ReportPlus Embedded Desktop SDK Guide

Disclaimer

ReportPlus™ Desktop – Embedding Guide 1.0.4

Table of Contents

Introduction

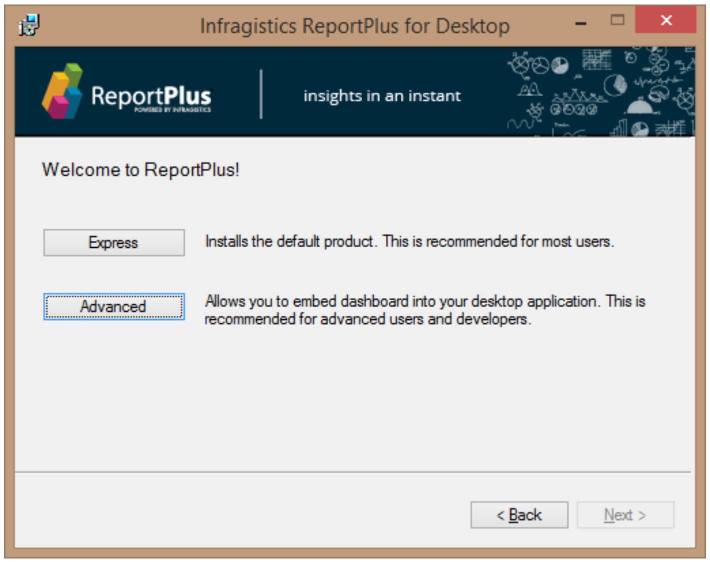
Welcome to the ReportPlus Desktop SDK Embedding Guide.

The main goal of this document is to allow ReportPlus dashboards (.rplus files) to be displayed and embedded into 3rd party applications. The document will provide the means to view existing dashboards within a 3rd party application.

Chapter 1
SDK Installation

SDK Installation

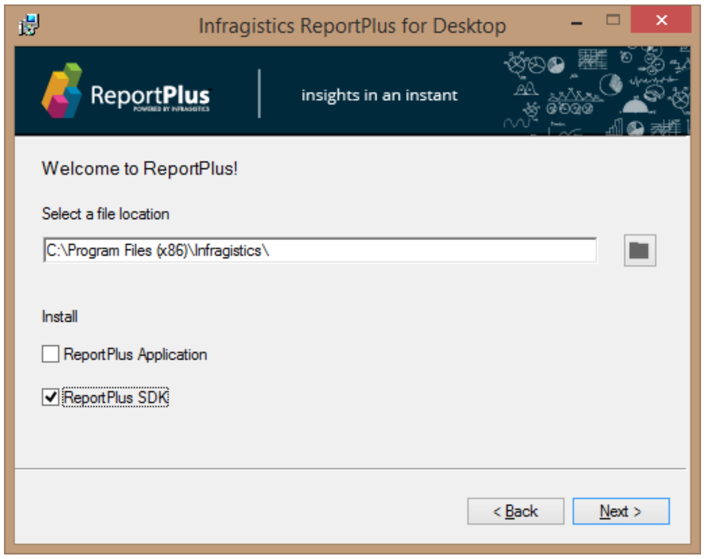
Currently, the ReportPlus Desktop SDK is provided as an optional feature included in the ReportPlus Desktop application installer when **Advanced** is selected.



Welcome to ReportPlus!

Express — Installs the default product. This is recommended for most users.

Advanced — Allows you to embed dashboard into your desktop application. This is recommended for advanced users and developers.



Welcome to ReportPlus!

