ComponentOne Studio

Web API Edition

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

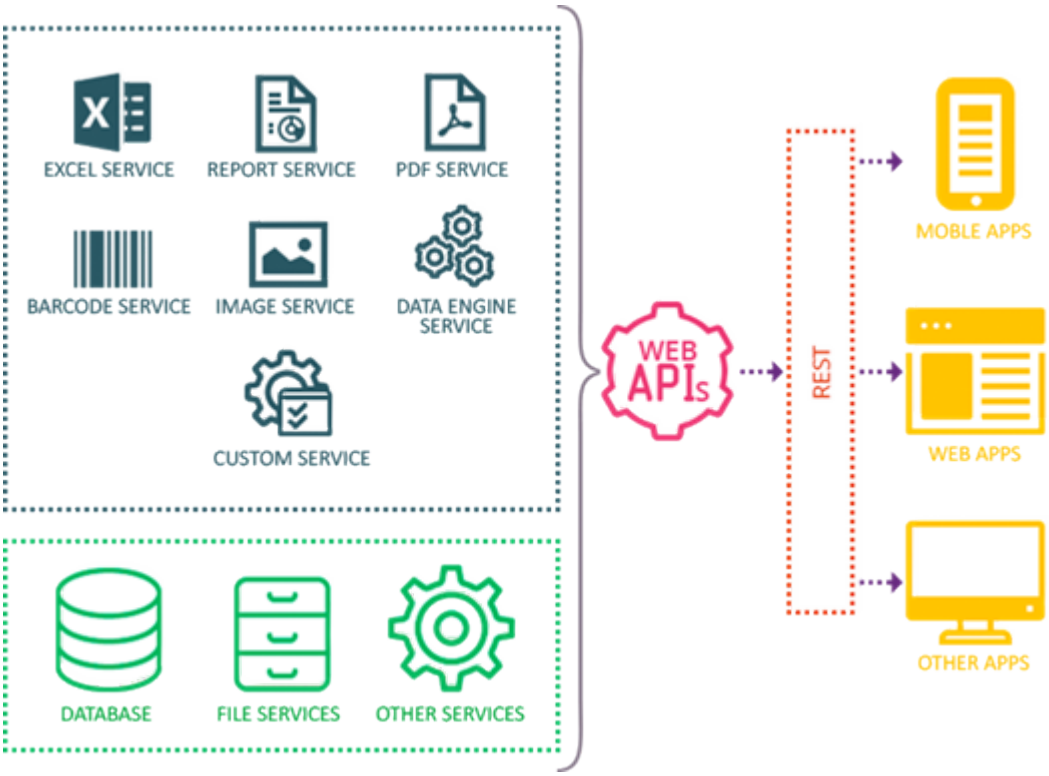
While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

ComponentOne Studio Web API Edition

ComponentOne Studio Web API Edition is a set of HTTP services built on ASP.NET Web API and ASP.NET Core technologies. Available as Visual Studio templates, Web API Edition enables you to create Web API services (RESTful services as well) in Visual Studio. You can add further customizations to your service applications, which can then be exposed to web over HTTP.

Web API Edition is designed to be deployed on multiple platforms and to support various hosting options. It runs as IIS (Internet Information Services) deployment, cloud deployment, and can also be self-hosted. Once deployed, the service application serves as server side for a wide range of clients (such as mobile devices and browsers).



The following topics include information on installing Web API Edition, licensing, technical support, and more. This section is aimed at helping you get started with Web API Edition services, as well as build and consume the services. For specific information on using services, see the Services section.

Installation

Download the installer, **C1StudioInstaller.exe**from http://www.componentone.com/. Follow the steps through the installation wizard to install the ASP.NET DLLs and templates for Web API Edition, its dependency DLLs, stylesheets and scripts.

Web API services are provided as Visual Studio template, which is available after installation of Web API Edition. Once you create Web API service application using the ComponentOne template on Visual Studio, the template adds the references C1.Web.API.dll, C1.C1Excel.dll, and other related assemblies to your service project.

> **Note:** It is advisable that you either log off or restart your Windows 10 machine once you have installed the Web API Edition, and then create the service application, so that the required Environment Variables are set. However, if you install the Web API Edition in Windows 10 followed by creating the service application, the required Environment Variables are not set.

Samples

Samples for the product are installed in the ComponentOne samples folder by default.

C:\Users\<User Name>\Documents\ComponentOne Samples\Web Api

> **Note:** In case you get any missing assembly errors while running the sample, you need to add the required references from **GrapeCity** NuGet source.

To Add Web API Libraries from NuGet

ComponentOne Web API libraries are provided through NuGet, a Visual Studio extension that automatically adds libraries and references to your project. You can add the following NuGet packages for Web API to your Web API service application from http://nuget.grapecity.com/nuget, which is available in Visual Studio once Web API Edition is installed.

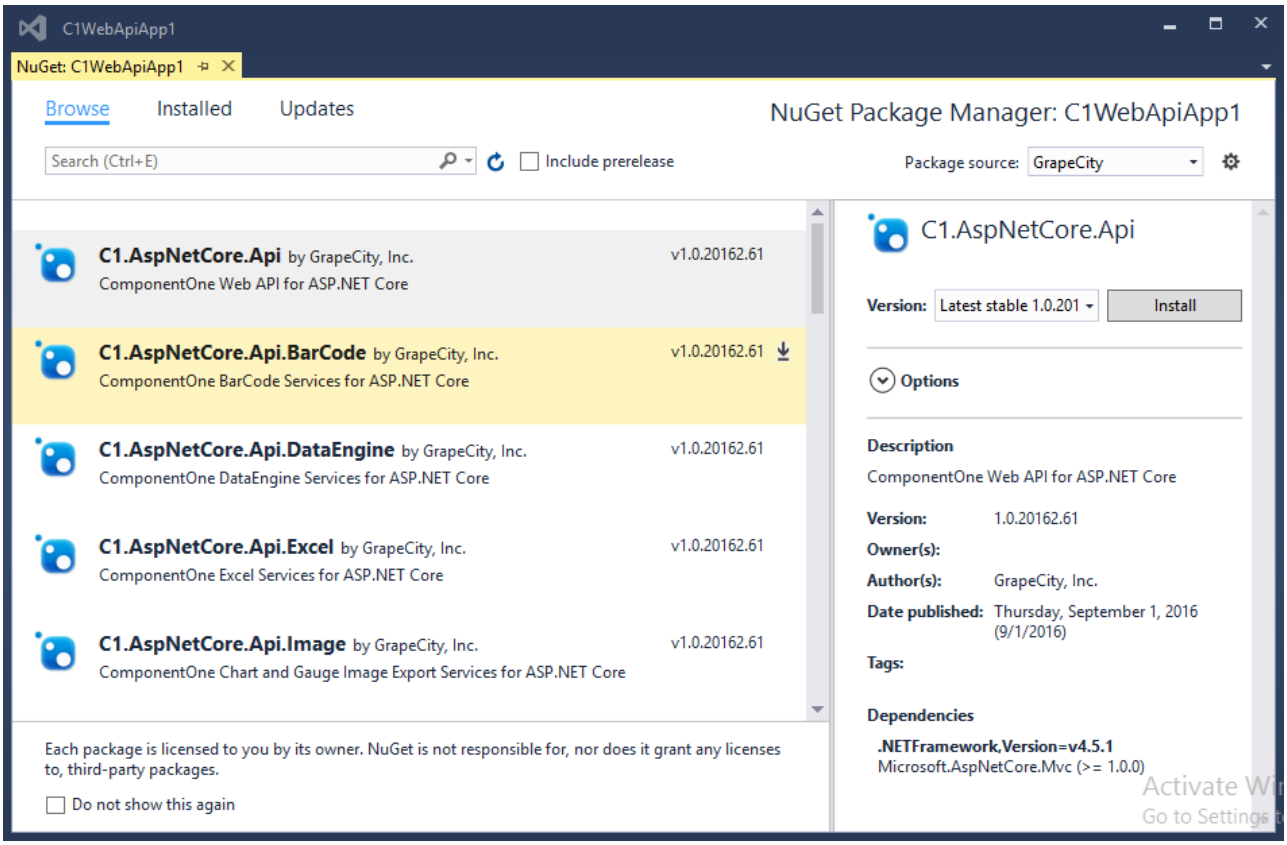
- C1.Web.Api Nuget package
- C1.Web.Api.BarCode Nuget package
- C1.Web.Api.DataEngine Nuget package
- C1.Web.Api.Document Nuget package
- C1.Web.Api.Excel Nuget package
- C1.Web.Api.Image Nuget package
- C1.Web.Api.Pdf Nuget package
- C1.Web.Api.Report Nuget package
- C1.AspNetCore.Api Nuget package
- C1.AspNetCore.Api.Report Nuget package
- C1.AspNetCore.Api.BarCode Nuget package
- C1.AspNetCore.Api.Image Nuget package
- C1.AspNetCore.Api.Excel Nuget package
- C1.AspNetCore.Api.DataEngine Nuget package
- C1.AspNetCore.Api.Document Nuget package
- C1.AspNetCore.Api.pdf Nuget package
- C1.FlexReport Nuget package
- C1.Excel Nuget package
- C1.Document Nuget package
- C1.DataEngine Nuget package

To Install NuGet

1. Go to http://nuget.org/ and click **Install NuGet**.
2. Run the **NuGet.vsix installer**.
3. In the Visual Studio Extension Installer window, click **Install**.
4. Once the installation is complete, click **Close**.

