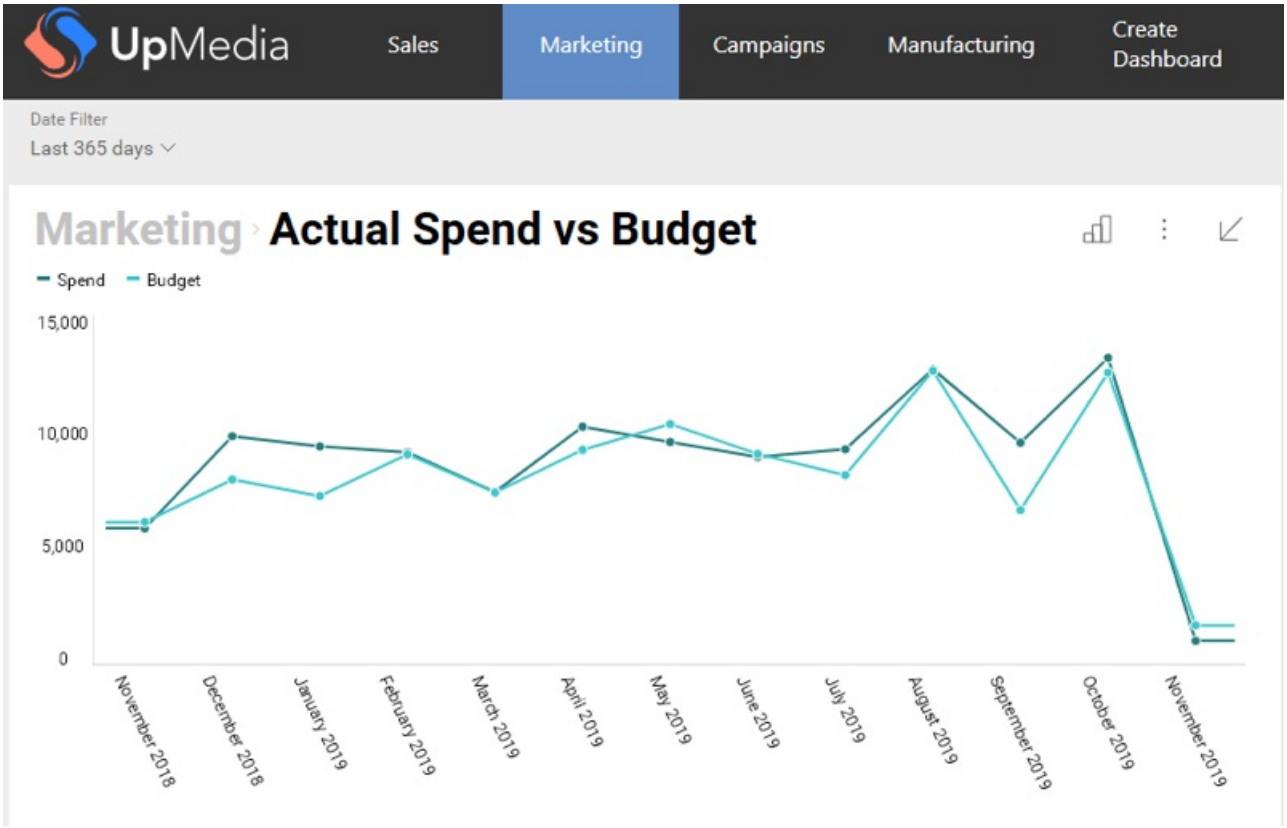
This is the initial state of one of the Marketing's sample visualizations – **Actual Spend vs Budget**:



Below, you will see an example of how to format the aggregated date data to show the full month name - for example "January 2001". To do this, you need to override the **FormatAggregatedDate** method in your implementation of **RVBaseFormattingService**:

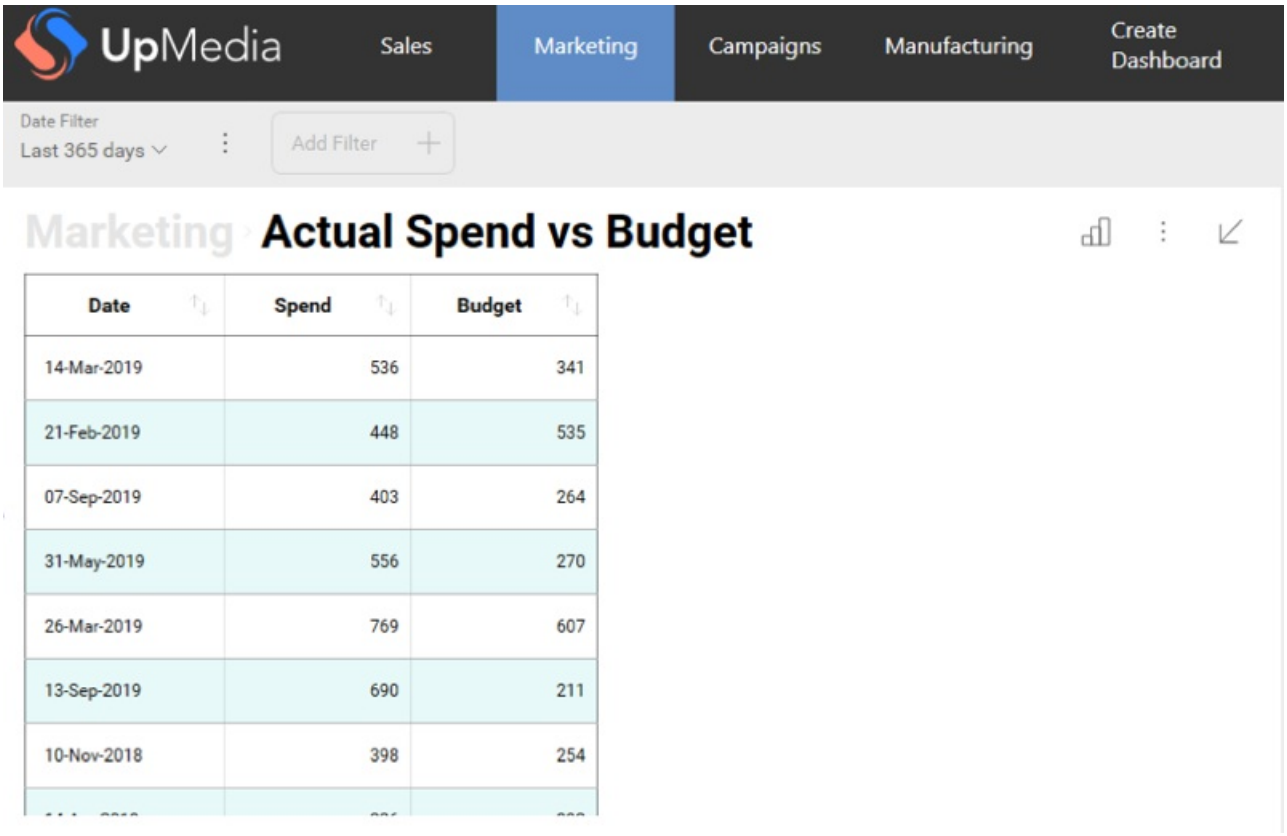
```
public class UpMediaFormattingService : RVBaseFormattingService
{
   public override string FormatAggregatedDate(DateTime value, RVDashboardDataType type, RVDashboardDateAggregationType aggregation,
RVDateFormattingSpec formatting)
   {
      if (aggregation == RVDashboardDateAggregationType.Month)
      {
         return string.Format("{0:MMMM yyyy}", value);
      }

      return base.FormatAggregatedDate(value, type, aggregation);
   }
}
```

After you run the app again you will see the updated dates:



Formatting (non-aggregated) DateTime data - example

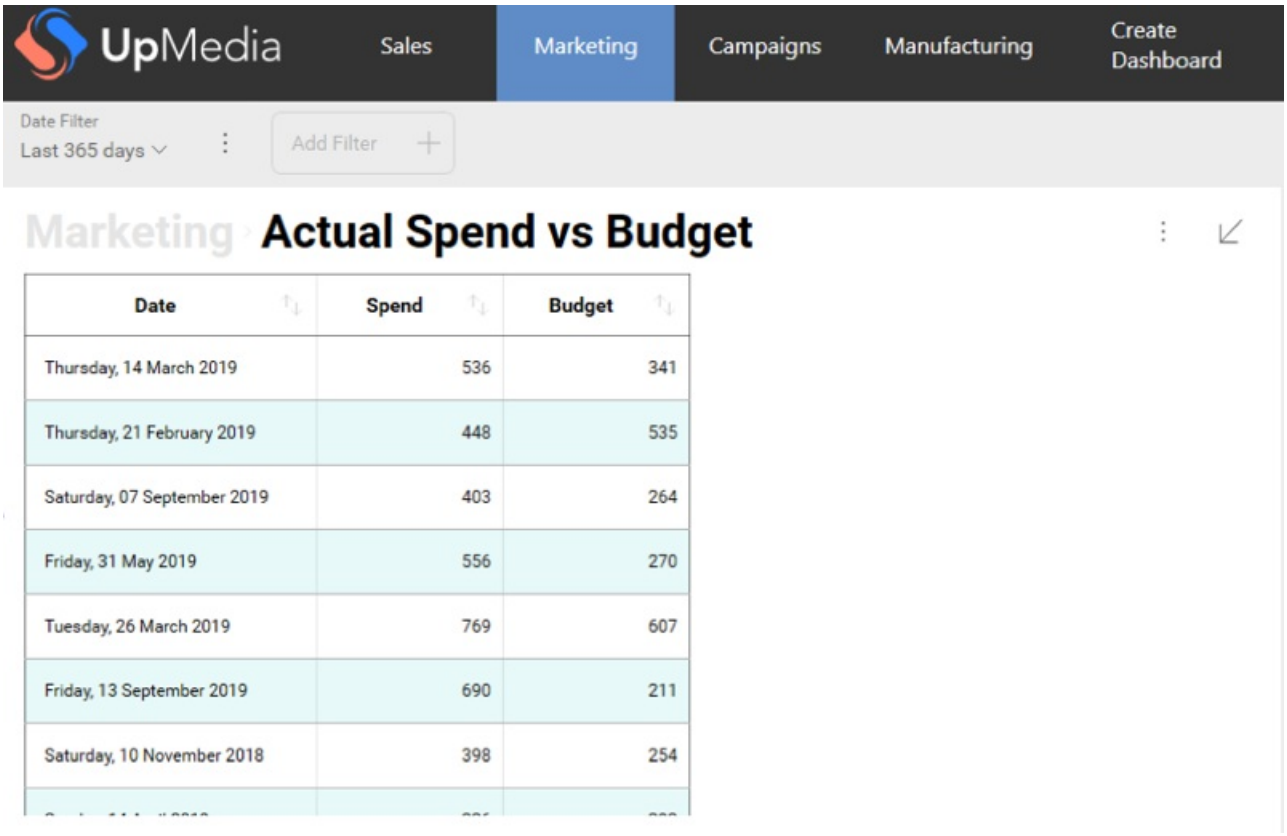
Below you will see an example of how to format non-aggregated date data. First, change the **Actual Spend vs Budget** visualization to **Grid** in order to exclude any aggregated data:



To change how the dates are displayed you have to override the **FormatNumber** method in your implementation of **RVBaseFormattingService**. Let's make the dates containing the day of the week and date – for example "Monday, 01 January 2001":

```
public class UpMediaFormattingService : RVBaseFormattingService
{
   public override string FormatDate(DateTime value, RVDashboardDataType type, RVDateFormattingSpec formatting, bool localTimeZone)
   {
      return string.Format("{0:dddd, dd MMMM yyyy}", value);
   }
}
```

And when you run the app again and change the visualization to Grid, you will see the updated dates (you may have to refresh the data because of the caching):



Date	Spend	Budget
Thursday, 14 March 2019	536	341
Thursday, 21 February 2019	448	535
Saturday, 07 September 2019	403	264
Friday, 31 May 2019	556	270
Tuesday, 26 March 2019	769	607
Friday, 13 September 2019	690	211
Saturday, 10 November 2018	398	254

Creating Custom Themes

Overview

When embedding analytics into your existing applications it is key that those dashboards match your app's look and feel. That's why you have full control over the Reveal dashboards through our SDK.

Creating your own theme in Reveal is done by setting the **Theme** property.

```
var regularFont = new FontFamily(new Uri("pack://application:,,,/ [Your ProjectName];component/[pathToFonts]/"), "./#Verdana Italic");
var boldFont = new FontFamily(new Uri("pack://application:,,,/ [Your ProjectName];component/[pathToFonts]/"), "./#Verdana Bold");
var mediumFont = new FontFamily(new Uri("pack://application:,,,/ [Your ProjectName];component/[pathToFonts]/"), "./#Verdana Bold Italic");

var currentTheme = RevealSdkSettings.Theme;
currentTheme.ChartColors.Clear();
currentTheme.ChartColors.Add(Color.FromRgb(192, 80, 77));
currentTheme.ChartColors.Add(Color.FromRgb(101, 197, 235));
currentTheme.ChartColors.Add(Color.FromRgb(232, 77, 137);

currentTheme.BoldFont = new FontFamily("Gabriola");
currentTheme.MediumFont = new FontFamily("Comic Sans MS");
currentTheme.FontColor = Color.FromRgb(31, 59, 84);
currentTheme.AccentColor = Color.FromRgb(192, 80, 77);
currentTheme.DashboardBackgroundColor = Color.FromRgb(232, 235, 252);

RevealSdkSettings.Theme = currentTheme;
```

Note

You first need to clear your chart colors list default values to have the new set of colors added.