Select a file location

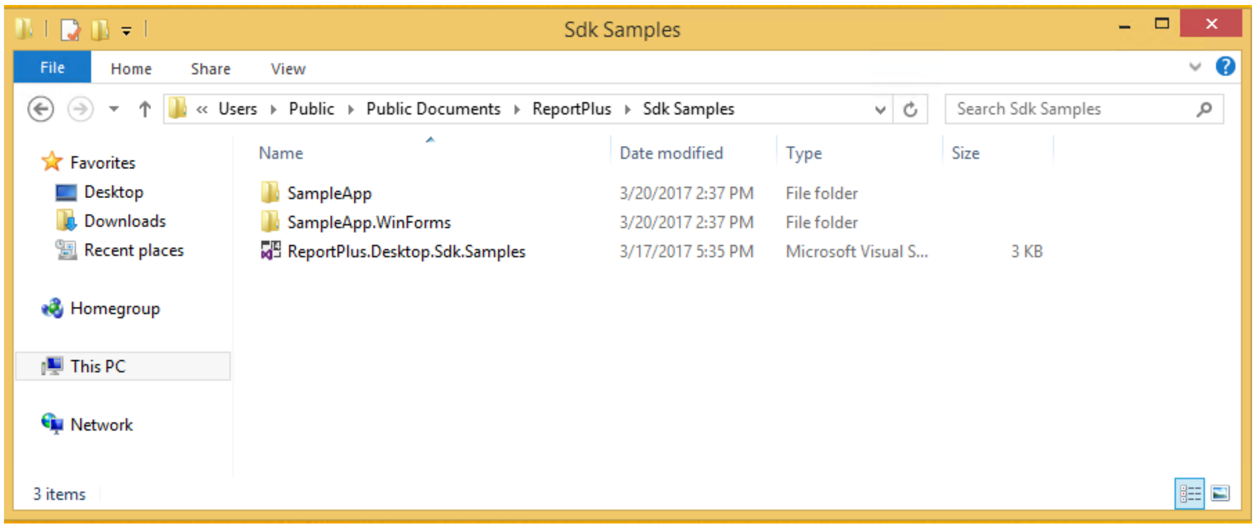
C:\Program Files (x86)\Infragistics\

Install

☐ ReportPlus Application

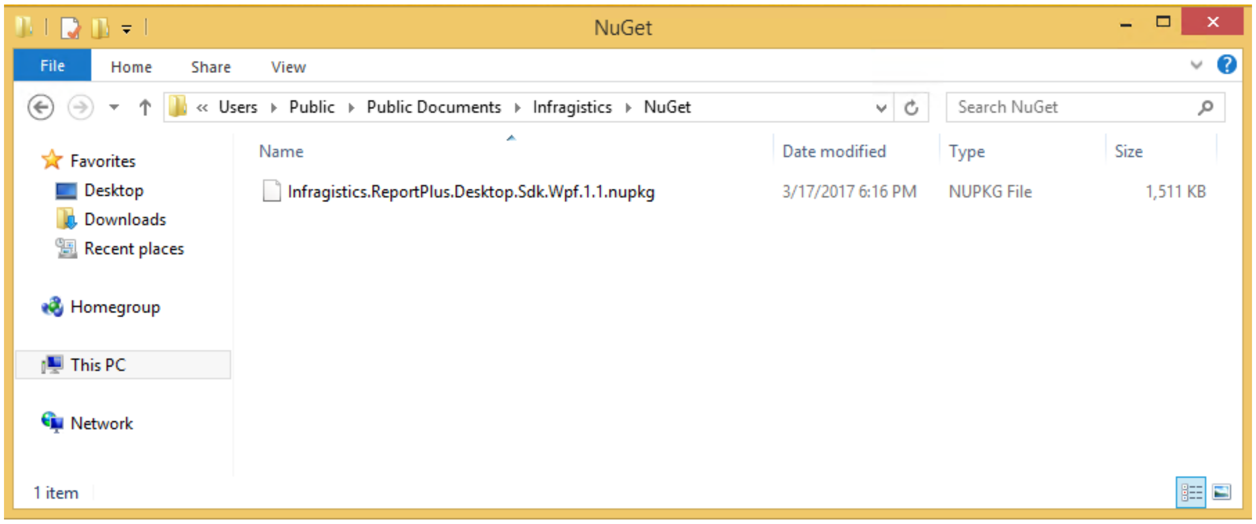
☑ ReportPlus SDK

Once the installation is completed, a sample application project can be found under the **%SystemDrive%\Users\Public\Documents\ReportPlus\Sdk Samples** folder.