To Add Web API References to your Service Application

1. Create a new Web API project (refer to Configuring Web API).
2. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**.
3. In **NuGet Package Manager**, select **GrapeCity** as the package source.



4. Select the NuGet package according to your project requirements.
5. Click **Install**.

To Manually Create NuGet Package Source

1. In Visual Studio, from the Tools menu select **NuGet Package Manager | Package Manager Settings**. The **Options** dialog box appears.
2. In the left pane, select **NuGet Package Manager | Package Sources**.
3. Click the **Add** button in top right corner. A new source is added under **Available package sources**.
4. Set a **Name** for the new package source, and set the **Source** as http://nuget.grapecity.com/nuget.
5. Click **Update** and then click **OK**.

NuGet package source has been created.

System Requirements

The following must be installed on your system, in order to install Web API Edition.

C1.Web.Api (for Web API 2.2)

- **.NET 4.5** and above
- **Visual Studio 2012, 2013, 2015**
- **IIS 7** or above (if hosted in IIS)

C1.AspNetCore.Api (for ASP.NET Core 2.0)

- **.NET 4.5** and above
- **Visual Studio 2015**
- **IIS 7.5** or above (if hosted in IIS)

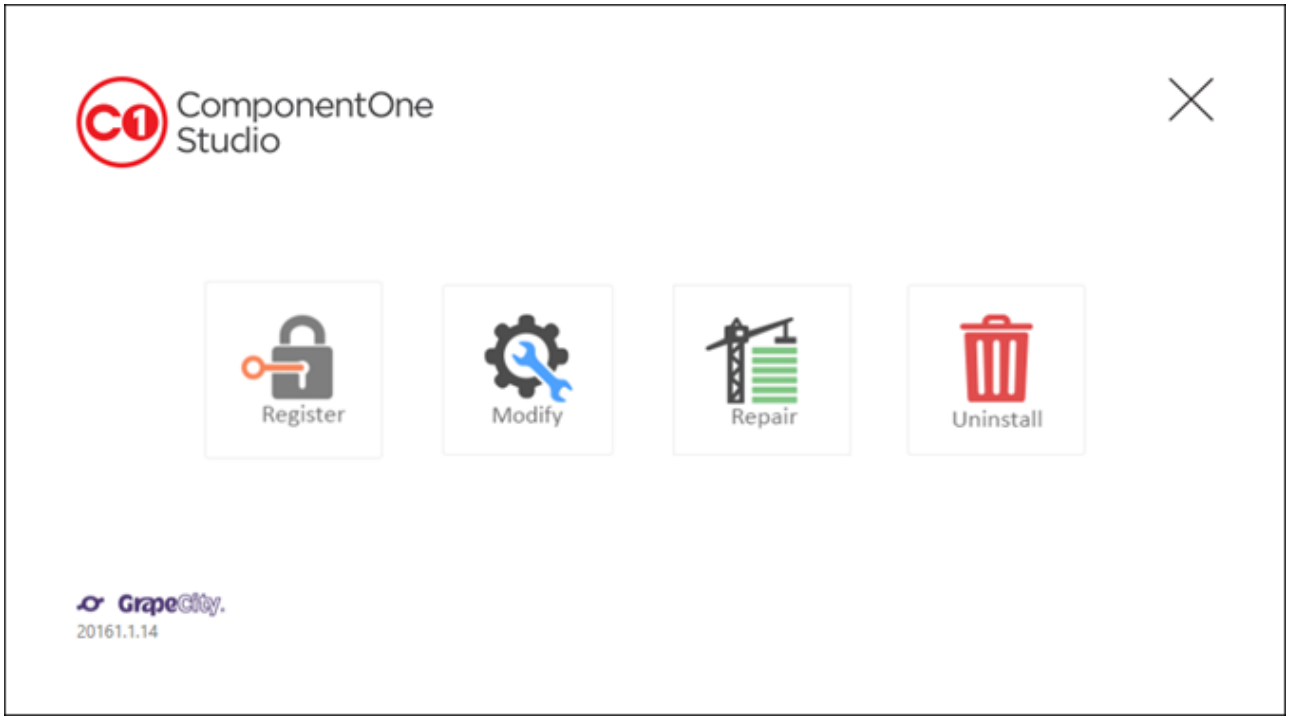
Browser Support

- Internet Explorer 9 or above
- Mozilla Firefox
- Safari

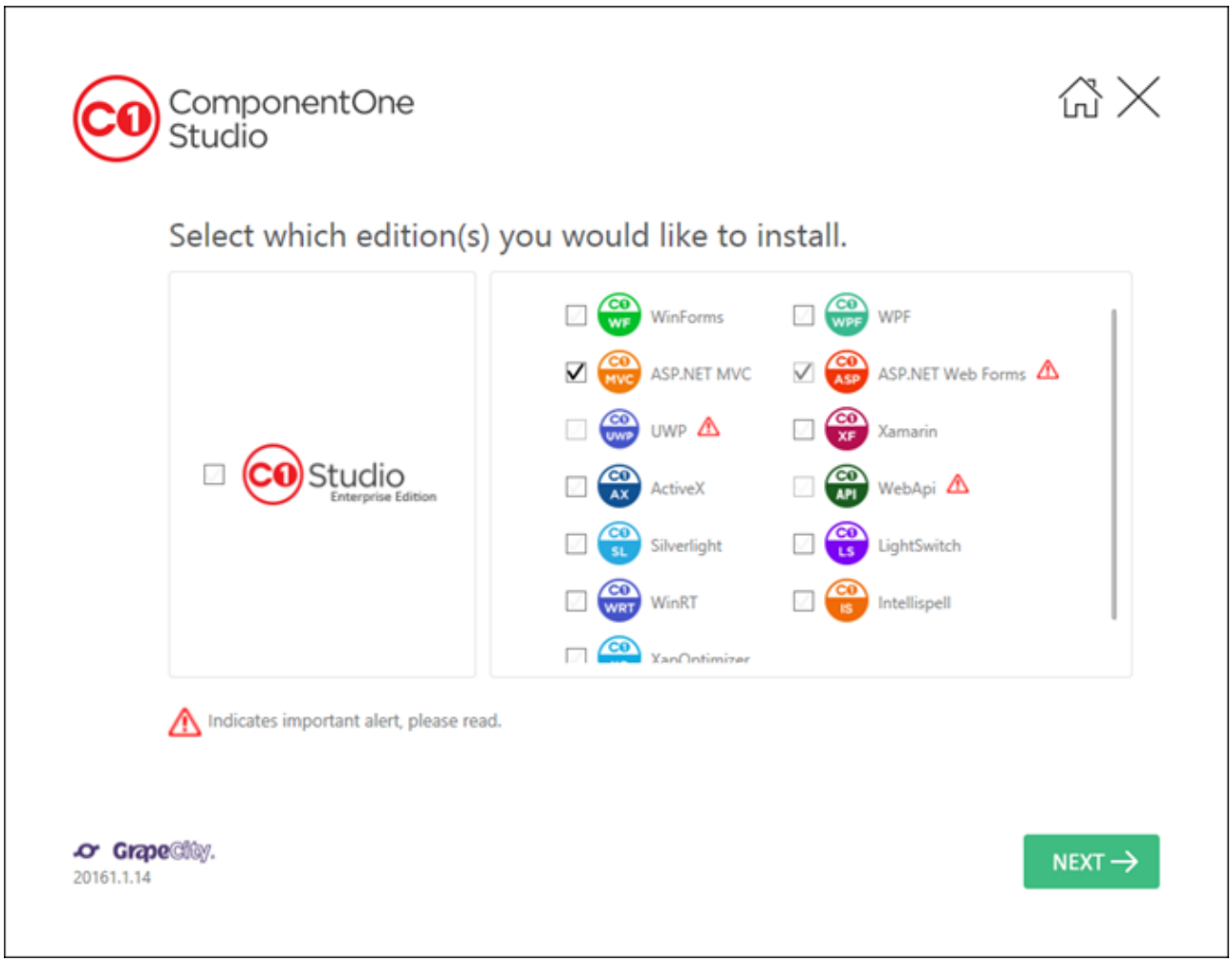
Uninstall Web API

To uninstall **Web API Edition**

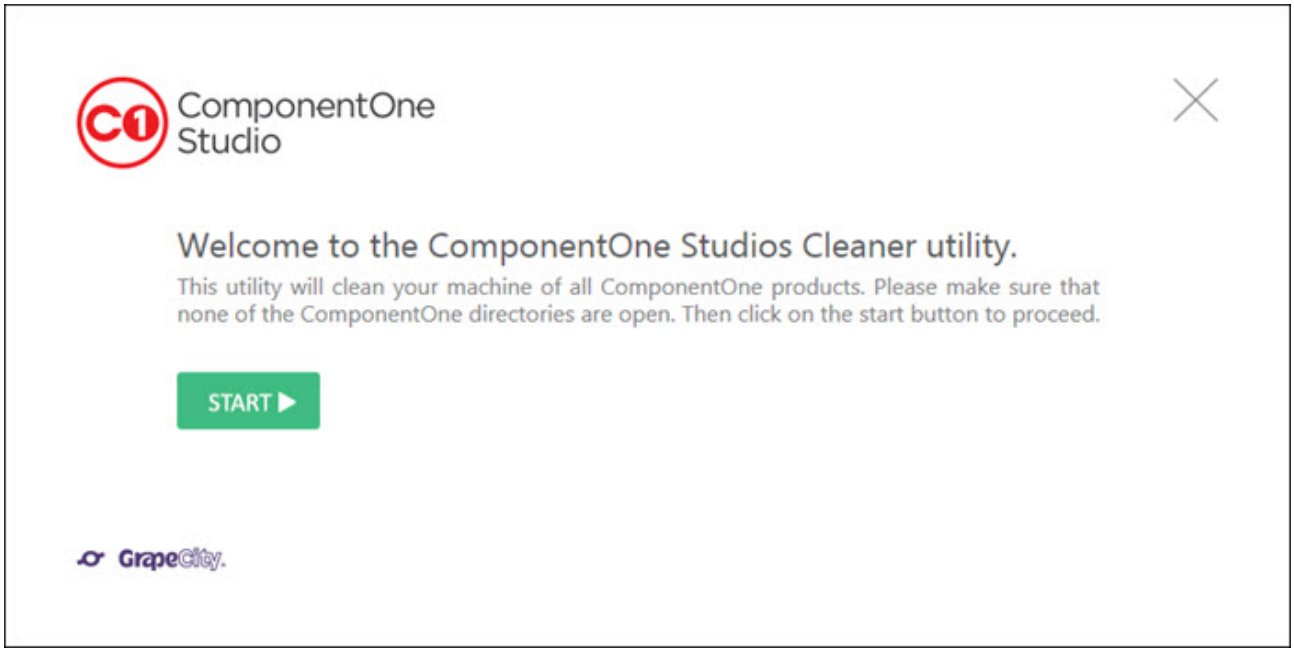
1. Open the **Control Panel** from the Start menu and then select **Uninstall a Program** under **Programs**.
2. Select **ComponentOne Studio** from the list and click **Uninstall** button.
3. In the Installer window that appears, click **Modify**.



4. In the Installer window, remove the check mark against **Web API** edition, and then click **Next** to uninstall the program.



You can also use **ComponentOne Studios Cleaner** utility to clean your machine of all the **ComponentOne** products. All you need to do is download and install the utility from here, run the cleaner, and click **Start** to clean all the registries and directories of **ComponentOne Studio**.



Licensing

Web API Edition is a part of ComponentOne Studio Enterprise and follows the standard subscription model like all other ComponentOne products.

Runtime license checks are done for Web API services, which means library requires a unique key to be validated at runtime. C1.Web.Api (for Web API 2.2) services use Development level license mode, while C1.AspNetCore.Api(for ASP.NET Core) services use Application level license mode.

What all is included in a subscription of Studio Enterprise?

A subscription of ComponentOne Studio Enterprise includes all updates, bug fixes and official releases for one year. For example, if you purchase 2015 v1, you are entitled to use versions 2015 v2, 2015 v3, 2016 v1 and all other versions released in between these versions. A subscription entitles you to unlimited, royalty-free app redistribution using any valid version of ComponentOne controls.