If you have a dashboard or another Reveal component already displayed on your screen, you will need to render it again in order to see the applied changes.

Customizable Theme Settings

The settings that you can use to customize your theme are part of the *RevealTheme()* class. The *RevealTheme() class* contains all Dashboard and App settings with their current values. In the table below, you will find a full list of the customizable settings, followed by their type and a short description.

NAME	TYPE	DESCRIPTION
ChartColors	List	The colors used to show the series in your visualizations. You can add an unlimited number of colors. Once all colors are used in a visualization, Reveal will autogenerate new shades of these colors. This way your colors won't repeat and each value will have its own color.
AccentColor	Color	The default accent color in Reveal is a shade of blue that you can find in the + Dashboard button and other interactive actions. You can change the color to match the same accent color you use in your applications.
DashboardBackgroundColor	Color	Sets the background color of the dashboards. This is the main background color.
VisualizationBackgroundColor	Color	Sets the background color of the visualizations. This is a secondary background color.

NAME	TYPE	DESCRIPTION
ConditionalFormatting	RVConditionalFormatting	Changes the default colors of the bounds you can set when using conditional formatting.
RegularFont	FontFamily	Sets the regular font style.
BoldFont	FontFamily	Sets the bold font style.
MediumFont	FontFamily	Sets the medium font style.
FontColor	Color	Sets the color of the font.
HighlightColor	Color	Sets the highlighting color in specific dashboard scenarios (forecast and outliers statistical functions).
UseRoundedCorners	bool	(By default) Rounded corners in buttons, tooltips, containers, visualizations, etc. If set to false, squared corners will be shown.

Built-In Themes

Reveal SDK comes with four pre-built themes: *Mountain Light*, *Mountain Dark*, *Ocean Light*, and *Ocean Dark*. You can set the one that best matches your application's design, or you can also use it as the basis for your custom theme modifications.

Apply the settings of a chosen built-in theme by creating a new instance.

Mountain Light Theme

```
RevealSdkSettings.Theme = new MountainLightTheme();
```

Note

Mountain Light contains the default values for the customizable theme settings. This means MountainLight and the RevealTheme look basically the same way.

Mountain Dark Theme

```
RevealSdkSettings.Theme = new MountainDarkTheme();
```

Ocean Light Theme

```
RevealSdkSettings.Theme = new OceanLightTheme();
```

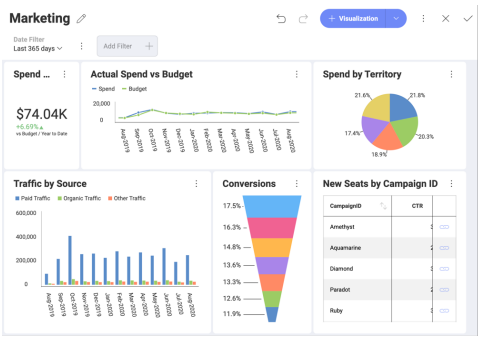
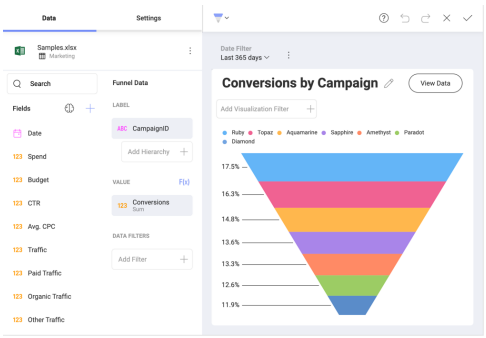
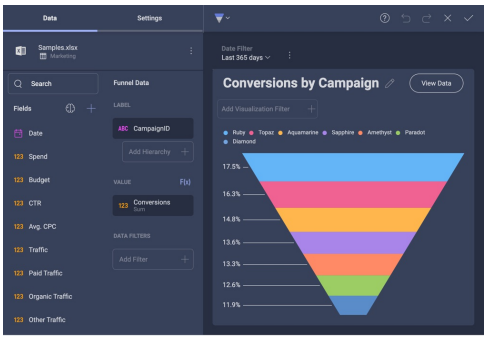
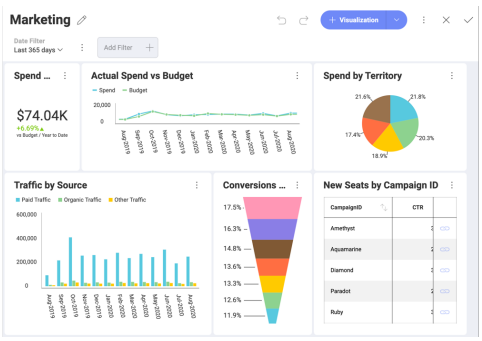
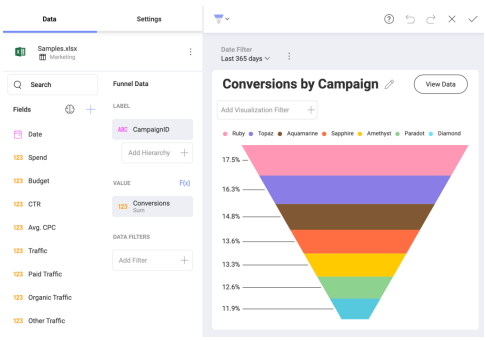
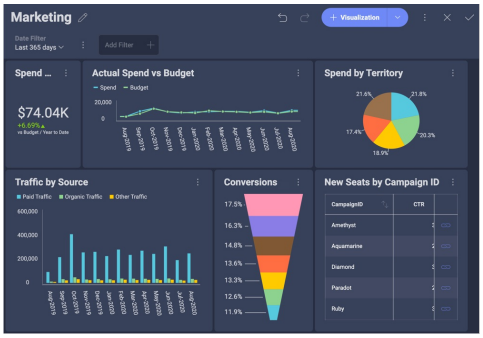
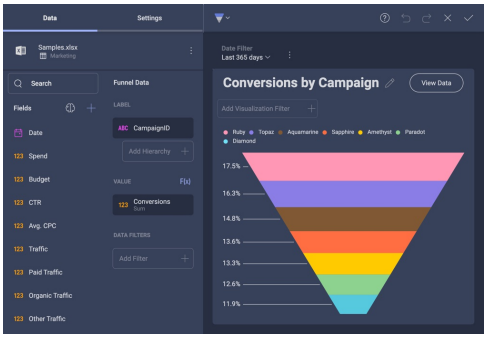
Ocean Dark Theme

```
RevealSdkSettings.Theme = new OceanDarkTheme();
```

How the Built-In Themes Look?

Below, you will find a table showing how the *Visualization Editor* and *Dashboard Editor* look when each of the pre-built themes is applied.

THEME	DASHBOARD EDITOR	VISUALIZATION EDITOR

THEME	DASHBOARD EDITOR	VISUALIZATION EDITOR
Mountain Light (Default?)		
Mountain Dark		
Ocean Light		
Ocean Dark		

Handling User Click Events

Overview

The SDK allows you to handle when the user clicks a cell with data within a visualization. This is very useful, for example, to provide your own navigation, to change existing selections in your app, among others.

Code

You can handle user click events by registering to the **VisualizationDataPointClicked** event:

```
//attach to VisualizationDataPointClicked event
revealView.VisualizationDataPointClicked += RevealView_VisualizationDataPointClicked;
```

In the callback function you receive information about the location of the click:

- the name of the visualization clicked;
- the values for the cell clicked (including value, formatted value, and the column's name);
- the rest of the values in the same cell.

```
private void RevealView_VisualizationDataPointClicked(object sender,
                VisualizationClickedEventArgs e)
{
   var vizTitle = e.Visualization.Title;
   var column = e.Cell.ColumnName;
   var formattedValue = e.Cell.FormattedValue;
   var value = e.Cell.Value;
   foreach (var c in e.Row)
   {
      var cValue = c.Value;
      //TODO: use value from column c.ColumnName
   }
}
```

You can use the rest of the values in the cell to search a specific attribute, like customer ID if you want to change the selected customer in your application. Doesn't matter that the user clicked another cell, like the sales amount for that customer, you'll still get the information you need.

Working with Tooltips

Overview

There is an event that is triggered whenever the end-user hovers over a series in a visualization or clicks on the series (as shown in the image below). This event called **revealView.TooltipShowing**, gives you more flexibility regarding how you show Tooltips in your visualizations.

Common Use Cases

You can cancel the Tooltip event or modify what is shown to the user. Most common examples include:

- You want to disable tooltips altogether or only show them for specific visualizations.
- You want to display data in the tooltip that is outside of the RevealView component that might be more valuable to your viewers.

Please note that this event will not be triggered for visualizations that do not support Tooltips, such as grids, gauges, and others.

Code Example

In the following code snippet, you can see how to disable tooltips for a visualization and still get additional information from the event arguments when the end-user hovers over or clicks on this visualization.

```
private void RevealView_TooltipShowing(object sender, TooltipShowingEventArgs e)
{
    if (e.Visualization.Title == "NoNeedForTooltips")
    {
        e.Cancel = true;
    }
    Debug.WriteLine($"TooltipShowing: Visualization: {e.Visualization.Title}, Cell: {e.Cell}, Row: {e.Row}");
}
```

The event arguments include information about the visualization that is being hovered over or clicked on, the exact cell of data hovered over or clicked, the whole row of this cell (in case you need information from other columns), and, of course, the Cancel boolean.

Showing/Hiding User Interface Elements

The **RevealView** component can be used to enable or disable different features and/or UI elements towards the end user. Many of the available properties are of the Boolean type and can be very straightforward to use, but others not so much.

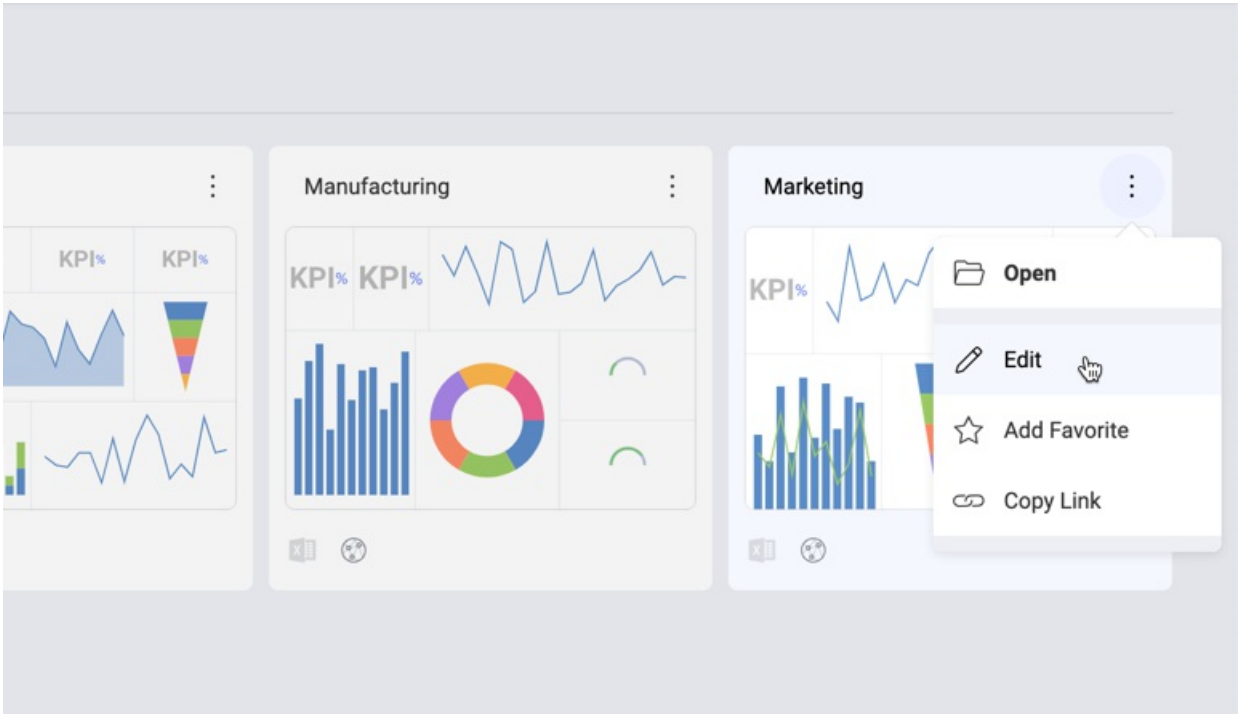
The *revealView* instance created below is assumed by all the code snippets in this topic:

```
var revealView = new RevealView();
```

All the properties are read by **RevealView** during initialization time and based on their values Reveal either shows or hides the different features/UI elements from the user.