Name	Date modified	Type	Size
SampleApp	3/20/2017 2:37 PM	File folder	
SampleApp.WinForms	3/20/2017 2:37 PM	File folder	
ReportPlus.Desktop.Sdk.Samples	3/17/2017 5:35 PM	Microsoft Visual S...	3 KB

The ReportPlus SDK is now distributed as a local *NuGet* package located in the **%SystemDrive%\Users\Public\Documents\Infragistics\NuGet** folder.



Name	Date modified	Type	Size
Infragistics.ReportPlus.Desktop.Sdk.Wpf.1.1.nupkg	3/17/2017 6:16 PM	NUPKG File	1,511 KB

Design Time Experience

The ReportPlusViewer component is no longer available through the WPF Toolbox. For more information on adding the ReportPlusViewer control and ReportPlus.SDK libraries to your project, please refer to the Integrating ReportPlus to your Project section.

Prerequisites

.NET Framework Version

IMPORTANT: The ReportPlus.Desktop.Sdk are built against Microsoft .NET Framework 4.6 and as such the minimum required application target framework is Microsoft .NET Framework 4.6.

Getting Started

The quick start helps you integrate ReportPlus components into your application with a step-by-step explanation. To work with ReportPlus and its components, follow these steps:

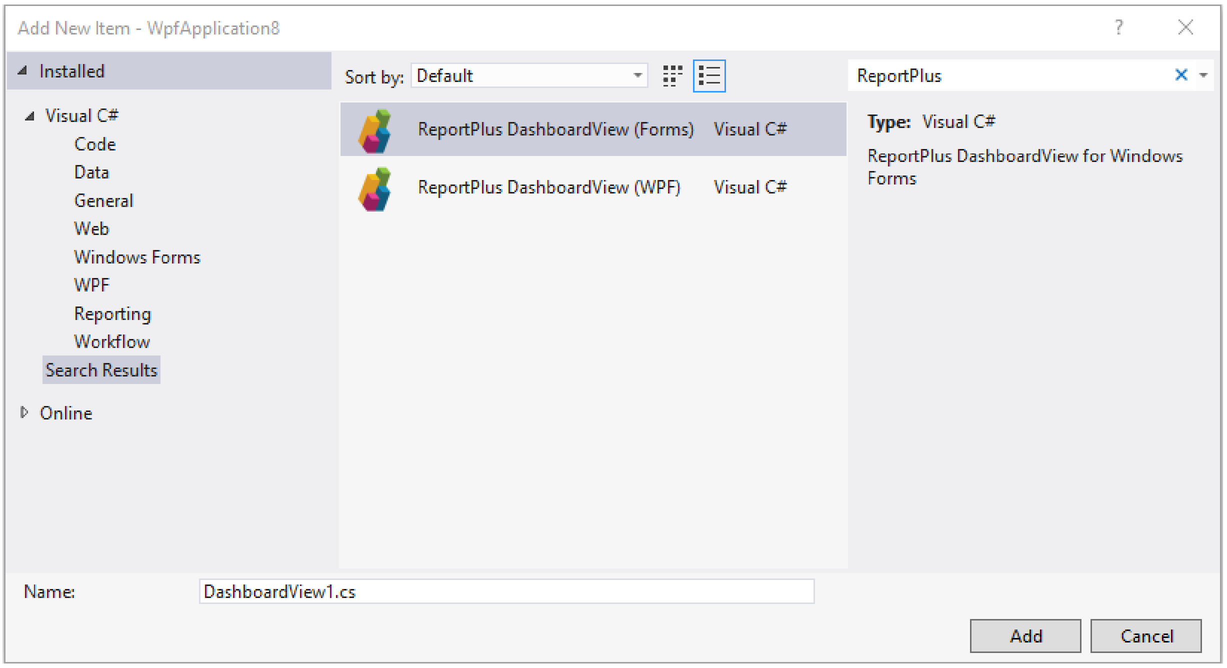
1. **Open an existing Windows Forms or WPF project in Visual Studio**.

Note: Your project must target .NET Framework 4.6 or higher for the item templates to be present in the Add New Item dialog.