> For more information on the ComponentOne licensing model, visit http://www.componentone.com/SuperPages/Licensing/.

How does Licensing Work?

Web API 2.2

You can work on multiple Web API 2.2 applications using the same license. The licenses.licx file containing the licensing information would automatically be generated by the Visual Studio template based on the ComponentOne license present on the system. The contents of the license file are as shown below.

Licenses.licx

```
C1.Web.Api.LicenseDetector, C1.Web.Api
```

You can even distribute your application across multiple users without worrying about paying any royalty or user fees for distribution. This is a huge cost-saver and also makes embedding ComponentOne controls into applications easier as there are no licensing pre-requisites.

When you download and install ComponentOne products, you are presented with the chance to activate a license. If you would rather evaluate our tools before buying, you can skip the license activation process. At that time, you will have a 30-day evaluation period. During that period, you will be able to use all features of ComponentOne products. At the end of the 30-day period, you will not be able to build applications that include unlicensed ComponentOne controls.

You can purchase and activate a license at that point or you can request an evaluation key that will grant you another 30-day evaluation period. Once you have the evaluation key, you would activate it through the ComponentOne License Activation utility found in your ComponentOne Start menu.

How to purchase a license or request a 30-day evaluation key?

- **License:** You can purchase a license through our online store or by contacting our sales department. As soon as you purchase, a key will be emailed to you.
- **Evaluation Key:** You can get a 30-day evaluation key by contacting sales or through our website's My Account section (you will need an account to proceed with this option).

Complete the following steps to generate a 30-day evaluation key through our website:

1. Visit www.componentone.com.
2. Open My Account and login or create an account. If you create an account, you must use the same email you used when you initially downloaded ComponentOne products.

3. Find your current evaluation listed under **My Evaluations** and click **Extend Evaluation** button.
4. You will receive an email with our key which you will then need to activate on your machine.

How to activate the key on your machine?

After you have received your license or evaluation key through email, complete the following steps to activate it on your machine.

1. Find the **ComponentOne License Activation** utility at the following location and open it: (**$/Program Files (x86)/ComponentOne/C1StartMenu**) .
2. Enter your key, choose an activation method, and follow the on-screen instructions.

> If you activate a 30-day evaluation key, your trial will proceed exactly as your previous trial did. If you activate a license key, you will be entitled to royalty-free perpetual use.

How to manually add the license file to the application?