CanEdit

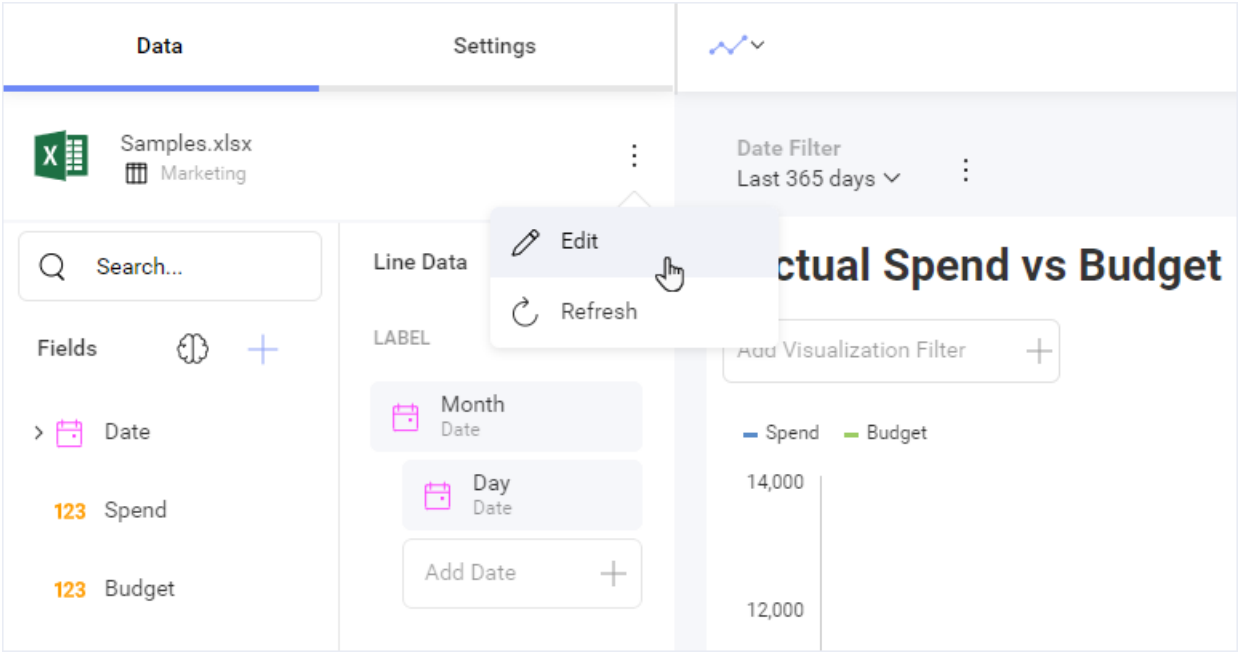
This property can be used to disable the user's ability to edit dashboards.




```
revealView.CanEdit = false;
```

ShowEditDataSource

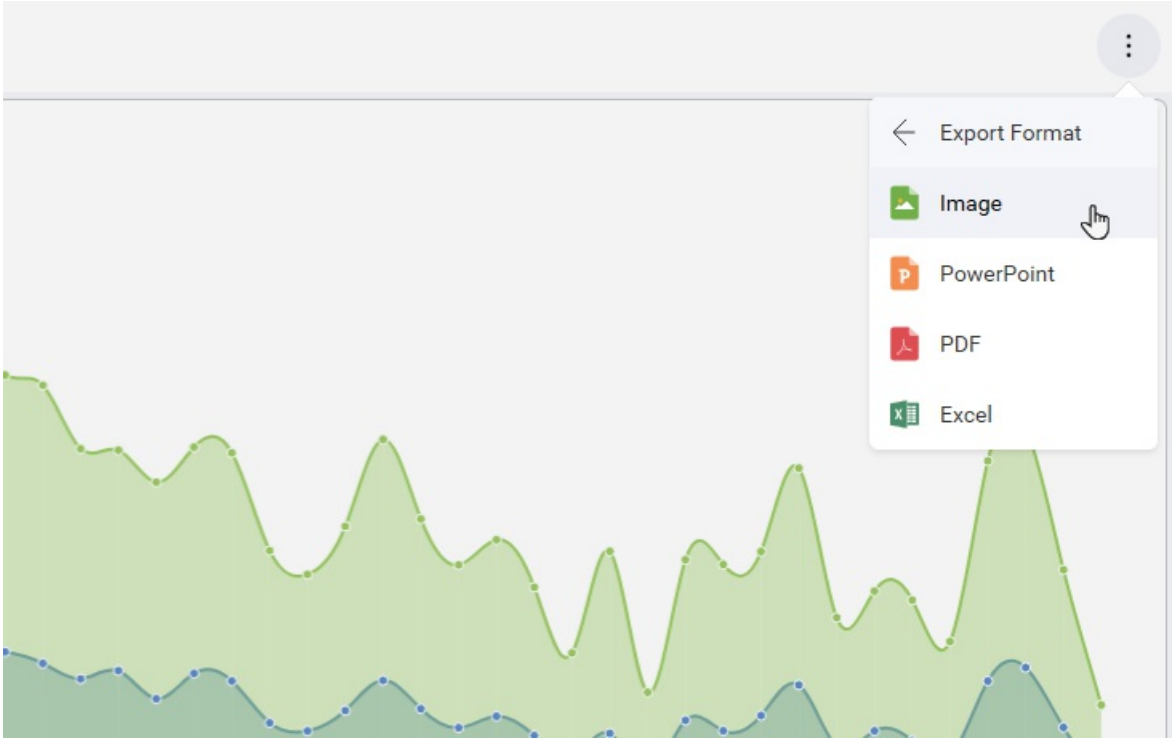
This property can be used to disable the editing of a dashboard datasource.




```
revealView.ShowEditDataSource = false;
```

ShowExportImage

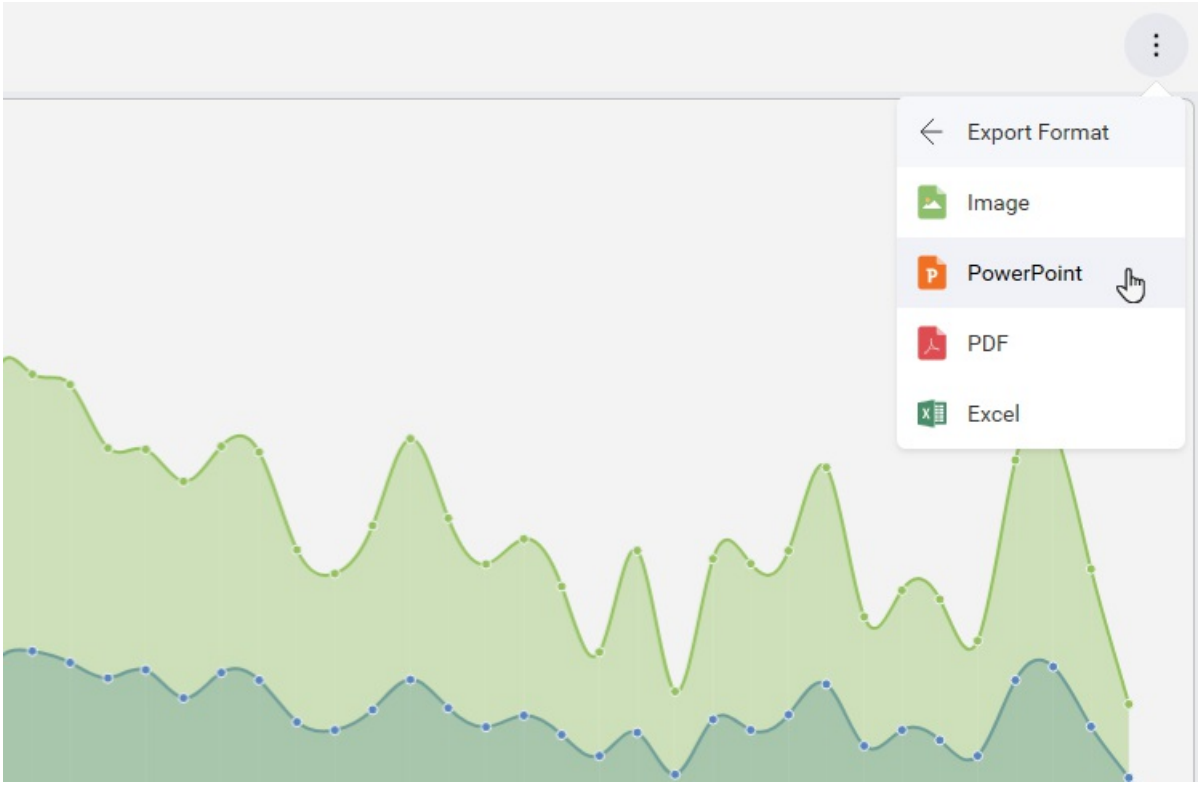
This property can be used to disable exporting the dashboad to an image.




```
revealView.ShowExportImage = false;
```

ShowExportToPowerpoint

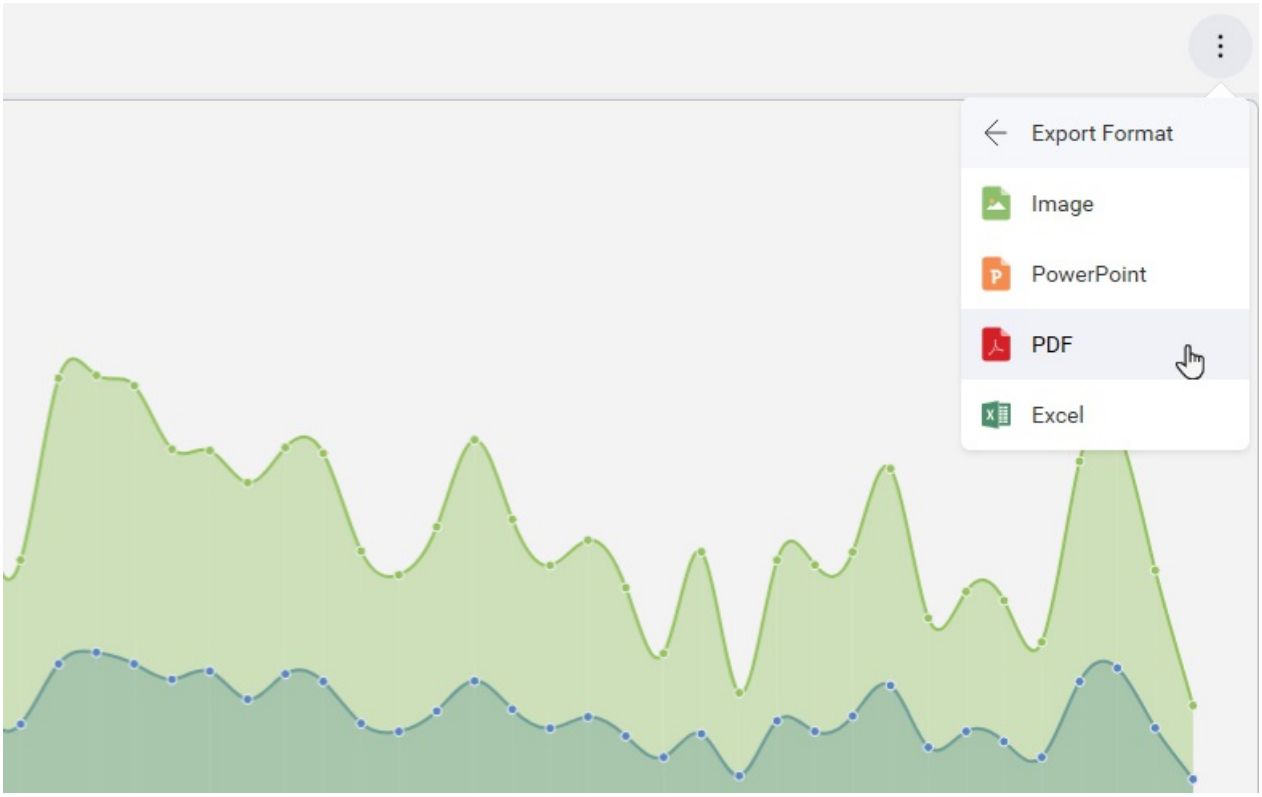
This property can be used to disable exporting the dashboad to PowerPoint.




```
revealView.ShowExportToPowerpoint = false;
```