2. **Right click the project** you want to add a DashboardView to.

3. Choose **Add** → **New Item**.

4. In the opened dialog**, you will find the item templates** under **Visual C#** → **WPF** and **VisualC#** → **Windows Forms**. You can also choose to use the search at the top right.



ReportPlus DashboardView (Forms) Visual C#
ReportPlus DashboardView (WPF) Visual C#

Type: Visual C#
ReportPlus DashboardView for Windows Forms

Name: DashboardView1.cs

5. **Add the newly added DashBoardView** (from step 4) **to a view or window in your application**:

```
<local:DashboardView1 x:Name="DashboardView1"></local:DashboardView1>
```

6. **Add a Dashboard property to a ViewModel** to use with the new view:

```
public class vm
{
    public Dashboard Dashboard { get; set; }
}
```

7. In the loaded event of the view or window, **add the following code**:

```
OpenFileDialog dialog = new OpenFileDialog();
dialog.ShowDialog(this);
var filestream = dialog.OpenFile();
var dashboard = Dashboard.Load(filestream);
DashboardView1.DataContext = new vm() { Dashboard = dashboard };
```

Note: if you need to load a dashboard with authentication, see the Working with OAuth Providers topic.

8. **Run** the application.

9. **Select a ReportPlus Dashboard** form the installed samples (C:\Users\Public\Documents\ReportPlus\My Dashboards\Sample Dashboards). These samples are installed only with the ReportPlus application.

Chapter 2
Configuration Procedures

Integrating ReportPlus to your Project

This section will cover how-to topics related to integrating ReportPlus components into your project.

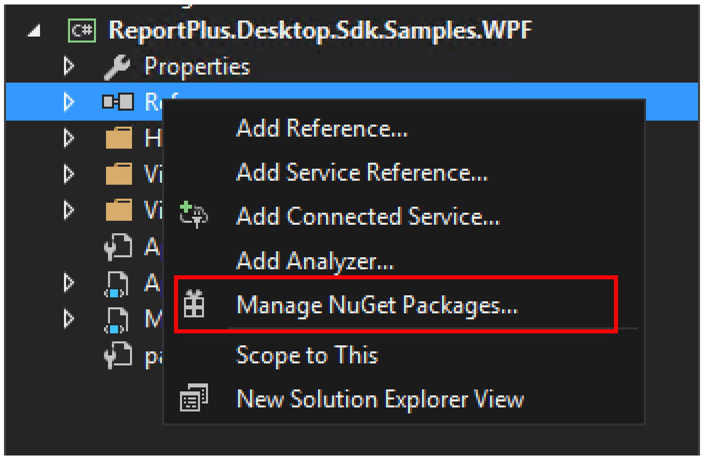
- Adding ReportPlus.SDK libraries using a NuGet package
- Adding the ReportPlusViewer control to your application

How to add ReportPlus.SDK libraries using a NuGet package

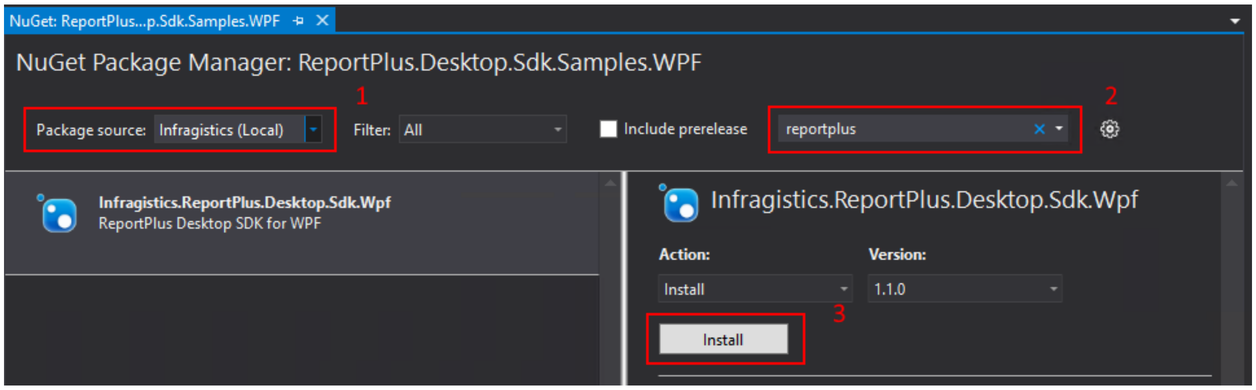
In order to use the ReportPlusViewer control in your application, you will need to install the SDK libraries first. These libraries are also available through the **Infragistics.ReportPlus.Desktop.Sdk.Wpf** NuGet package distributed as part of SDK installation. This package can be found under the **Infragistics (Local)** package source in Visual Studio.