1. In the **Solution Explorer**, right click the project and select **Add | New Item**. The **Add New Item** dialog appears.
2. In the **Add New Item** dialog, select **C# | General** and select **Text File** in the right pane.
3. Name the text file as **licenses.licx**.
4. Add the following to the text file:

Licenses.licx

```
C1.Web.Api.LicenseDetector, C1.Web.Api
```

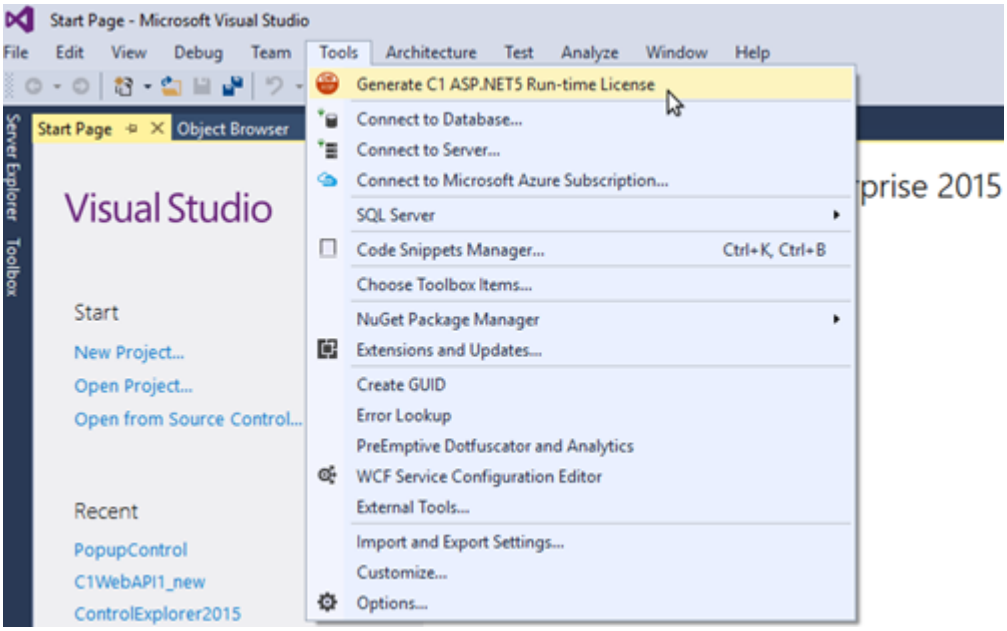
ASP.NET Core Web API

Applications that are created using ASP.NET Core Web API services require a unique license key. These licenses are bound to specified applications and are to be validated at runtime. Licenses are added to the applications by generating the same using runtime licensing add-in on Visual Studio. The add-in for generating Run-time License is available in all applications using ASP.NET Core Web API.

License key for the ASP.NET Core Web API applications created on Visual Studio is generated through **GrapeCity License Manager** add-in provided in Visual Studio.

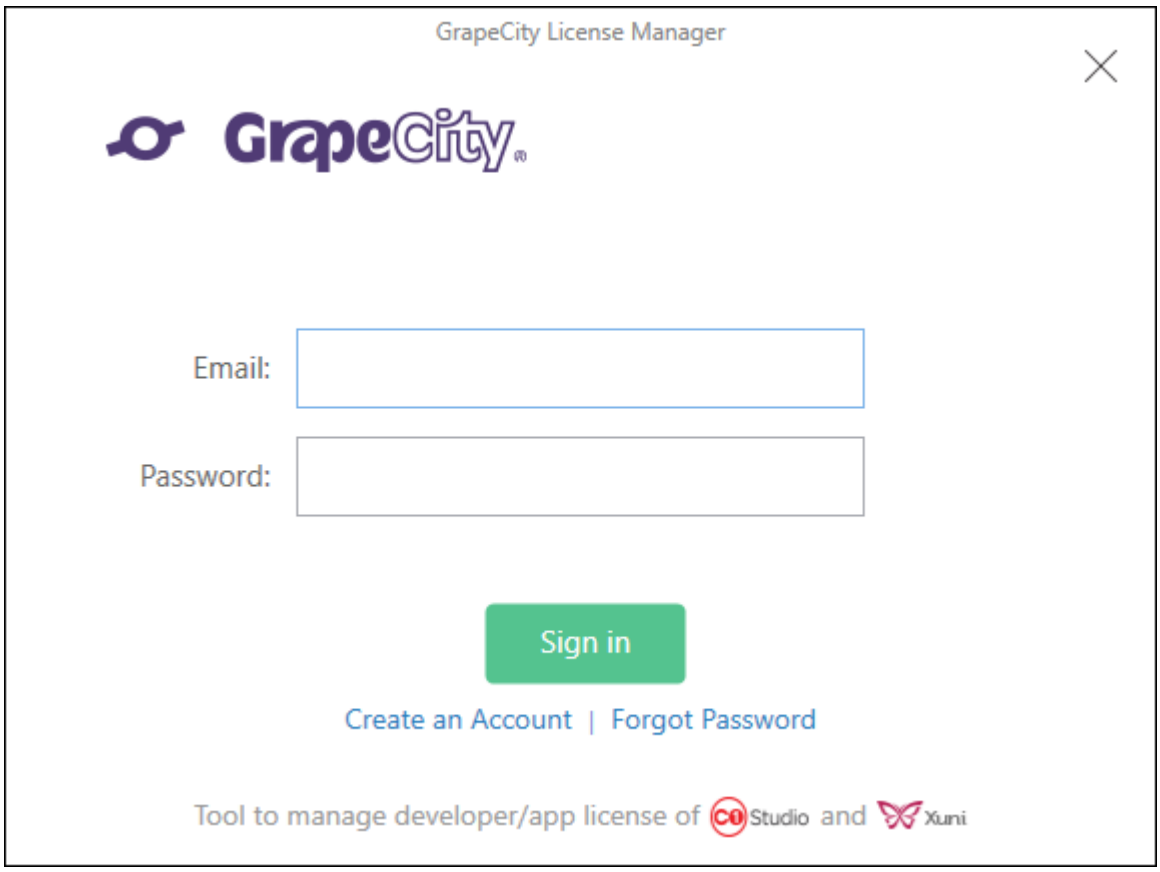
To Generate License using GrapeCity License Manager Add-in

The add-in for generating Run-time License is available for all the Web API applications which are created in Visual Studio.
The add-in is visible in options within **Tools** menu in Visual Studio.

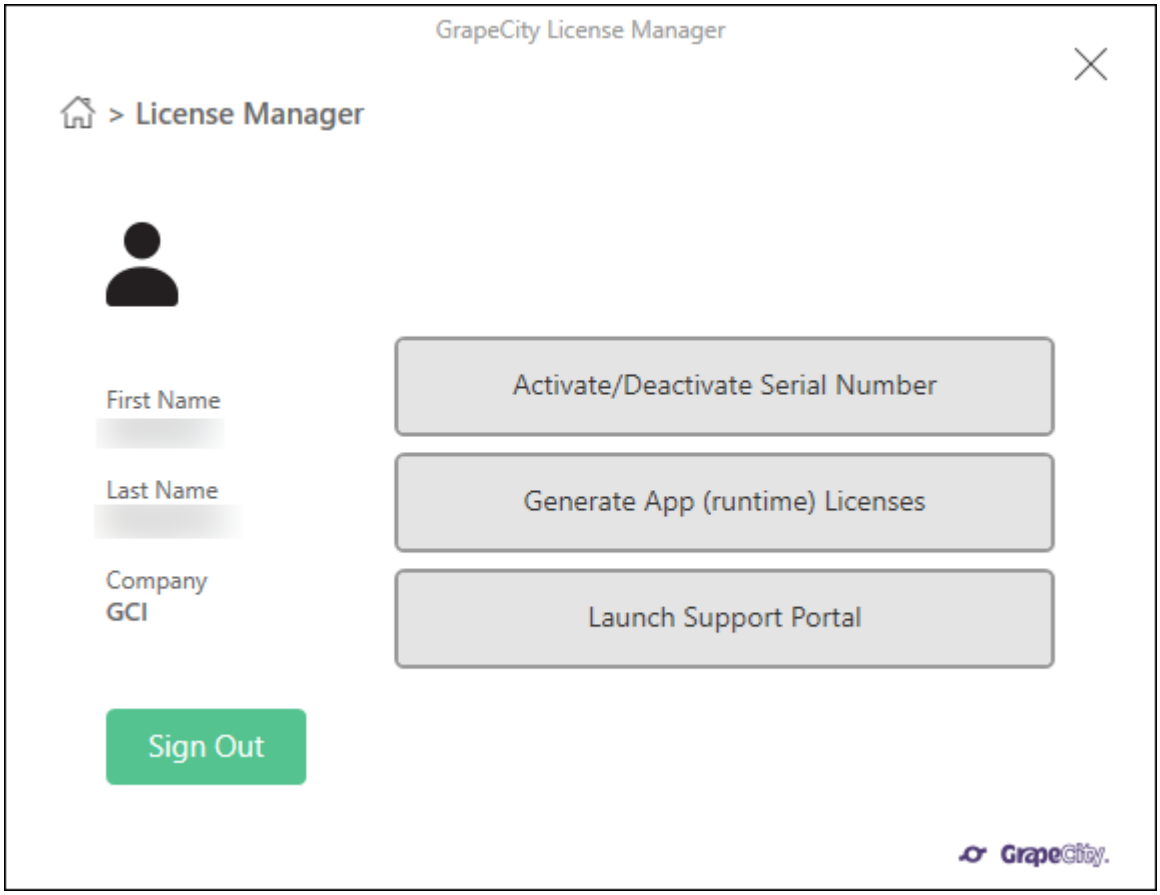


Complete the following steps to generate a trial or full license for your ASP.NET Core applications using Visual Studio add-in:

1. Create a new ASP.NET Core Web API application. For more information. see Configuring Web API topic.
2. Add the required NuGet packages to your application through the **NuGet Package Manager** (Refer to Installation for steps to add Nuget packages).
3. Click the **GrapeCity License Manager** add-in, from options within the **Tools** menu.
4. In the GrapeCity License Manager window, enter your registered **Email** and **Password** to log in. In case you are not registered with GrapeCity, you can create a new account using **Create an Account** option.