ShowExportToPDF

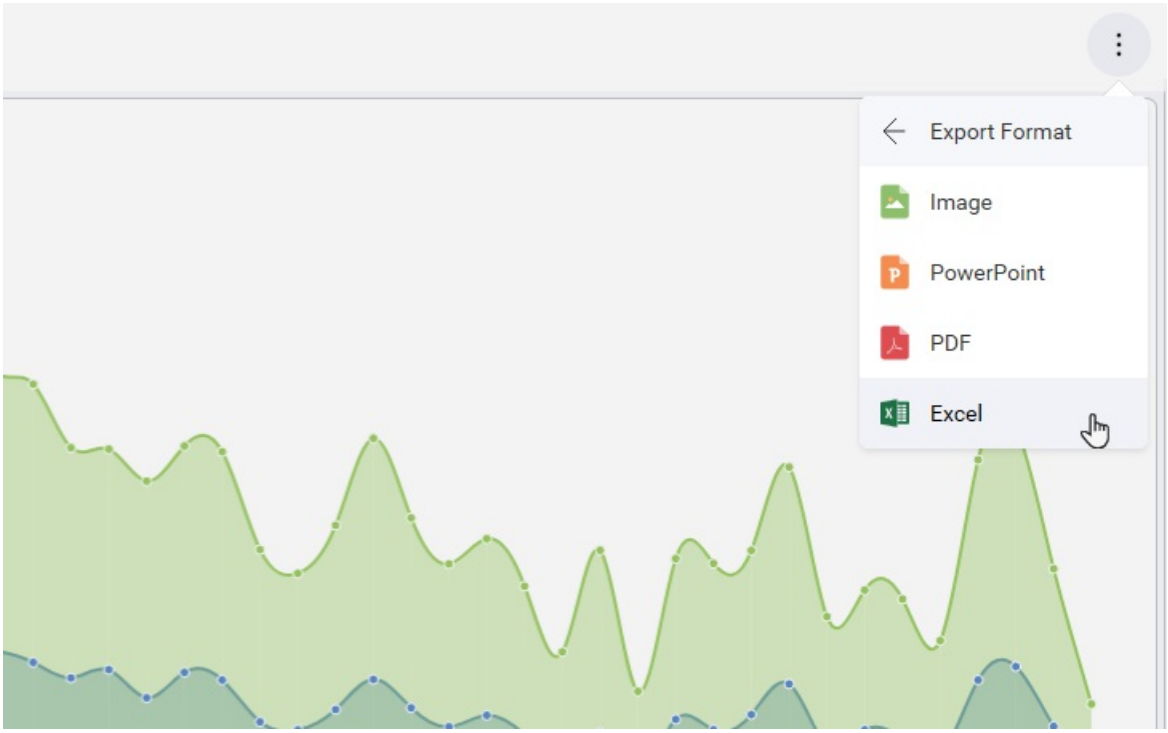
This property can be used to disable exporting the dashboad to PDF.




```
revealView.ShowExportToPDF = false;
```

ShowExportToExcel

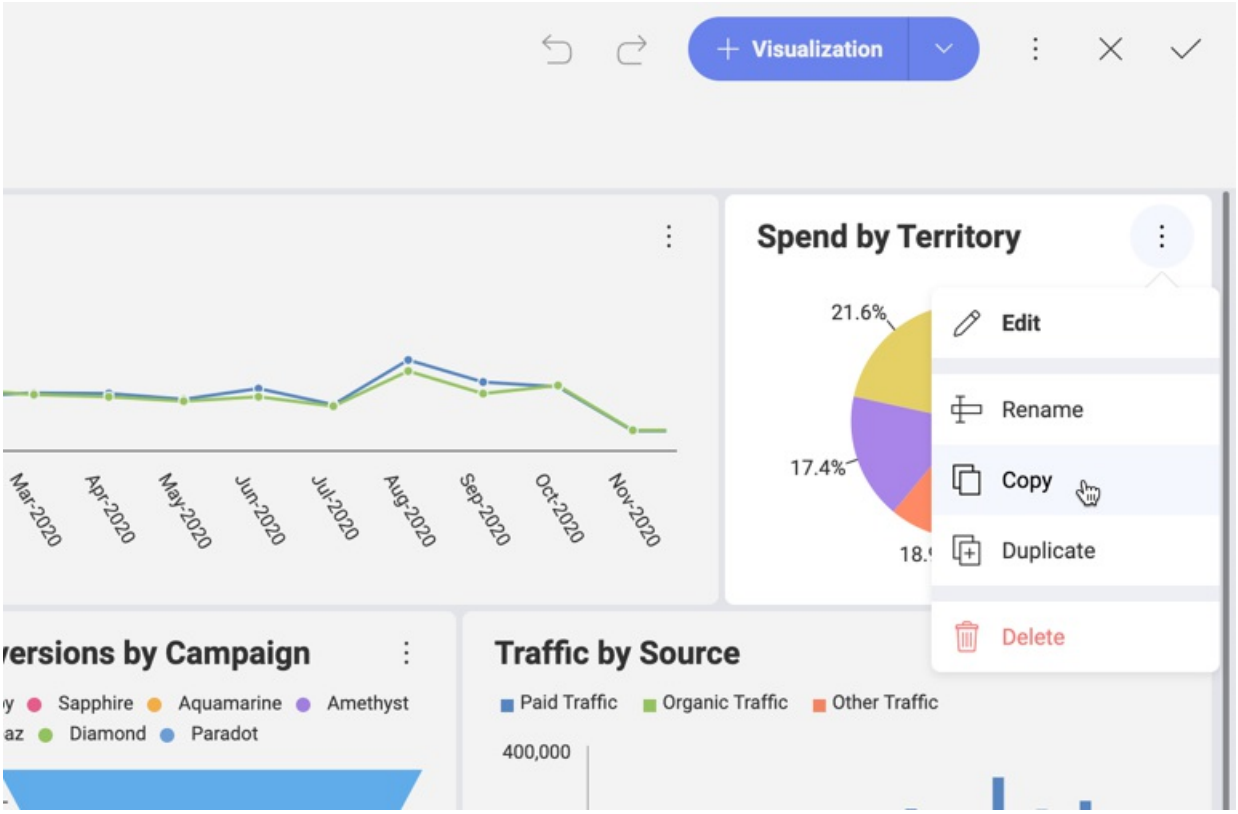
This property can be used to disable exporting the dashboad to Excel.




```
revealView.ShowExportToExcel = false;
```

CanCopyVisualization

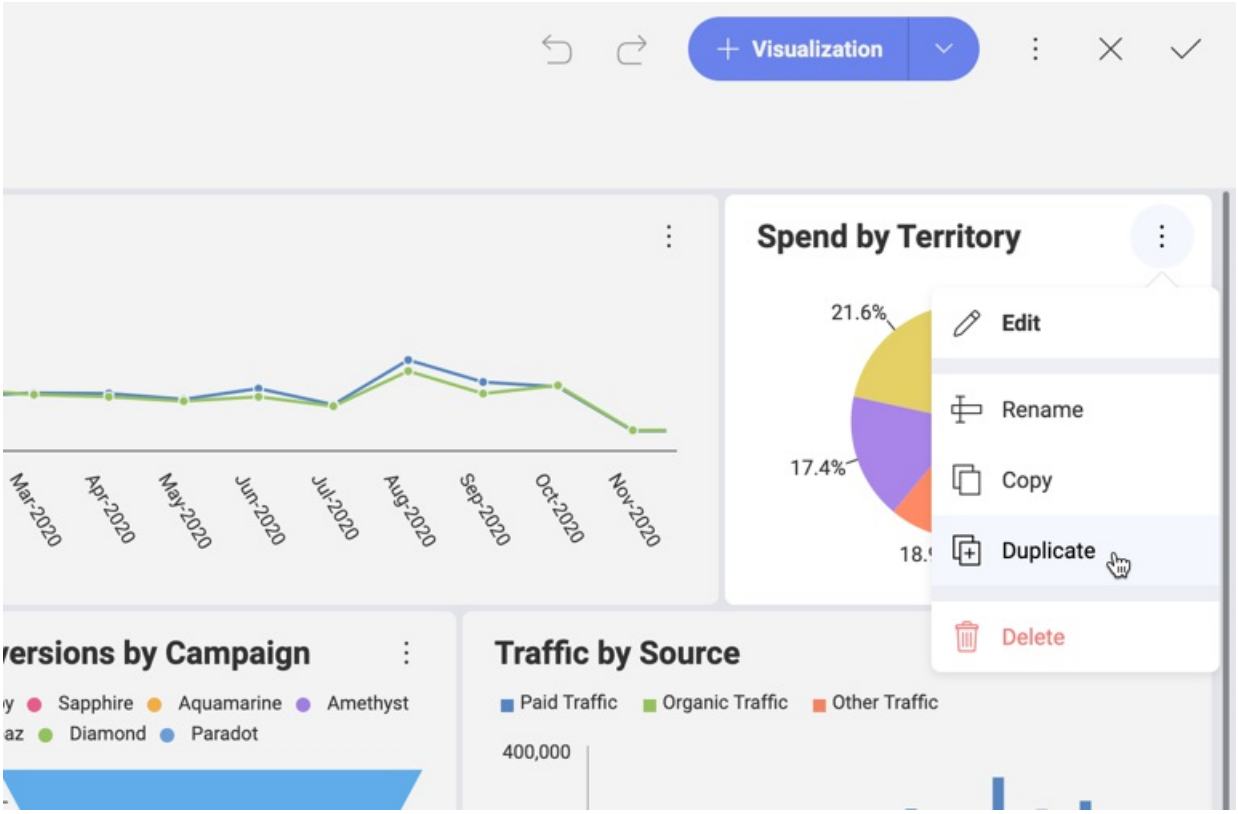
This property can be used to disable the ability to copy a visualization and later paste it in the current dashboard or a different one.




```
revealView.CanCopyVisualization = false;
```

CanDuplicateVisualization

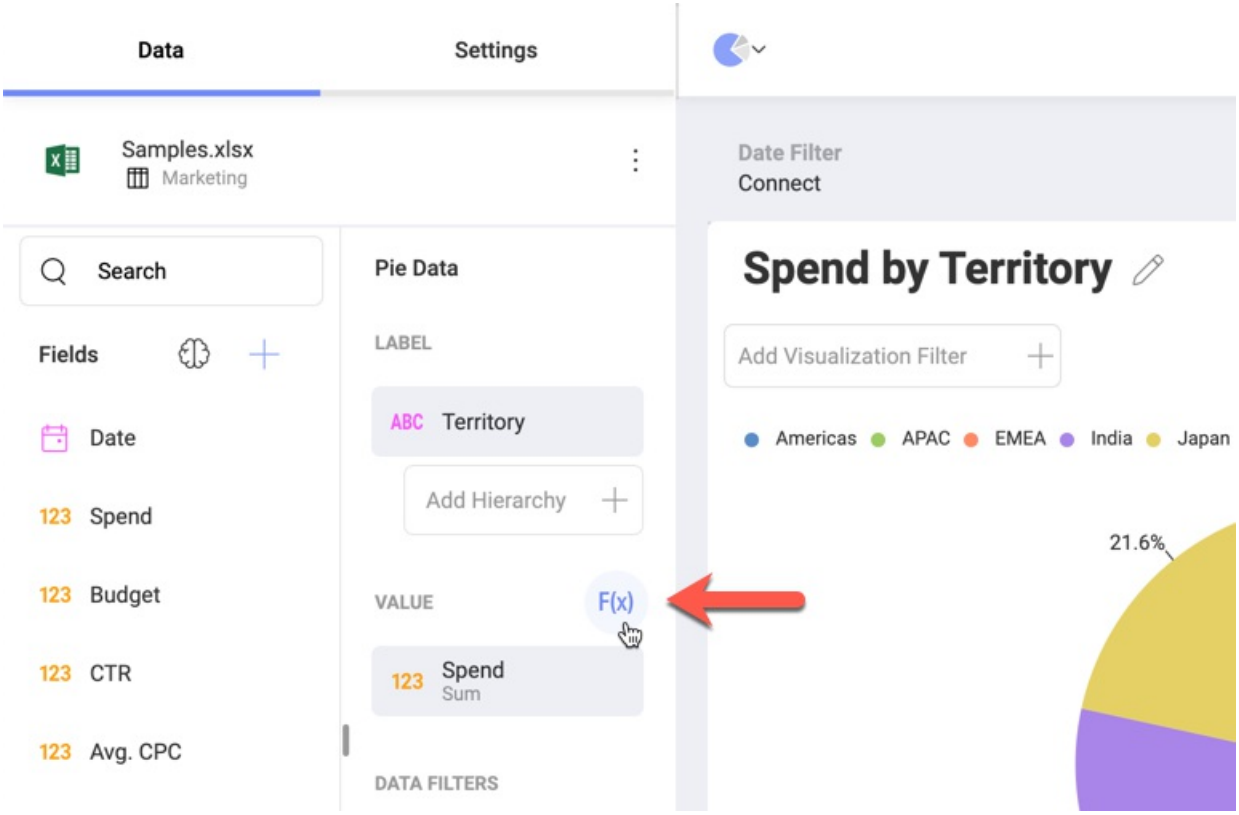
This property can be used to disable the ability to duplicate a visualization in the current dashboard.




```
revealView.CanDuplicateVisualization = false;
```

CanAddPostCalculatedFields

This property can be used to disable the ability to add a new post-calculated field in the current dashboard.