In order to add that NuGet package to your WPF solution:

1. Right click on the project's **References** node and then click **Manage NuGet Packages…**



2. In NuGet Package Manager window
   i. Select **Infragistics (Local)** as a package source.
   ii. Filter the packages by typing "reportplus" in the search box.
   iii. Select **Infragistics.ReportPlus.Desktop.Sdk.Wpf** NuGet package and then click the Install button.


NuGet: ReportPlus...p.Sdk.Samples.WPF

NuGet Package Manager: ReportPlus.Desktop.Sdk.Samples.WPF

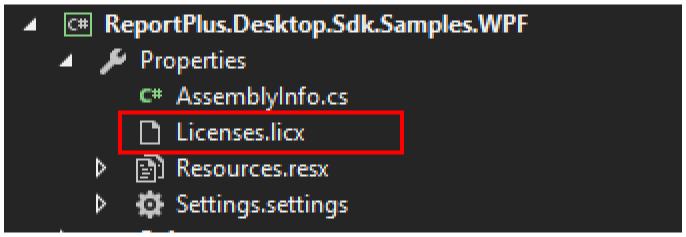
Package source: Infragistics (Local)
Filter: All
Include prerelease
reportplus

Infragistics.ReportPlus.Desktop.Sdk.Wpf
ReportPlus Desktop SDK for WPF

Infragistics.ReportPlus.Desktop.Sdk.Wpf
Action: Install
Version: 1.1.0

Please note that:

1) The NuGet package manager will also install other NuGet packages the **Infragistics.ReportPlus.Desktop.Sdk.Wpf** package depends on.
2) The NuGet package manager will also add the Licenses.licx file to your project.



That file provides the licensing information concerning the ReportPlusViewer control and is need at compile time to verify if ReportPlus.SDK product is licensed or not.

```
Infragistics.ReportPlus.Desktop.Controls.ReportPlusViewer,
Infragistics.ReportPlus.Desktop.Sdk, Version=1.1.327, Culture=neutral,
PublicKeyToken=ebbf8110063acd1a
```

How to add the ReportPlusViewer control to your application

Once theReportPlus.SDK libraries are added to your project, you can add the ReportPlusViewer control to your user control. Please refer to the Integrating ReportPlus to your Project section for more information about how ReportPlus.SDK libraries can be added to your project.

```
<UserControl x:Class="ReportPlus.Desktop.Sdk.Samples.Views.DashboardView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:rplus="http://schemas.infragistics.com/reportplus/ReportPlusDesktop" >
    <Grid>
        <rplus:ReportPlusViewer Dashboard="{Binding Dashboard}" />
    </Grid>
</UserControl>
```

Working with your Dashboards

This section will cover how-to topics related to working with your dashoards using the components integrated to your project.

- Loading a Dashboard
- Providing a local data files location
- Loading a dashboard with a maximized widget

How to Load a Dashboard

In order to open dashboard (**.rplus**) files and visualize them in the ReportPlusViewer component, the SDK API exposes the `Dashboard.Load` method, which has three overloads to cover different security scenarios.

Note that all three methods consume a stream of that .rplus file. A stream object can be obtained using different methods, i.e. using file dialog or directly opening it through calling File.Open.