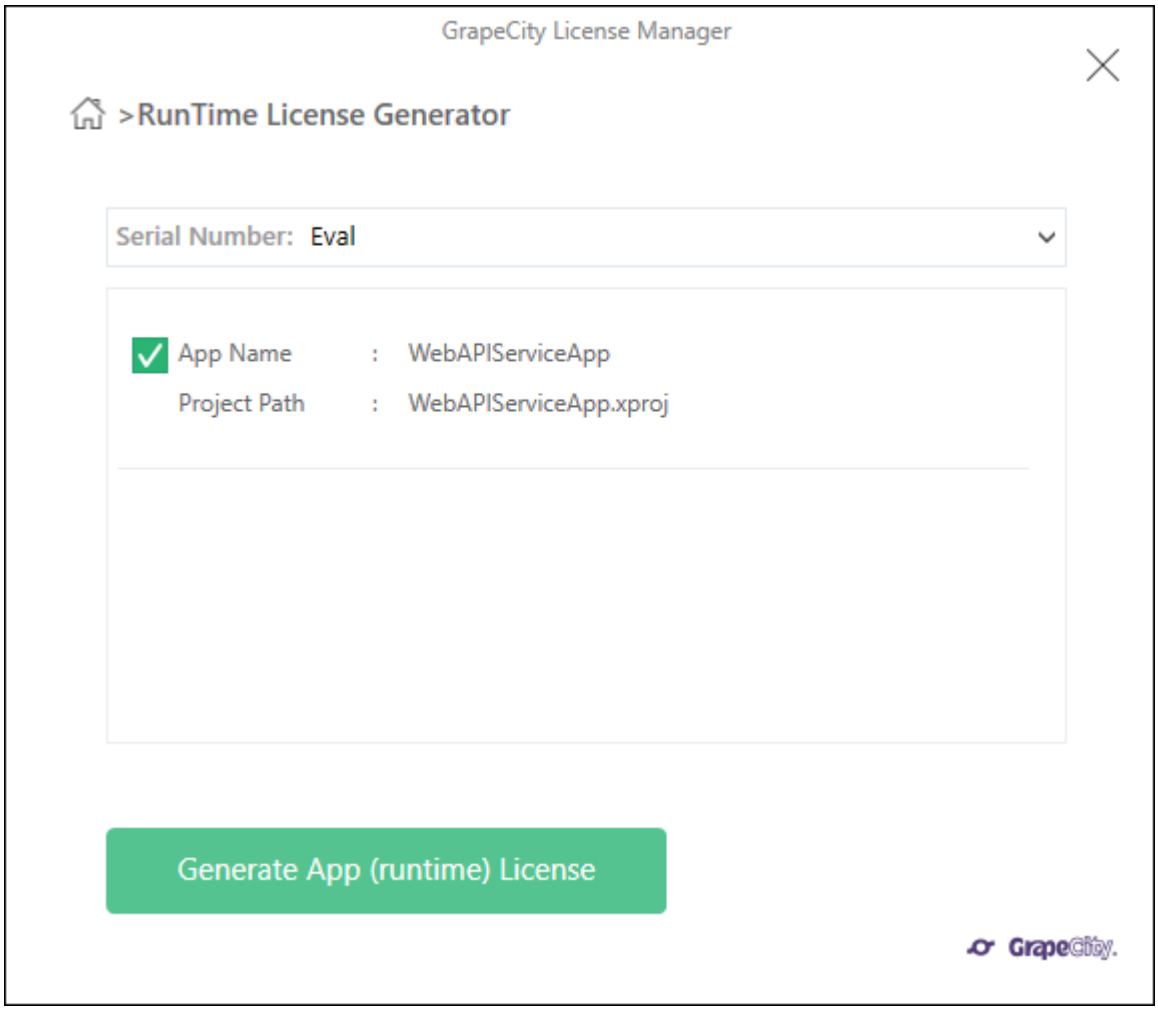
5. Once you log-in, you can choose any one of the following options. Your login information will be cached for 30 days in Visual Studio.



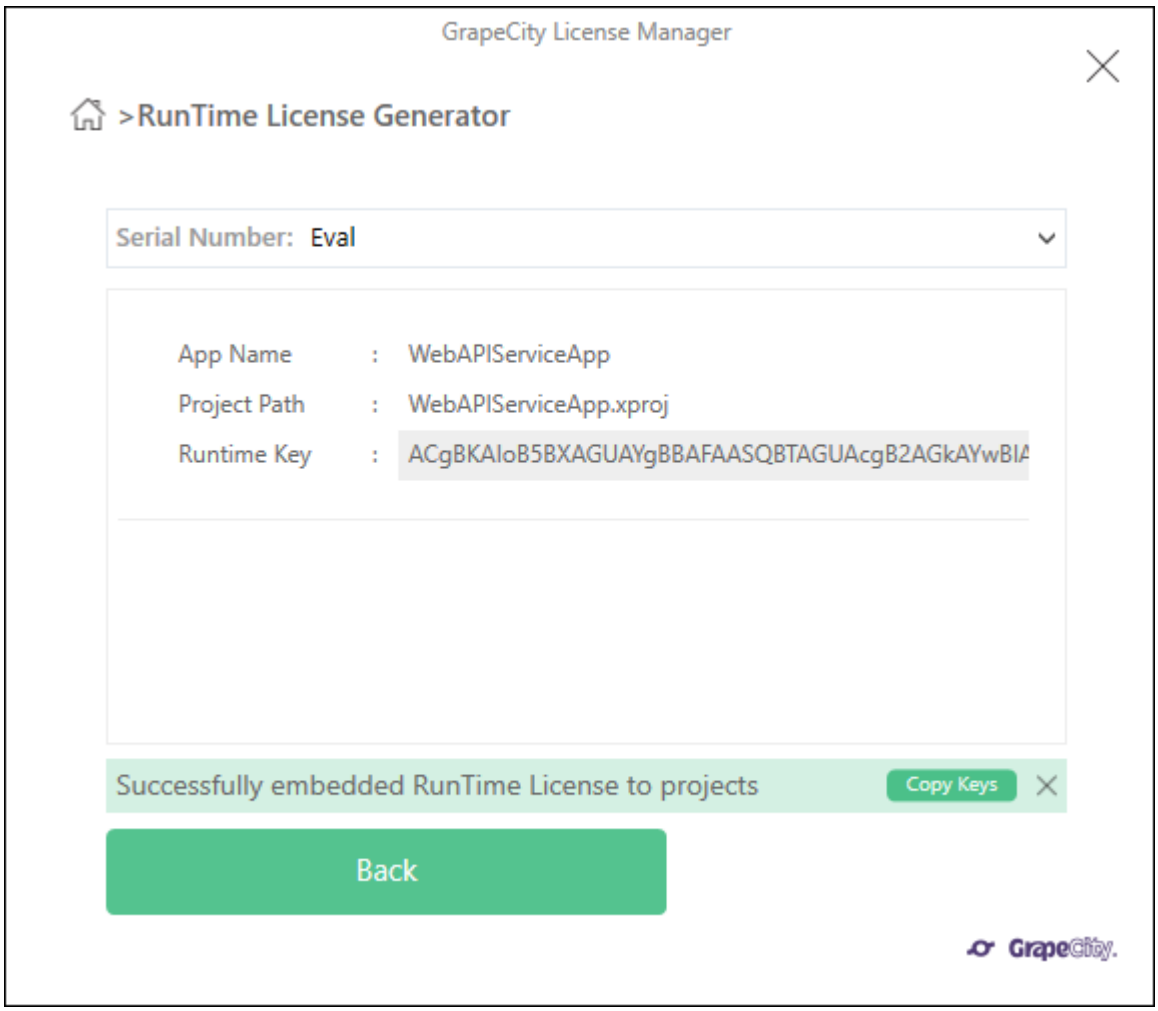
- **Activate/Deactivate Serial Number** - Allows the users to activate or deactivate the serial number using the internet, C1 Website, By Email, or Over the phone.
- **Generate App (runtime) Licenses** - Allows the users to generate and activate license for each Web API application you are working on your system.
- **Launch Support Portal** - Allows the user to open http://supportone.componentone.com/ website where the users can communicate with the support team for any support related issues.