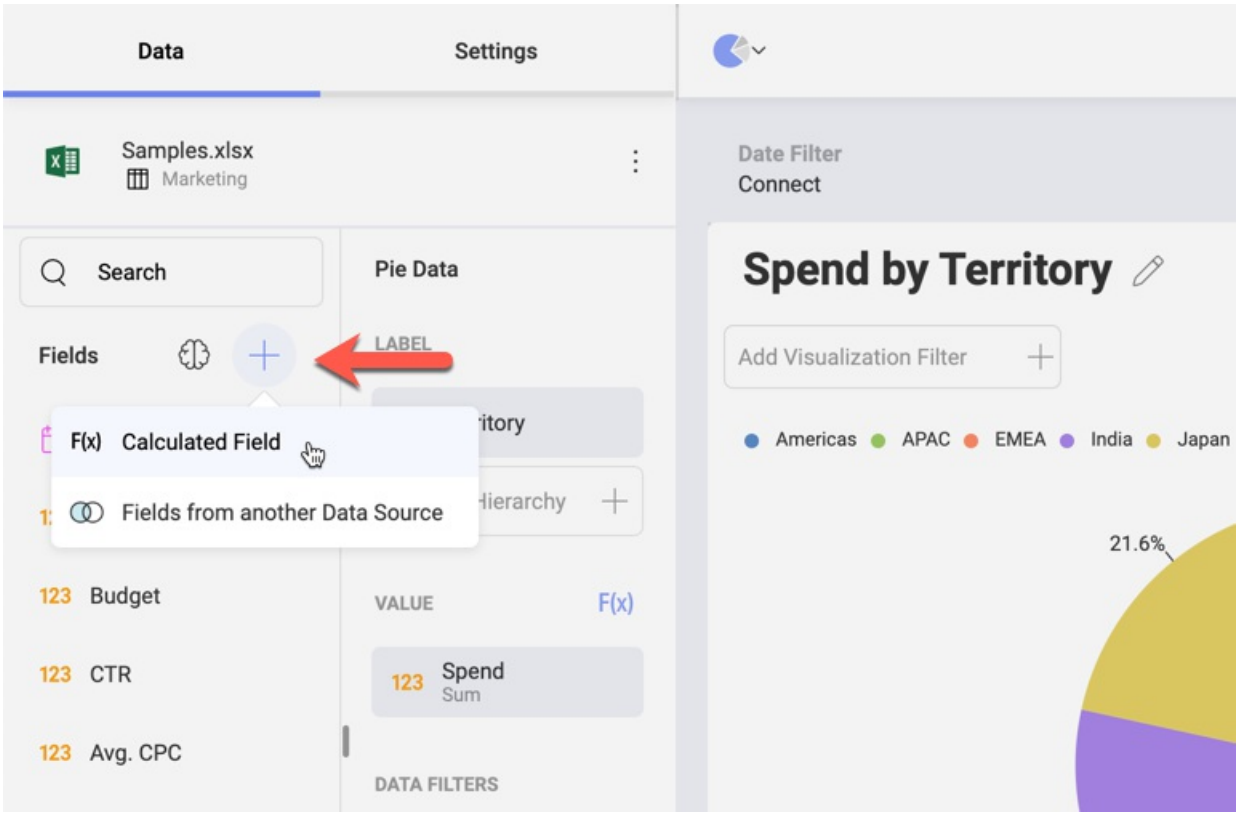


Post-calculated fields are new fields in the data set and are created by applying a formula on already summarized values.
For further details, please refer to the Reveal Help.

```
revealView.CanAddPostCalculatedFields = false;
```

CanAddCalculatedFields

This property can be used to disable the ability to add a new pre-calculated field in the current dashboard.

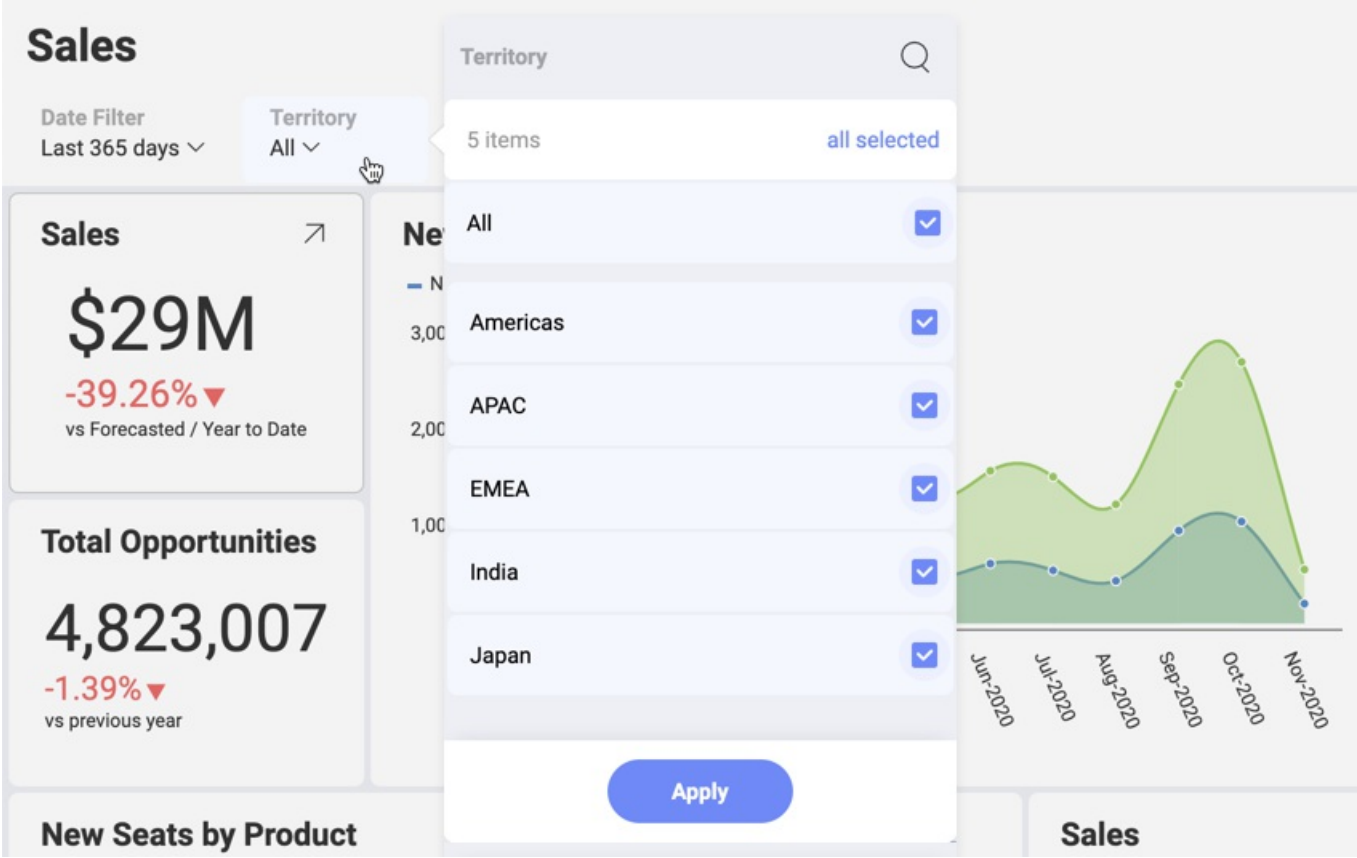


Pre-calculated fields are new fields in the data set and are evaluated before executing data editor aggregations.
For further details, please refer to the Reveal Help.

```
revealView.CanAddCalculatedFields = false;
```

ShowFilters

This property can be used to show or hide the Dashboard Filters UI to the user.

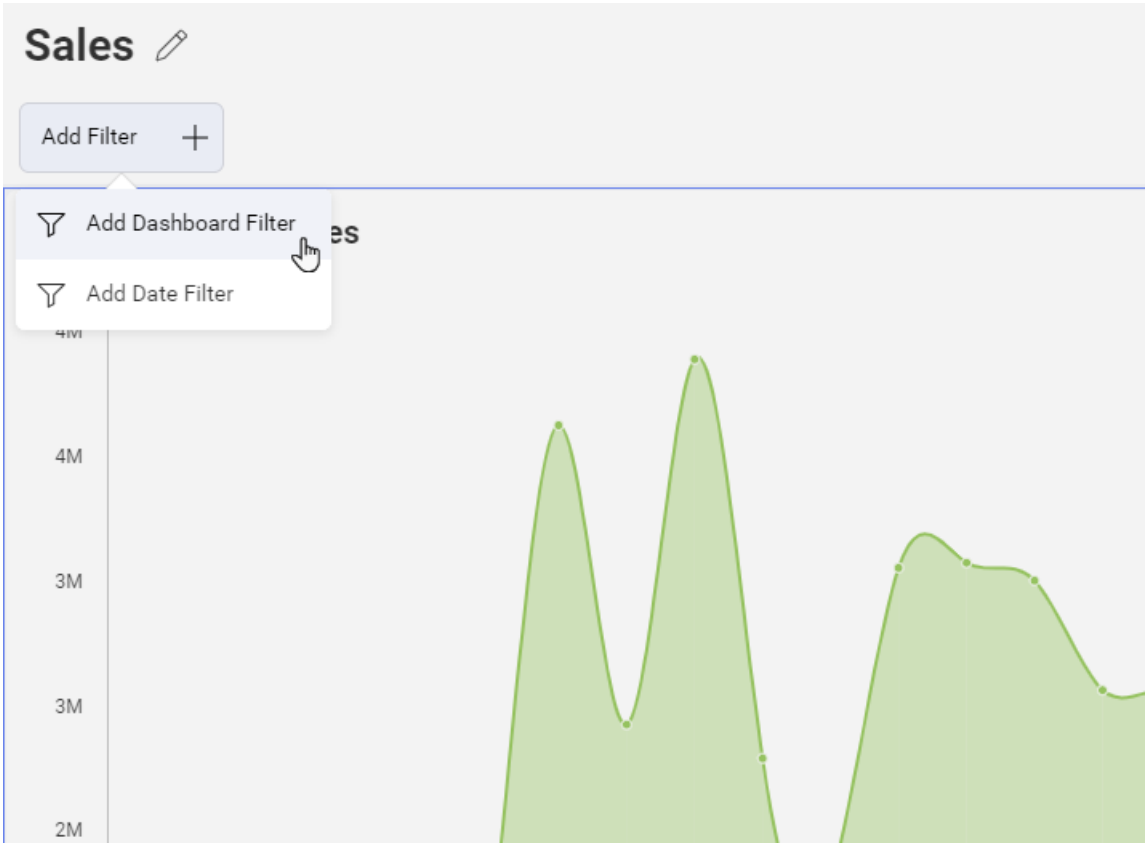


Dashboard filters allow you to slice the contents of the visualizations in a dashboard, all at once.

```
revealView.ShowFilters = false;
```

CanAddDashboardFilter

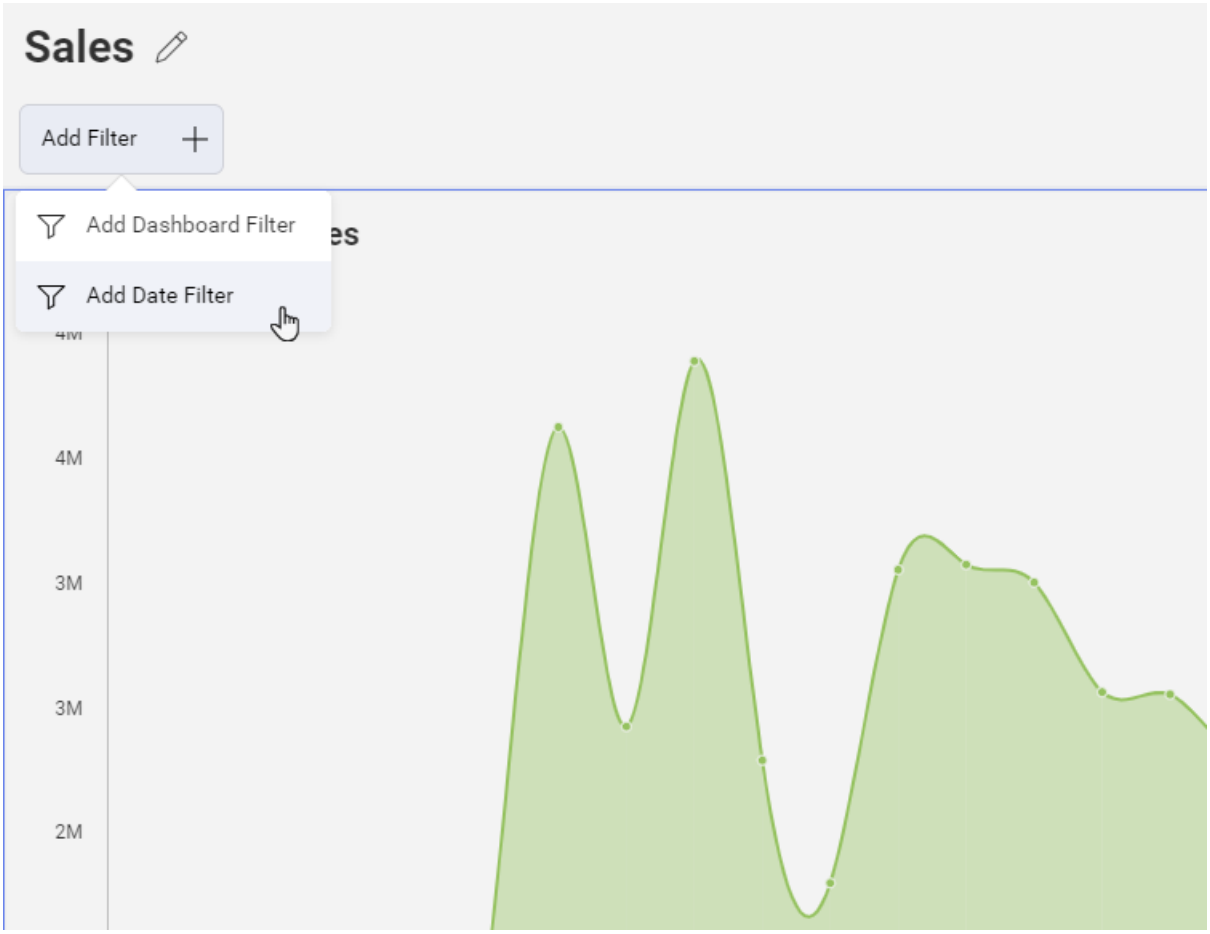
This property can be used to show or hide the Add Dashboard Filter menu item.



```
revealView.CanAddDashboardFilter = false;
```