1. **Opening a dashboard file with no authentication**. In this scenario, it is assumed that opened dashboard has widgets connected only to data sources where authentication is not needed.

```
var fileStream = dialog.OpenFile();
var dashboard = Dashboard.Load(fileStream, settings);
this.Dashboard = dashboard;
```

2. **Opening a dashboard file with a callback to provide the security authentication information for each data source**. In this scenario, the callback will be invoked for each data source where authentication is needed:

```
var fileStream = dialog.OpenFile();
var dashboard = Dashboard.Load(fileStream, ProvideSecurityInfoCallback, settings);
this.Dashboard = dashboard;
```

A sample callback method is shown below:

```
private ISecurityInfo ProvideSecurityInfoCallback(DataSource dataSource)
{
    switch (dataSource.UniqueId)
    {
        case "[specific id value]":
        // provide appropriate security information

        default:
            return null;
    }
}
```

3. **Opening a dashboard with a batch callback which allows the security information to be provided in an interactive manner by the end user.**

A sample callback used in such scenario is shown below:

```
private DataSourceSecurityInfo[] ProvideBatchSecurityInfoCallback(DataSource[] dataSources)
{
    var securityInfoData = new DataSourceSecurityInfo[dataSources.Length];
    for (int i = 0; i < dataSources.Length; i++)
    {
        var dataSource = dataSources[i];
        switch (dataSource.UniqueId)
        {
            case "[specific id value]":
                // provide appropriate security information
                securityInfo[i] = new DataSourceSecurityInfo(dataSource, new Credentials());
                break;

            default:
                break;
        }
    }

    return securityInfoData;
}
```

How to Provide a Local Data Files Location

The ***settings*** object passed to the `Dashboard.Load` method is optional in all scenarios. As for now, this object exposes only a single property to specify the location of the data files used by the dashboard. This parameter is required only for dashboards with widgets connected to local data files. Since the location for these files is associated only with the dashboard being loaded, there might be multiple ReportPlusViewer components in the same application with data stored in different locations.

```
var docs = System.Environment.GetFolderPath(System.Environment.SpecialFolder.MyDocuments);
var settings = new DashboardSettings
{
    DataFilesLocation =  docs + @"\ReportPlus\Dashboards Data",
};
```

How to Load a Dashboard with a Maximized Widget

If needed, any of the widgets in a dashboard can be shown maximized by default.

```
var fileStream = dialog.OpenFile();
var dashboard = Dashboard.Load(fileStream, ProvideBatchSecurityInfoCallback, settings);
dashboard.MaximizedWidget = dashboard.Widgets.First();
this.Dashboard = dashboard;
```

Advanced ReportPlus Configuration

The topics covered in this section are:

- Working with OAuth Providers
- Initializing ReportPlus Engine Services
- Working with in-memory data (CTP)

How to Work with OAuth Providers

The ReportPlusViewer component supports connections too and can use data stored in repositories using OAuth authentication. The client application, however, should provide an authentication info object with valid AccessToken and RefreshToken values.

In order to help developers easily connect to OAuth storages. there are some helper classes distributed with the sample application. Those classes, found under project's **Helpers** folder, can be used in the process of authentication object retrieval.

Here is an excerpt from the samples' code that demonstrates the usage of some of the helper classes:

```
var oAuthSecurityProvider = OAuthSecurityProviderResolver.Resolve(dataSource.Provider);
    if (oAuthSecurityProvider != null)
    {
        var oAuthWindow = new OAuthWindow(oAuthSecurityProvider);
        oAuthWindow.ShowDialog();

        return oAuthSecurityProvider.TokenResponse;
    }
```

The client code should also provide the ClientId and ClientSecret of the hosting application. This will happen while the code shown above fallbacks to the values shown below, and will depend on which provider is being used:

```
    class OAuthClients
    {
        public static class GoogleDrive
        {
            public const string ClientId = "";
            public const string ClientSecret = "";
        }

        public static class DropBox
        {
            public const string ClientId = "";
            public const string ClientSecret = "";
        }
    }
```