6. In the GrapeCity License Manager window, select **Generate App (runtime) Licenses** option to generate a license for your Web API application. Once you select this option the tool will detect the relevant C1 assemblies that are being used in the project or active solution.

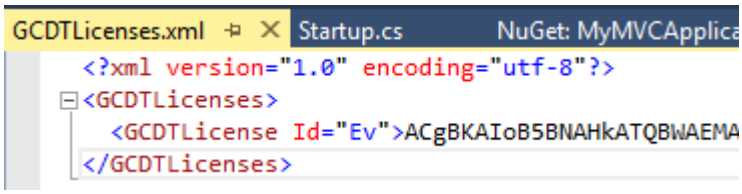
7. In the GrapeCity License Manager window, edit or select the **Serial Number** from the drop-down list, and then click **Generate App (runtime) License** to generate a license. In case you have a serial number which is not activated using **C1LicenseActivation.exe** tool, you can activate the serial number using the **GrapeCity License Manager**.



8. Once you click on **Generate App (runtime) License**, a success message appears in the GrapeCity License Manager window.



An xml file **GCDTLicenses.xml** file is created and placed in the same location as the project file and it is an embedded resource to the project. It is recommended to close GCDTLicenses.xml and project.json files before executing the GrapeCity License Manager tool.



If you are generating a evaluation license, your application is now ready to use for evaluation purposes. You can repeat this process for any number of applications. You must generate a new evaluation license for each application because they are unique to the application name.

Note: The generated license key for the application under trial license will stop working after 30 days.

ASP.NET Core Web API Evaluation Version

Note that the Evaluation Version is limited to 30 days and this begins when you generate your first runtime license. This should not be distributed with published apps.

You may extend your license by contacting our sales team.

ASP.NET Core Web API Fully Licensed Version

Fully licensed keys do not expire so long as your application uses a version of Studio Enterprise or Ultimate included

with your subscription. You can update applications beyond your subscription end date so long as you continue to use a valid version of Web API Edition.

If you purchase ComponentOne Ultimate you are given a serial number. This serial number must be registered before you can generate full runtime licenses.

Complete the following steps to register your ASP.NET Core Web API Serial Number:

1. Visit https://www.componentone.com/MyAccount/MyLicenses.aspx and login using the ID that you want to use to generate runtime licenses for your Web API service applications.
2. Click **Register a Product**.
3. Enter the **Serial Number** and **Purchase Date** and click **Register Product**.

> **Note:** The ComponentOne account that registers the serial number is the only account that can generate runtime keys for applications. This account, however, can generate keys from any system.

Licensing ASP.NET Core Web API on Website

Applications that are created using ASP.NET Core Web API services require a unique license key. These licenses are bound to specified applications and are to be validated at runtime. Licenses are added to the applications by generating the same on ComponentOne website.

Complete the following steps to generate a trial or full license for your applications:

1. Create a new Web API service application (for detailed steps refer to Configuring Web API).
2. Add the required NuGet packages to your application through the **NuGet Package Manager** (see Installation).
3. Visit http://www.componentone.com/MyAccount/MyASPNet.aspx.

   > **Note:** You must create a ComponentOne account and login to access this web page.
4. If you are generating a full license, select your serial number from the drop-down menu at the top of the page. If you are generating a trial license, leave it selected as **Evaluation**.
5. In the **App Name** textbox, enter the name of your application.

   > To find your application's name, right click your project's name in the **Solution Explorer** and select **Properties**. Open the **Application** tab, the application name is the same as the **Default Namespace** displayed.
6. Click the **Generate** button. A runtime license will be generated in the form of a string.
7. Copy the runtime license and complete the following steps to add it to your application.
   1. Open your application in Visual Studio.
   2. In the **Solution Explorer**, right click the name of your project.
   3. Select **Add | New Item**. The **Add New Item** dialog appears.
   4. Under installed templates, select **C# | Class**.
   5. Set the name of the class as License.cs and click **OK**.
   6. In the class License.cs, replace the content with the copied runtime license key.

      C#

      ```
      public static class License
      {
      public const string Key = "Your Key"
      }
      ```
   7. From the **Solution Explorer**, open Startup.cs and assign the key to it as shown below.

      C#

      ```
      public void ConfigureServices(IServiceCollection services)
      {
      ```

```
C1.Web.Api.LicenseManager.Key = License.Key;
services.AddMvc(); //To add Mvc services
```

If you are generating a trial license, your application is now ready to use for trial purposes. You can repeat this process for any number of applications. You must generate a new trial license for each app because they are unique to the application name.

ASP.NET Core Web API Evaluation Version

Note that the Evaluation Version is limited to 30 days and this begins when you generate your first runtime license. This should not be distributed with published apps.

- Generate an evaluation key for each app (unlimited for 30 days).

You may extend your license by contacting our sales team.

ASP.NET Core Web API Fully Licensed Version

Fully licensed keys do not expire so long as your application uses a version of Studio Enterprise or Ultimate included with your subscription. You can update applications beyond your subscription end date so long as you continue to use a valid version of Web API Edition.

If you purchaseComponentOne Ultimate you are given a serial number. This serial number must be registered before you can generate full runtime licenses.

Complete the following steps to register your ASP.NET Core Web API Serial Number:

1. Visit https://www.componentone.com/MyAccount/MyLicenses.aspx and login using the ID that you want to use to generate runtime licenses for your Web API service applications.
2. Click **Register a Product**.
3. Enter the **Serial Number** and **Purchase Date** and click **Register Product**.

Visit http://www.componentone.com/MyAccount/MyASPNet.aspx to generate runtime key for each app.

> **Note:** The ComponentOne account that registers the serial number is the only account that can generate runtime keys for applications. This account, however, can generate keys from any system through the ComponentOne website.

Adding Web API Client JavaScript

Client application sends request to Web API service application. Web API Edition provides JavaScript client code to help raise this request in web platform. ComponentOne Web API Client JavaScript file, **webapiclient.min.js**, provides export/import extension JavaScript for Wijmo and MVC controls. It provides API to call the server side from client application.

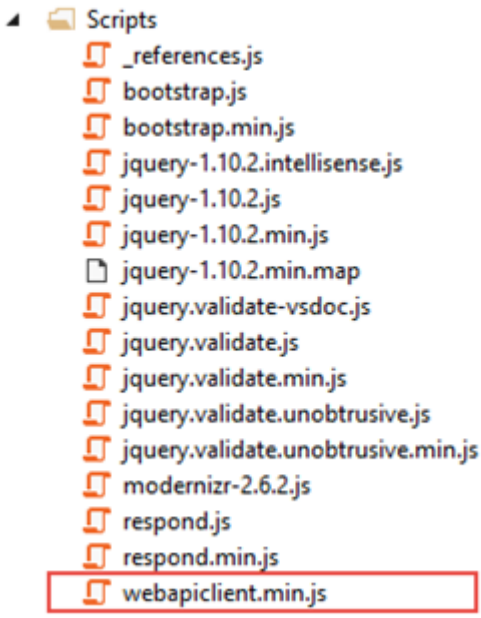
Note that, for consuming REST based API services, client JavaScript file is not required. The request methods- GET and POST- are used for calling RESTful services. For more information on using services, see the Services section.

> **Note:** webapiclient.min.js file is by default installed in C:\Program Files (x86)\ComponentOne\Web Api Edition\client.

Complete the following steps for adding and using Web API Client JavaScript file in your client application:

MVC

1. Add **webapiclient.min.js** file to **Scripts** folder of your client project.



2. Add the **webapiclient.min.js** reference to **Views | Shared | _Layout.cshtml** file. Drag the **"webapiclient.min.js"** file from **Solution Explorer** to the location of scripts settings at the top of this file.