CanAddDateFilter

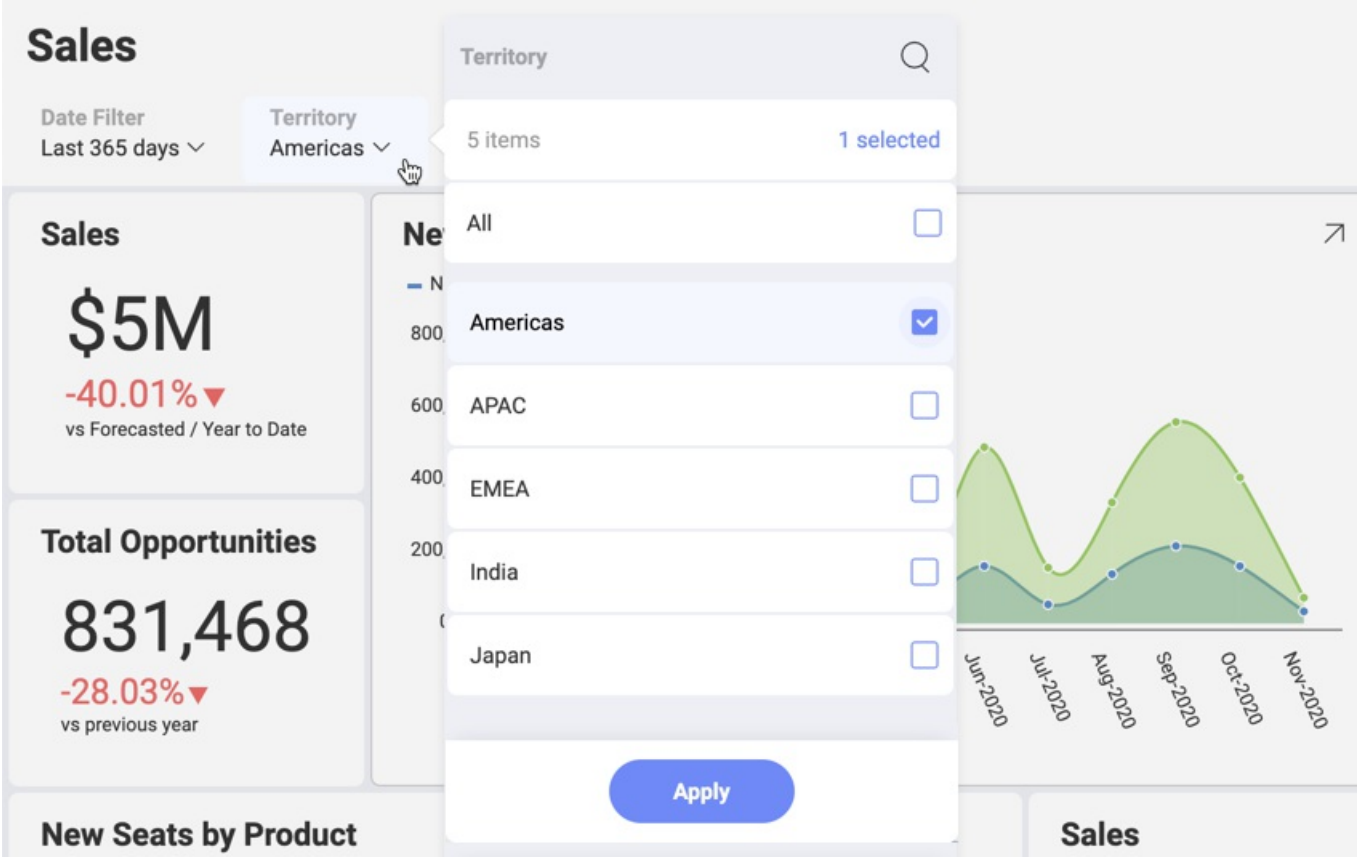
This property can be used to show or hide the Add Date Filter menu item.




```
revealView.CanAddDateFilter = false;
```

Preselected Filters

You can specify which values are initially selected among existing Dashboard Filters when loading a dashboard.

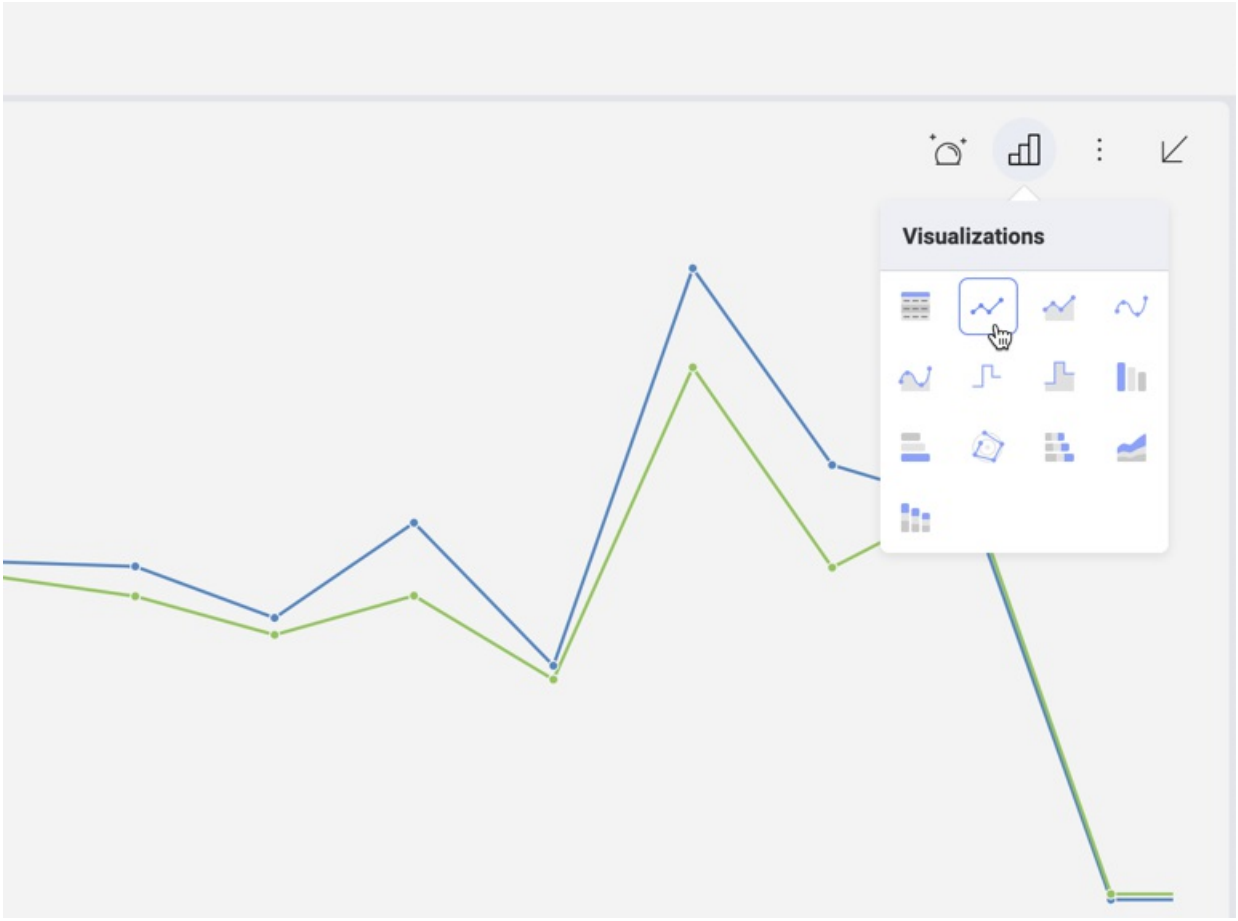


The following code snippet shows how to load a dashboard and set the “Territory” selected value to be “Americas”, thus the dashboard will be showing data filtered by “Americas”

```
var dashboard = new RVDashboard(path);
dashboard.filters.GetByTitle("Territory").selectedValues = new List<object>() { "Americas" };
revealView.Dashboard = dashboard;
```

AvailableChartTypes

This property can be used to filter the visualization types available to the user.



You can, for example, add or remove visualizations as shown below:

```
revealView.AvailableChartTypes.Add(RVChartType.BulletGraph);
revealView.AvailableChartTypes.Remove(RVChartType.Choropleth);
```

In addition, you can create a brand new List that includes only the visualizations you want to be available:

```
revealView.AvailableChartTypes = new List<RVChartType>() { RVChartType.BulletGraph, RVChartType.Choropleth };
```

Setting Up Initial Filter Selections

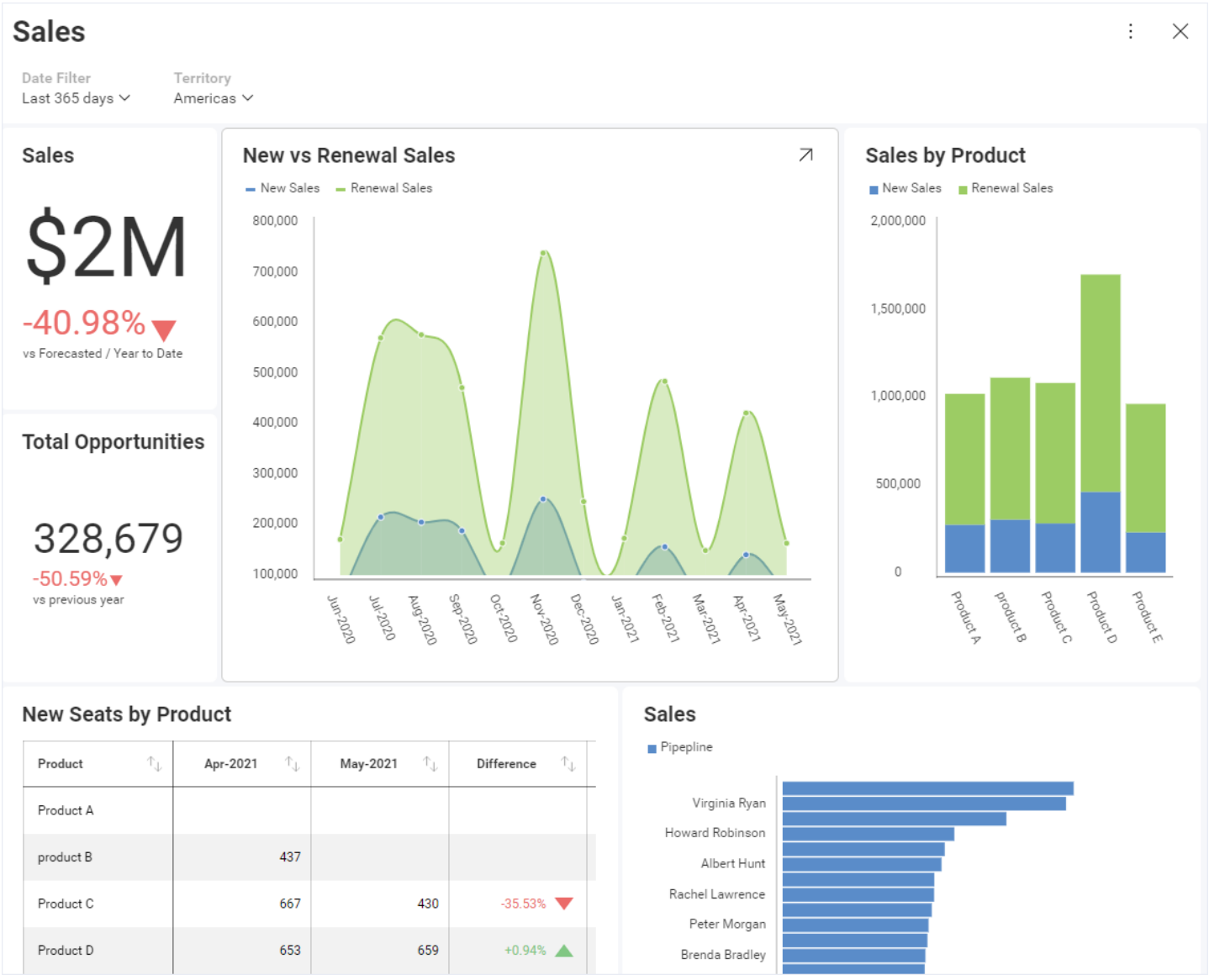
Overview

Sometimes, you want to display a dashboard with filters already applied. Dashboard filters are very useful to slice the contents of all the widgets at once. Because of this, you can use the SDK to set up initial dashboard filter selections that remain in context for all the dashboard's widgets.