How to Initialize the ReportPlus engine services

The ReportPlusViewer component depends on different services; for example, in-memory data provider, or different formatting services. When needed, those services can be overridden by placing the code shown below in your App.xaml.cs file:

```
static App()
{
    ReportPlusBootstrapper.Initialize(GetApplicationEngineSetup());
}
private static Discoverables.EngineSetup GetApplicationEngineSetup()
 {
   return new Discoverables.EngineSetup().
   Service<ICustomDataSourceResolver>(r => new DataSourceResolver()).
   Service<IInMemoryDataProvider>(r => new InMemoryDataProvider());
 }
```

How to Work with in-memory data (CTP)

The ReportPlusViewer can also work with in-memory data. However, that requires two additional services to be registered at the application's startup:

```
        static App()
        {
            ReportPlusBootstrapper.Initialize(GetApplicationEngineSetup());
        }

        private static Discoverables.EngineSetup GetApplicationEngineSetup()
        {
            return new Discoverables.EngineSetup().
            Service<ICustomDataSourceResolver>(r => new DataSourceResolver()).
            Service<IInMemoryDataProvider>(r => new InMemoryDataProvider());
        }
```

The process of working with in-memory data goes through two stages:

1. Creating in-memory data source replacement of the original data source.
2. Providing a schema and data for each of your in-memory data sources.

Below are shown sample implementations of these two services – based on context information could be created a new in-memory data source instance. Each in-memory data source instance is assigned an identifier provided by the customer - i.e. `"SalaryData2016"`. Later the same identifier is provided as a parameter when the schema and the data for that in-memory data source are requested.

```
    class DataSourceResolver : ICustomDataSourceResolver
     {
         public DataSource Resolve(DataSource modelDataSource, DataSourceContext context)
         {
             if (context.DashboardTitle == "Salary Data")
             {
                 return new InMemoryDataSource("SalaryData2016");
             }

             // reuse the original model data source
             return modelDataSource; // or return null;
         }
     }

     class InMemoryDataProvider : IInMemoryDataProvider
     {
         public IEnumerable<IList<object>> GetData(string dataSourceId)
         {
             if (dataSourceId == "SalaryData2016")
             {
                 return new List<IList<object>>
                 {
                     new object[] { "Name1", "Family1", 120000},
                     new object[] { "Name2", "Family2", 120500},
                 };
             }

             return System.Linq.Enumerable.Empty< IList<object>>();
         }

         public IList<ISchemaColumn> GetSchema(string dataSourceId)
         {
             if (dataSourceId == "SalaryData2016")
```

```
            {
                return new List<ISchemaColumn>()
                {
                    new SchemaColumn("first_name", DashboardDataType.String),
                    new SchemaColumn("last_name", DashboardDataType.String),
                    new SchemaColumn("salary", DashboardDataType.Number),
                };
            }

            return new List<ISchemaColumn>();
        }
    }
```

IMPORTANT

The order in the data cells in the data rows must match the order of the columns defined in the schema.

In addition, the schema should match the schema of data used by the original widget.

Appendices

Appendix 1: Document Changelog

Appendix 1: Document Changelog

Version	Chapter	Section	Description
1.0.4	SDK Installation	SDK Installation	The ReportPlus SDK is now distributed as a local NuGet package. All information updated accordingly.
		Design Time Experience	Changes to the ReportPlusViewer component updated.
		Prerequisites	Added prerequisite to the SDK installation process.
	Configuration Procedures	-	Added new procedures and re-organized information for easier access.
1.0.3	SDK Installation	SDK Installation	The installation of samples is now available within the Public Documents folder.
1.0.2	Dashboard Procedures	How to work with in-memory data (CTP)	Added new section with instructions on how to work with in-memory data (CTP).
1.0.1	Limitations & Known Issues	ReportPlus .NET Requirements	Added new section with details on the x86 Target Platform.
	Dashboard Procedures	How to Configure the Location for the ReportPlusViewer modules	Added new section with instructions on how to configure the location for the ReportPlusViewer modules.
		How to Initialize the ReportPlusViewer component	Added new section with instructions on how to initialize the ReportPlusViewer component.
		How to Work with OAuth Providers	Added new section with instructions on how to work with OAuth Providers.
1.0	All chapters	All sections	Document creation.