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - ComponentOne ASP.NET MVC Project</title>
    @Styles.Render("~/Content/css")
    @RenderSection("styles", required: false)
    @Html.C1().Resources("default")
    <script src="~/Scripts/webapiclient.min.js"></script>
</head>
```

HTML

- Add **webapiclient.min.js** file to a folder where your client project resides.
- Add reference to **webapiclient.min.js** file within <head> tag of your HTML page along with other Wijmo references. If you place the client javascript file within "Scripts" folder, you can add the following reference.

1. Add **webapiclient.min.js** file to a folder where your client project resides.
2. Add reference to **webapiclient.min.js** file within <head> tag of your HTML page along with other Wijmo references. If you place the client JavaScript file within "Scripts" folder, you can add the following reference.

HTML

```
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

Working with Web API

Web API Edition is a set of next generation services which can be implemented with ASP.NET Web API or ASP.NET Core. You need to first create a service project in Visual Studio, and deploy it on IIS or self-host it. Client applications (MVC, HTML based, desktop applications or other generic applications) can then call these Web API services for exporting and importing excel files, exporting images (to desired format), generating excel from given data and template, view reports and generating barcode from a given text. For more information related to the Web API Services, see Services topic.

The following topics discuss how to build a Web API service project, deploy it, and create a client application that will consume the service. See the System Requirements before proceeding.

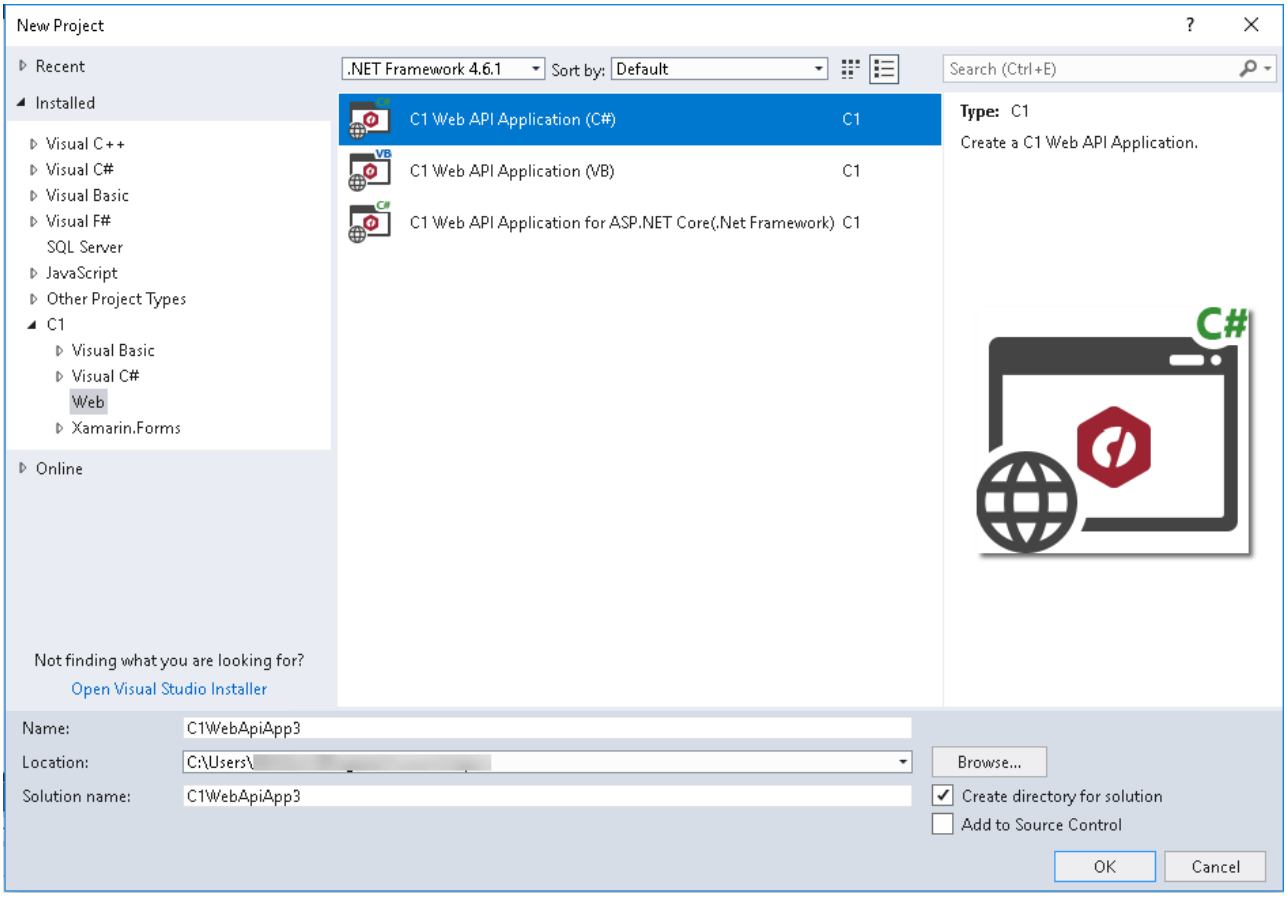
Configuring Web API

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a wide range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework. The **C1Studio Web API Edition** is a set of API's that enable exporting of FlexChart, Gauge, Image, and import and export of Excel sheets with FlexGrid. The APIs are available as Visual Studio templates with support for Web API 2.2 and ASP.NET Core 2.0 Web API.

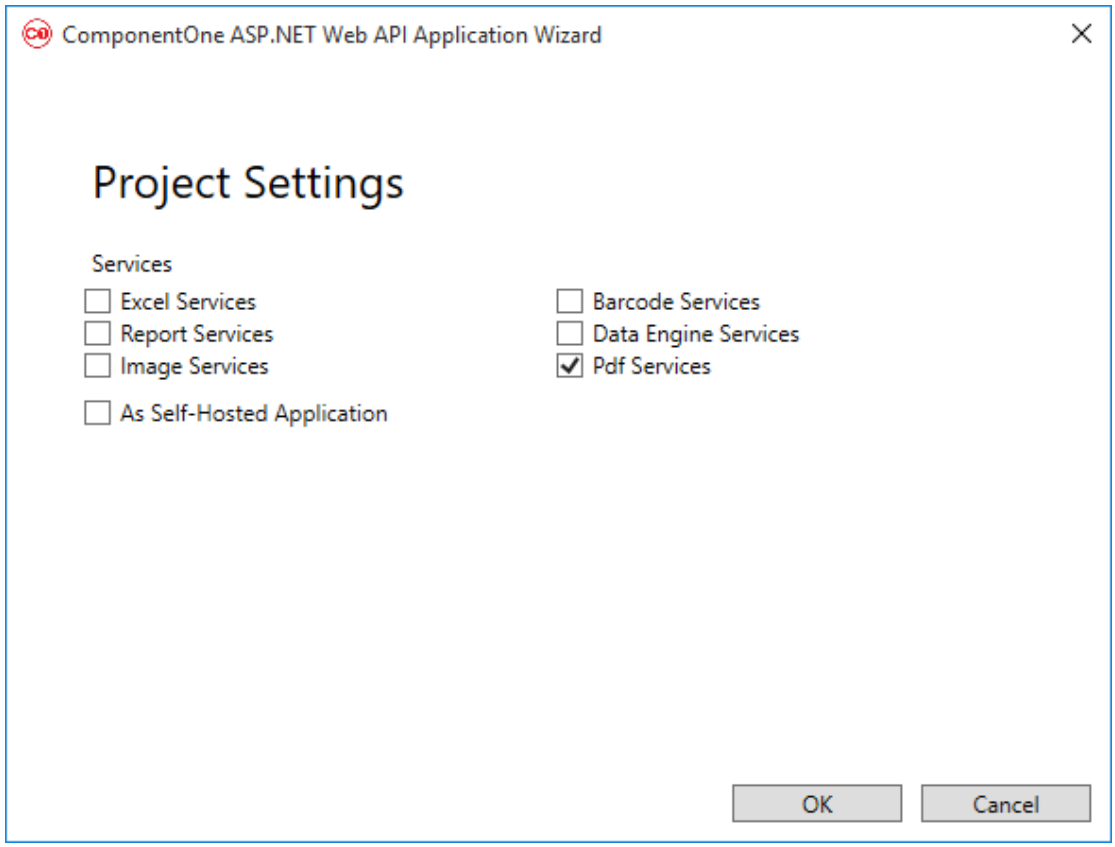
This topic demonstrates how to create a Web API service application in Visual Studio. This builds the server side (or endpoint) for a client application. The client application sends a request to the endpoint for accessing its service and gets a response in return.