Example Details

In this example, you have a dashboard showing Sales data with the following filters:

- A given period of time (last 365 days, Year to Date, etc.);
- Territory (Americas, Europe, Asia, etc.).



Code Example

In this case, you want to set the initial filters selection to:

- "Year to Date" (instead of "Last 365 days", the default setting for this dashboard);
- Sales associated to the Territory of the current user.

As part of the initialization process and once the dashboard is loaded, you can retrieve the list of filters in the dashboard and use these filters to set the initially selected values in **RevealView**:

```
var revealView = new RevealView();

using (var fileStream = File.OpenRead(path))
{
   var dashboard = new RVDashboard(fileStream);

   dashboard.DateFilter = new RVDateDashboardFilter(RVDateFilterType.YearToDate);
   dashboard.Filters.GetByTitle("Territory").SelectedValues = new List<object>() { CurrentUser.Territory };

   revealView.Dashboard = dashboard;
}
```

Note

The code above assumes that **CurrentUser.Territory** returns the name of the territory for the current user.

Hiding filters

It is possible that you might not want users to access data from territories different than their own. In that case, you can restrict the access to filters by configuring the **RevealView** object to hide the panel containing the dashboard filters:

```
revealView.ShowFilters = false;
```

That setting will restrict users to see data only for their associated territory.

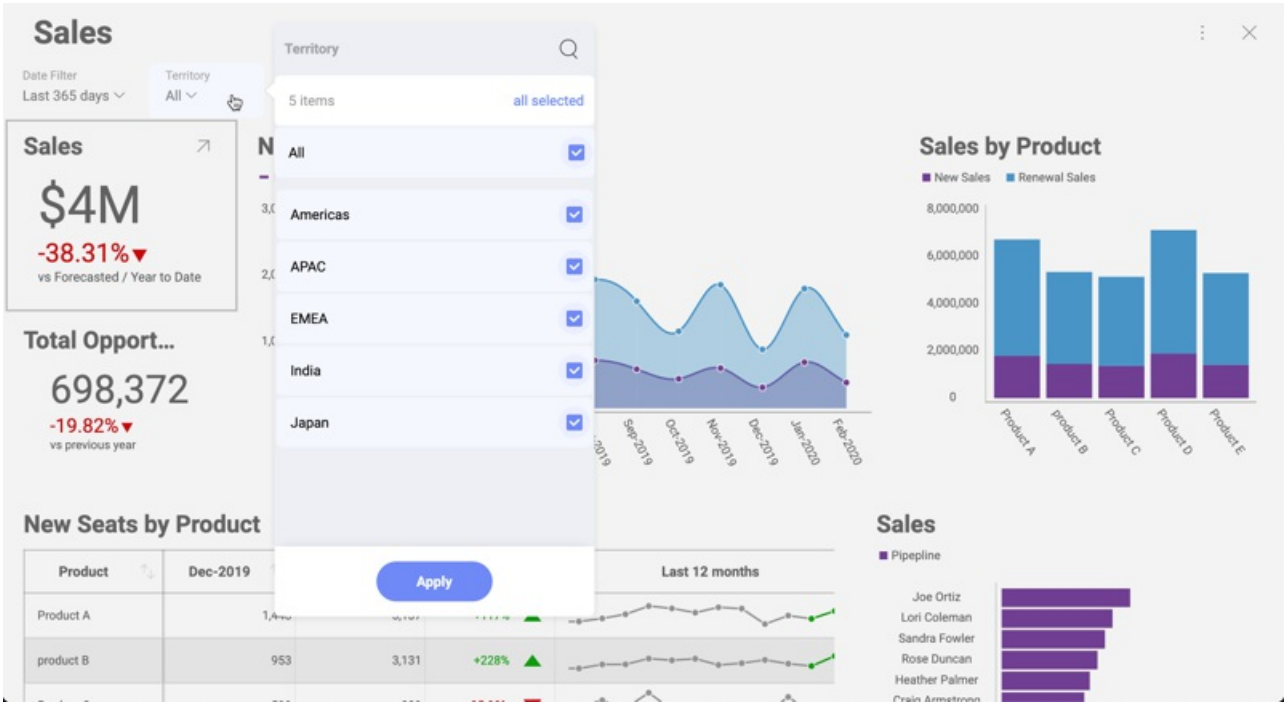
Finally, in the case that you still want users to change the date filter selection, take a look to **Setting up Dynamic Filter Selections**. There you'll find information about how to create your own UI that, allowing the user to change the date filter.

Setting up Dynamic Filter Selections

Overview

Sometimes your application will integrate a custom UI, to present the user with a list of values to select. And you might want that user selection to be synchronized with a filter in the dashboard.

For example, you can have a Sales dashboard that changes figures based on the current Territory and a custom UI to select the Territory. After the user selection changes, you'd want the Sales dashboard to reflect that change. Most of the times, you would hide the filter selection normally displayed in the dashboard. This way the user won't be confused with two different ways to change the Territory in the screen.



In the following code snippet, you'll find details about how to achieve the described scenario:

```
private void Americas_Click(object sender, RoutedEventArgs e)
{
   revealView.Dashboard.Filters.GetByTitle("Territory").SelectedValues = new List<object>() { "Americas" };
}
private void APAC_Click(object sender, RoutedEventArgs e)
{
   revealView.Dashboard.Filters.GetByTitle("Territory").SelectedValues = new List<object>() { "APAC" };
}
```

As shown above, two buttons were added to change the Territory selected between Americas and APAC. The code includes only the click handlers.

You can do the same when the selection is changed on your selector, just setting the new territory name using **SetFilterSelectedValues**.

Working with Dynamic Lists

Territories like Americas, APAC, India, etc. do not change over time, but other lists of values might change. In this case, if a new Territory is added to the list, a new button will not be automatically added.

You can use **RevealUtility.GetFilterValues** to get the list of values for a given filter. In this case the following call will leave an array with **RVFilterValue** objects in the *territories* variable, then loading the list of items in a ComboBox with all territories:

```
using (var stream = File.OpenRead(@"..\..\Sales.rdash"))
{
    var dashboard = new RVDashboard(stream);

    var filterValues = await dashboard.Filters.GetByTitle("Territory").GetFilterValuesAsync();
    var territories = filterValues.ToList();

    revealView.Dashboard = dashboard;

    foreach (var t in territories)
    {
        cmbTerritories.Items.Add(t);
    }
    cmbTerritories.SelectionChanged += CmbTerritories_SelectionChanged;

}
```

You can then use the *label* attribute from **RVFilterValue** to display the name of the territory and the *Value* attribute to set the selection in the filter.

The following code snippet shows how to handle the *selection changed* event for the ComboBox to update the filter in the dashboard:

```
private void CmbTerritories_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var selectedItems = new List<object>();
    var filterValue = cmbTerritories.SelectedItem as RVFilterValue;
    if (filterValue != null)
    {
        selectedItems.Add(filterValue.Value);
    }

    await dashboard.Filters.GetByTitle("Territory").SelectedItems = selectedItems;
}
```

Working with Date Filters

Date filters are a special kind of filter as they have no data associated. There's not a list of values to select from, instead you actually select a time interval (from date A to date B). You can select from a list of predefined filters like "Year To Date", "Previous Month", etc., or specify your own range (from 2019-Jan-12 to 2019-Jan-30).

Working with Predefined Filters

To set one of the predefined filters you can use a similar code to the following:

```
revealView.Dashboard.DateFilter = new RVDateDashboardFilter(RVDateFilterType.YearToDate);
```

If you want a list of all predefined date filters, please refer to **RVDateFilterType** in the API Reference.

Working with a Custom Range

To set a custom range, for example for the last 15 days, you can use a code similar to the following:

```
revealView.Dashboard.DateFilter =
    new RVDateDashboardFilter(
        RVDateFilterType.CustomRange,
         new RVDateRange(DateTime.UtcNow.AddDays(-15), DateTime.UtcNow)
));
```

There is a working example in the **Sales.xaml.cs** view, in the *UpMedia* WPF application distributed with the SDK. That sample view contains two custom filtering components: a date range selector and the territories displayed as a list of toggle buttons.

Desktop .NET API Reference

Here you will find technical information about Reveal SDK, specifically about the Desktop .NET API. For a complete reference, please follow the link

Most commonly used classes and interfaces

Main SDK concepts and features

RevealView class

RVDashboard class

RevealSdkSettings class

Datasources

IRVDataSourceProvider interface

RevealDataSources class

RVSqlServerDataSource

RVSqlServerDataSourceItem

RVRESTDataSource

RVRESTDataSourceItem

RVJsonDataSource

RVJsonDataSourceItem

IRVDataProvider

RVInMemoryData

RVInMemoryDataSource

RVInMemoryDataSourceItem

Authentication

IRVAuthenticationProvider

IRVDataSourceCredential

RVBearerTokenDataSourceCredential

RVUsernamePasswordDataSourceCredential

Localization

IRVLocalizationProvider

IRVLocalizationService

Formatting dashboards

IRVFormattingProvider

RVBaseFormattingService

Known Issues

- Web, Desktop
- Only Desktop

Web, Desktop

- When using the NuGet package, the watermark is still displayed after licensing the Reveal SDK (entering a valid key in the SDK installer).

  As a **workaround**, you can uninstall the NuGet package from the project, clear the NuGet's cache, and install the package again. In the case that you don't want to clear all NuGet's cache, you can lookup the location of that cache and clear only the Infragistics Reveal items. The location depends on the NuGet version and whether *packages.config* or *PackageReferece* is used.

Desktop

- After updating Reveal SDK to the new version in a project using package.config, uninstalling the old NuGet version or updating it to the new one will fail.

  As a **workaround**, prior to updating the Reveal SDK you can uninstall the NuGet package from your project. Then, update the Reveal SDK using the installer and finally you can reinstall the updated NuGet package.

  As an **alternative workaround**, prior to updating the Reveal SDK, you can back up the existing NuGet packages. To do that, go to *"%public%\Documents\Infragistics\Nuget"* (the location of the local NuGet package store created by the installer). Back up the existing packages to another folder, run the updated installer, and then copy the backed up packages back to the same location. Finally, you can now upgrade the NuGet version used in your project.

Release Notes

All future updates and new features added to Reveal SDK will be included here.

DATE	SDK VERSION	DESCRIPTION
Sep-2021	1.0.2013	*[Public Bug Fix] Calculated field export to excel resulting in empty cells* When exporting to excel a calculated field doing division by zero, the result included empty cells.
		[Public Bug Fix] Data blending with custom queries and server-side processing issues When turning on “Process Data On Server” in Web .NET and performing a custom query, data blending was not working as expected.
Sep-2021	1.0.2012	*[Public Bug Fix] [SDK] Decimal point hidden when Sparkline chart exceeds 100%* In the Web SDK, the property canSaveAs was not being honored if it was changed after a dashboard is set.
		[Public Bug Fix] [SDK] Small window sizes render Text chart unreadable In both Web and Desktop, the Text Chart font becomes unreadable when the window’s size is small.
		[Public Bug Fix] [SDK] Issues getting the list of date formats When getting a list of date formats for a field editor in the Desktop SDK, you can now use *RVBaseFormattingService* with aggregated dates.
Aug-2021	1.0.2008	*[Public Bug Fix] [SDK] Saving dashboard as a stream has issues* When saving dashboards as a stream, in some specific cases *dashboard.Serialize.Async()* was returning null.
June-2021	1.0.2005	*Scatter Maps now support OpenStreetMap!* You can now configure and use OpenStreet Map image tiles in Desktop (WPF) and Web-client (JS).
		New Thumbnail component! You can now render a thumbnail of a dashboard with *RevealDashboardThumbnailView*.
		Credentials from Web client to server-side data source A new type of credentials *RVHeadersDataSourceCredentials* allow you to send authentication headers and cookies to REST and Web Resource data sources. For further details, check the following sample in GitHub.
		SDK AspNetCore services injection You can now register the *RevealSdkContext* and *RevealUserContext* implementations as a type only (not passing an instance), allowing these classes to get any other AspNetCore services injected through the constructor. For further details check the following sample in GitHub.
		[Public Bug fix] Calculated field filter not working with data process on server When enabling server-side aggregation of the data, calculated fields used as filters were not filtering data as expected.
		[Public Bug fix] Google Analytics issues with dashboard filters When getting data from Google Analytics data sources, you were unable to create dashboard filters.
	1.0.7	*Scatter Maps now support OpenStreetMap!* You can now configure and use OpenStreet Map image tiles in the SDK Web-client (JS).
		[Public SDK Bug fix] Text Box content not visible after component is remounted When using React with a dashboard and a Text Box visualization, content was not visible after component remount. A page reload was required.

DATE	SDK VERSION	DESCRIPTION
June-2021	1.0.7 JAVA	
		[Public Bug fix] Calculated field filter not working with data process on server When enabling server-side aggregation of the data, calculated fields used as filters were not filtering data as expected.
		[Public Bug fix] Google Analytics issues with dashboard filters When getting data from Google Analytics data sources, you were unable to create dashboard filters.
June-2021	1.0.6 JAVA	*[Bug Fix] [SDK] Grizzly server throws an exception* When running Reveal in Grizzly, a *NoClassDefFoundError* exception was being thrown because of a wrong dependency in *javax.servlet.ServletContext* class (javax.servlet:javaz.servlet-api assembly).
		New sample for JAVA SDK released! There is a new GitHub sample showing how to use Reveal with Grizzly server.
June-2021	1.0.5 JAVA	*Credentials from Web client to server-side data source* A new type of credentials *RVHeadersDataSourceCredentials* allow you to send authentication headers and cookies to REST and Web Resource data sources. For further details, check the following sample in GitHub.
May-2021	1.0.1956 (1.0.4 JAVA)	*[Public Bug Fix] [SDK] Full list of Data Sources displayed by mistake* When using *DataSourcesRequested* callback in the Desktop SDK, the whole list of data sources was being displayed instead of the ones explicitly added.
		[Public Bug Fix] [SDK] Desktop SDK export to Excel not working as expected When reloading a dashboard and then exporting a single visualization to Excel, the first visualization of the dashboard was always the one exported.
		[Public Bug Fix] [SDK] Dashboard with SQL data source using a dynamic port not loading When loading a dashboard with an SQL data source defined using a dynamic port (providing an instance in the host field), the data source connection was not working because of issues with the dynamic port configuration.
		[Public Bug Fix] Calculated field set as Visualization filter were throwing an error When configuring a Visualization filter based on a calculated field that depends on another calculated field, an error was being shown ("Invalid column name").
		[Public Bug Fix] Drill down scenario with different "sorty by" configurations not working as expected When the fields in a hierarchy were configured with a combination of "sort by: " and a descending sorting, the result was the dashboard not loading.
		Credentials from Web client to server-side in cross-domain applications When the backend is not in the same domain as the frontend and you need authentication cookies, you can request credentials using the following Web SDK setting: **$.ig.RevealSdkSettings.requestWithCredentialsFlag = true;**
May-2021	1.0.3 JAVA	*New Snowflake connector!* Reveal Java SDK now supports Snowflake data source connector, also including data blending between tables in the same Snowflake database.
		Reveal BI Engine for Java was enhanced Java platform is now as robust as other platforms, helping to avoid server crashes when a visualization sends a big amount of data back to the client. Several new properties in **InitializeParameterBuilder** control this: *maxInMemoryCells*, *maxStorageCells*, *maxStringCellSize*, and *maxTotalStringSize*.
	1.0.0	*New JAVA SDK released!* Reveal now supports JAVA as another Web Server option besides .NET. The JAVA SDK requires JAVA 11+ and is distributed as a set of Maven modules. For further details, please refer to Setup and Configuration.

DATE	JAVA SDK VERSION	DESCRIPTION
Apr-2021	1.0.0	
		JAVA SDK Samples released! You can get the JAVA SDK UpMedia samples available in Github.
Mar-2021	1.0.1866	*New Properties for Web and Desktop SDK:* *showEditDataSource* - can be used to disable the Edit button normally available in the data source overflow menu. *canAddDashboardFilter* - this property can hide the "Add Dashboard Filter" option in the Add Filter menu. These options are available in Dashboard Edit Mode. *canAddDateFilter* - this property can hide the "Add Date Filter" option in the Add Filter menu. These options are available in Dashboard Edit Mode.
		[Public Bug Fix] [SDK] revealView.canSaveAs property not working as expected In the Web SDK, the property canSaveAs was not being honored if it was changed after a dashboard is set.
		[Public Bug Fix] [SDK] HttpContextAccessor.HttpContext property not working as expected In the Web SDK, HttpContextAccessor.HttpContext was null when saving a dashboard (accessing it from SaveDashboardAsync method).
Mar-2021	1.0.1821	*[Public Bug Fix] [SDK] SDK apps sometimes throw an NRE exception* When an SDK application was opened for more than 90 minutes without users interacting with it, performing an action was throwing an exception.
Feb-2021	1.0.1772	*[Bug Fix] [SDK] Installation of WPF NuGet package failing with packages.config* The installation of WPF NuGet package was failing when the host project used packages.config.
Feb-2021	1.0.1763	*[Public Bug Fix] [SDK] HasPendingChanges property not working as expected* In Desktop SDK, the HasPendingChanges property was not set to false after saving a dashboard with changes.
		[Public Bug Fix] [SDK] Custom filtering not working In Desktop SDK, custom queries were not filtering data as expected.
		[Public Bug Fix] [SDK] Hiding SQLServer tables also hides views When using RVDataSourceItemsFilter to hide all tables and show only views, the Views tab was also hidden.
		[Public Bug Fix] [SDK] AzureSQL Data Provider throwing an error When adding an AzureSQL connection, an error message was displayed.
		[Public Bug Fix] [SDK] Date filters not displayed if LocalizationProvider set When a LocalizationProvider was set, date filters from/to were not displayed in the visualizations editor.
		[Public Bug Fix] Word not localized to Japanese The word "Others" was not localized to "その他" in Japanese.
Jan-2021	1.0.1712	*[Public Bug Fix] [SDK] The server component relies on Newtonsoft.Json serializer* The Reveal server component was relying on the default JSON serialization settings of the MVC application. Now the hosting app can configure JSON serialization settings as needed.
		[Public Bug Fix] [SDK] SQL Server filtering not working for NVARCHAR columns Filtering for Microsoft SQL Server was not working for NVARCHAR columns when the filtered value contained multibyte characters.
		[Public Bug Fix] [SDK] Pivot hierarchies filtering not working with "Processing Data on Server" If the option "Processing Data on Server" was checked, drill down hierarchies in the Pivot Editor were not filtering data.

DATE	SDK VERSION	DESCRIPTION
Dec-2020	1.0.1669	*[Public Bug Fix] [SDK] Custom filtering not working with "Processing Data on Server* If the option "Processing Data on Server" was checked, custom queries were not returning the correct number of rows.
Dec-2020	1.0.1629	*Save/Load Dashboards using JSON* You can now use Reveal SDK to save/load dashboards to/from JSON files.
		[Public Bug Fix] Category field label not being shown In Category Charts, tooltips were not displaying the field label but the original field name of a category instead.
		[Public Bug Fix] Dates in drill down breadcrumbs wrongly displayed When drilling down on a date field, breadcrumbs did not display values properly. Now breadcrumbs give clear information about your drill down level.
		[Public Bug Fix] Hover Tooltips and Crosshairs not shown by default In Dashboard View mode, Hover Tooltips and Crosshairs were not displayed until users enable them. Now they are enabled by default.
Sep-2020	1.0.1422	*Amazon Athena connector in BETA* You can now connect to Amazon's serverless, interactive query service Athena.
		NEW Pre-built Themes We added four pre-built app themes. Set one of them and use the customizable settings to additionaly personalize the look and feel of the Visualization and Dashboard editor. You can choose from one of the following themes: MountainLightTheme (Desktop) / $.ig.MountainLightTheme (Web); MountainDarkTheme (Desktop) / $.ig.MountainDarkTheme (Web); OceanLightTheme (Desktop) / $.ig.OceanLightTheme (Web); OceanDarkTheme (Desktop) / $.ig.OceanDarkTheme (Web).
		Marketo provider is Now Available You can now connect to the marketing platform Marketo and use your data in Reveal.
		Amazon Redshift provider is Now Available You can now use and gain new insights from your data in the Amazon Redshift cloud data warehouse.
		New "Data Process on Server" feature You can now have server-side aggregation of the data coming from the MS SQL, MySQL and Postgres data sources
Jul-2020	1.0.1374	*New API to set axis bounds for charts* You can now programmatically change the axis bounds in runtime for a particular visualization.
		Salesforce data source enhancements Now you can use your Salesforce reports in Reveal.
		New QuickBooks data source Connect to your Quickbooks account and use your entities to perform data analysis in Reveal.
		New Hubspot data source You can now connect to Hubspot.
		Sharepoint lists and document libraries support You can now use the metadata (name, type, etc.) collected for all files in a SharePoint library as a data source in Reveal.

DATE	SDK VERSION	DESCRIPTION
		New Choropleth Map Visualization The Choropleth map visualization allows you to create beautiful thematic maps. You can now present geospatial data in an incredibly digestible manner. Let color guide you and help you quickly discover patterns, trends and anomalies on the map.
May-2020	1.0.1255	*New Azure Analysis Services data source* With this new data source, you can create dashboards using your data models in Azure Analysis Services.
		New icon for Google Sheets files The look of the Google Sheets files icon was changed.
May-2020	1.0.1222	*New Hover Events API* This new event is called *revealView.TooltipShowing* in WPF and .onTooltipShowing in Web and is triggered whenever the end-user hovers over a series in a visualization or clicks on the series.
		New TreeMap visualization You can use this new visualization type to present large hierarchies with a set of nested rectangles. Rectangles' size will show you part-to-whole relationships amongst a variety of metrics, helping you identify patterns and relations between similar data.
		Export to Excel enhancements You can include more visualization types in your spreadsheets upon export. Scatter, Bubble and Sparkline charts are now available.
		Various UI/UX improvements Various minor changes were added to improve user experience in the Visualization, Dashboard, New Data Source dialog, etc.
		Added support for Shared Drives in Google Drive If you have a GSuite Business account, you can now access your Shared Drives data and use it to build visualizations in Reveal.
April-2020	1.0.1136	*New Custom Theming* Now you can create your own theme in Reveal by configuring some or all of the customizable settings in the new RevealTheme (Desktop) / $.ig.RevealTheme (Web) class.
Feb-2020	1.0.981	*New Properties in RevealSettings* We added multiple new properties to $.ig.RevealSettings to control different features, including: ShowExportToPDF, ShowExportToPowerpoint, ShowExportToExcel, ShowStatisticalFunctions, ShowDataBlending, ShowMachineLearningModelsIntegration, StartWithNewVisualization, InitialThemeName.
		Accent Color is Now Available You can now find the SetAccentColor method added to $.ig.RevealView.
		A Trigger Property Added to DataSourceRequested Event We added a Trigger (of type DataSourcesRequestedTriggerType) property to the DataSourcesRequested event arguments. The users of this event will now gain more context about the DataSourcesRequested purposes.
Nov-2019	1.0.825	*Export to Image Functionality is Now Working* Exporting images server-side (both programmatically and through user interaction) was enabled again. For further details about the fix, please refer to: Enabling server-side screenshot generation
		Localization Service for Reveal Desktop SDK You can now localize titles and labels of a variety of dashboard elements. The Localization service also enables you to change the formatting settings of numeric and non-aggregated date fields.

DATE	SDK VERSION	DESCRIPTION
Sep-2019	1.0.80x	*Formatting Service for Reveal Desktop SDK* You can now format numeric data, aggregated and non-aggregated date fields to your own preferences. Ignore the default formatting and format your dashboard data the way you like it.
		Changes in Setup and Configuration (Server SDK) Reveal Server SDK now supports .NET Core 2.2+ as well as .NET Framework 4.6.1+ ASP MVC application projects. In addition, you will now use exclusively the NuGet package manager to reference assemblies and install dependency packages.
Sep-2019	1.0.70x	*Step by Step Guide* With this detailed guide, you will start with prerequisites and go through every step needed to setup and configure Reveal's SDK.
		Change the Widget's Data Source You can now enable or disable the possibility to change a widget's data source to end users. When opening the Visualization Data screen in edit mode, Reveal will either show or hide the change data source button in the UI.
		Formatting Service for Reveal Desktop SDK You can now enable or disable the possibility to change the dashboard's theme to end users. When entering edit mode for a dashboard, Reveal will either show or hide the button used to display the available themes.

Third-Party Software Used by Reveal SDK

PLATFORM	CODE USED	LICENSE
Web	Emoji One Area	MIT
Web	textcomplete	MIT
Web	Google Analytics	Custom
Web	Bootstrap	MIT
Web	JS Foundation Project	MIT
Web	JQuery validation	MIT
Web	JQuery validation unobtrusive	Apache 2.0
Web	Roboto Google Font	Apache 2.0
Web	Web Font Loader	Apache 2.0
Web	Google Maps Marker Clusterer	Apache 2.0
Web	Google Maps Platform	Custom
Web	SignalIR	Apache 2.0
Web	Babel JS Compiler	MIT
Web	LitElement	BSD 3-Clause
Web	Simple JavaScript Inheritance	MIT
WPF	ANTLR	BSD 3-Clause
WPF	Json.NET	MIT
WPF	SQLite	Public
WPF	Oracle.ManagedDataAccess	OTN
WPF	Microsoft .NET Library	Microsoft Software License
WPF	Npgsql	PostgreSQL License
WPF	Microsoft.AnalysisServices.AdomdClient	Custom download
WPF	Sybase.AdoNet4.AseClient	Apache 2.0

WPF	CefSharp	CefSharp
WPF	Cef	Cef
WPF	SkiaSharp	MIT
WPF	CSharpAnalytics	Apache 2.0
WPF	NLog	BSD 3-Clause
WPF	SuperSocket	Apache 2.0
WPF	WebSocket4Net	Apache 2.0
WPF	WebSocket4Net	Apache 2.0
WPF	Windows Credentials Management Library	Apache 2.0
WPF	Fody	MIT
WPF	LumenWorks.Framework.IO	MIT
WPF	CoreFX	MIT
WPF	Strongnamer	MIT
WPF	MySql Connector	MIT