Web API 2.2

1. In Visual Studio, select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **C1 | Web | C1 Web API Application (C# or VB)** to create a new C1 Web API Service application.



3. Set a **Name** and **Location** for your application, and then Click **OK.**
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select the Web API Services according to your project requirements.



5. Once you have selected the **Services** from the wizard, click **OK** to create a new C1 Web API Service application. For more information, see Services topic.

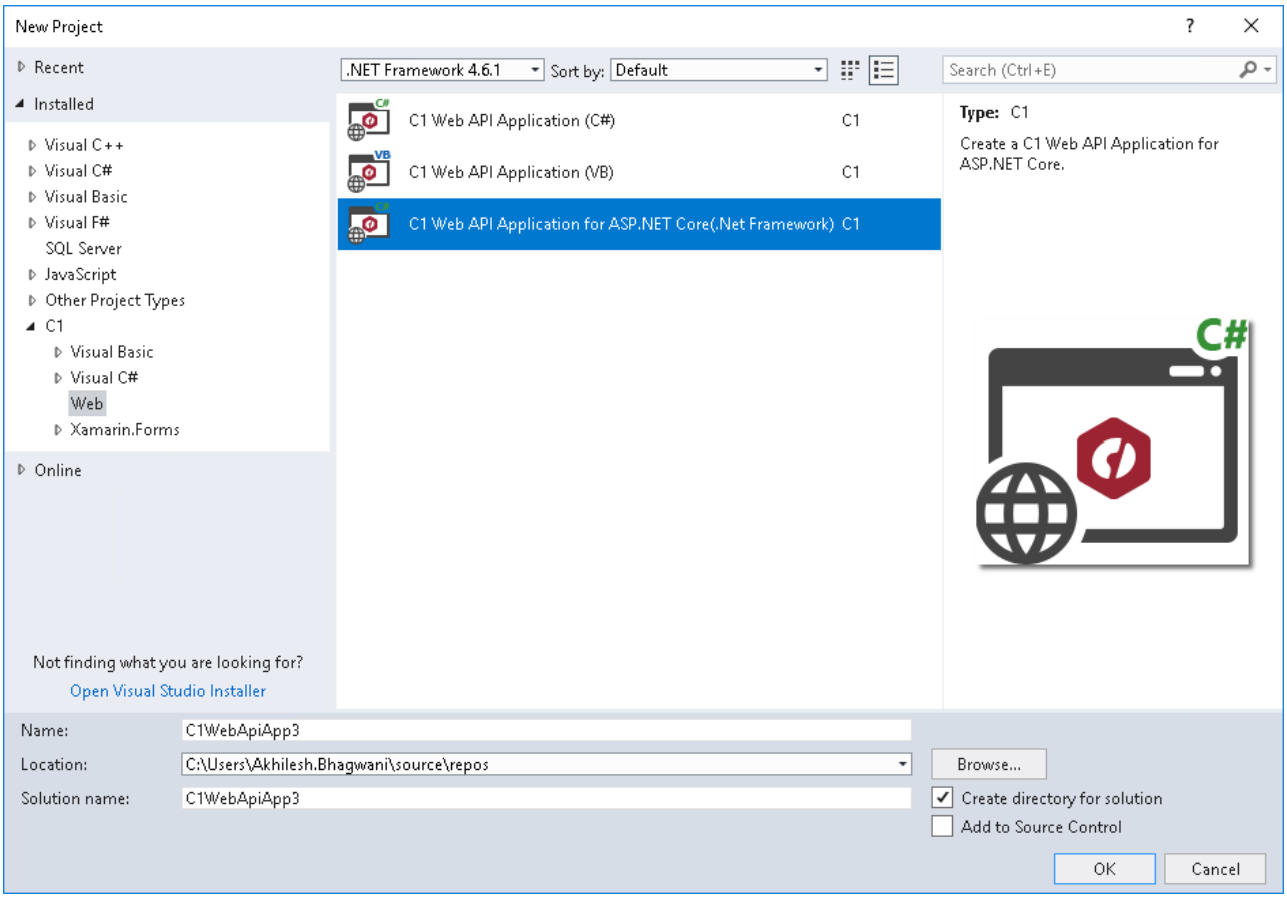
   In this new project, ComponentOne template adds the references C1.Web.Api.dll, C1.C1Excel.dll, and other related assemblies.

   > **Note:** license.licx file is automatically added to your application. For more information on how to add licensing to your application, refer to Licensing.

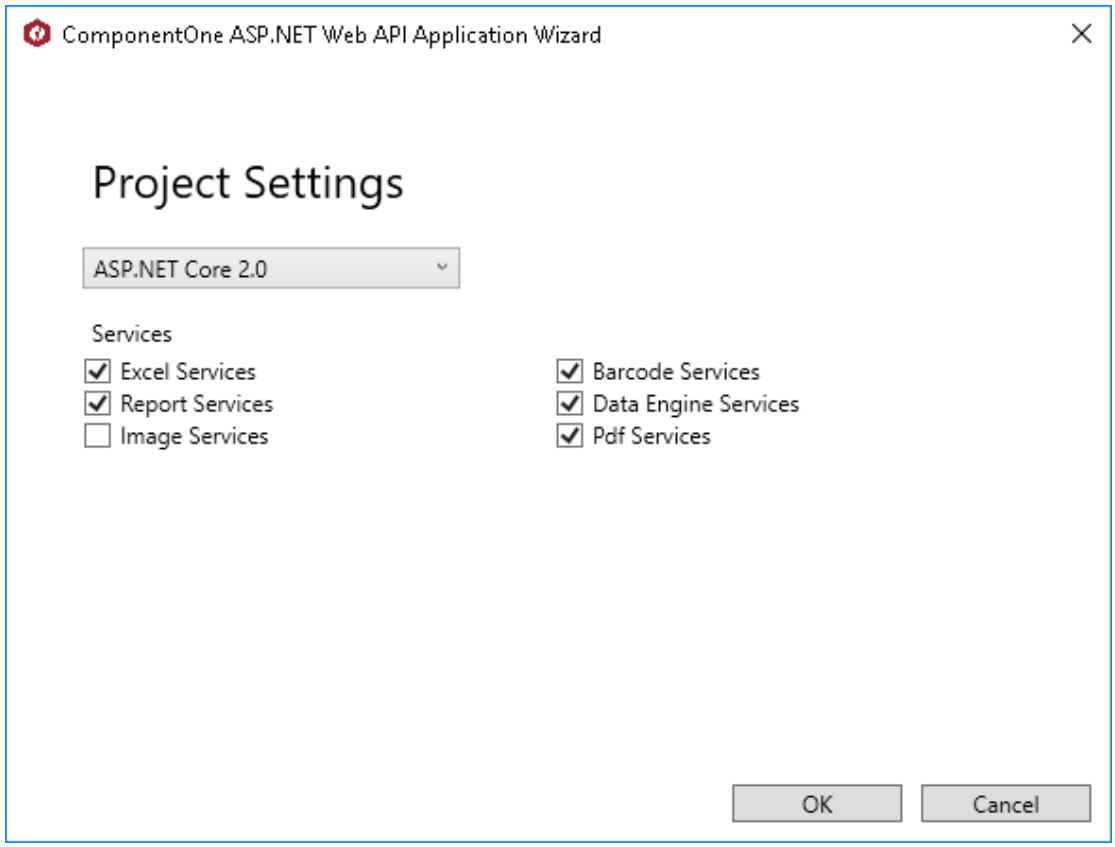
6. You can also access data from a configured remote or local storage. For more information on how to configure local data storage, see Configuring Storage, Configuring Server-side Data, and Configuring .NET collections.
7. You have configured the service project. Deploy the project on IIS (for service application hosted on IIS) or simply run the application (if Self-hosted), and start using it.

ASP.NET Core

1. Open Visual Studio, and select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **C1 | Web | C1 Web API Application for ASP.NET Core**.



3. Set a **Name** and **Location** for your application.
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select the ASP.NET Core Framework version and Web API Services according to your project requirements.



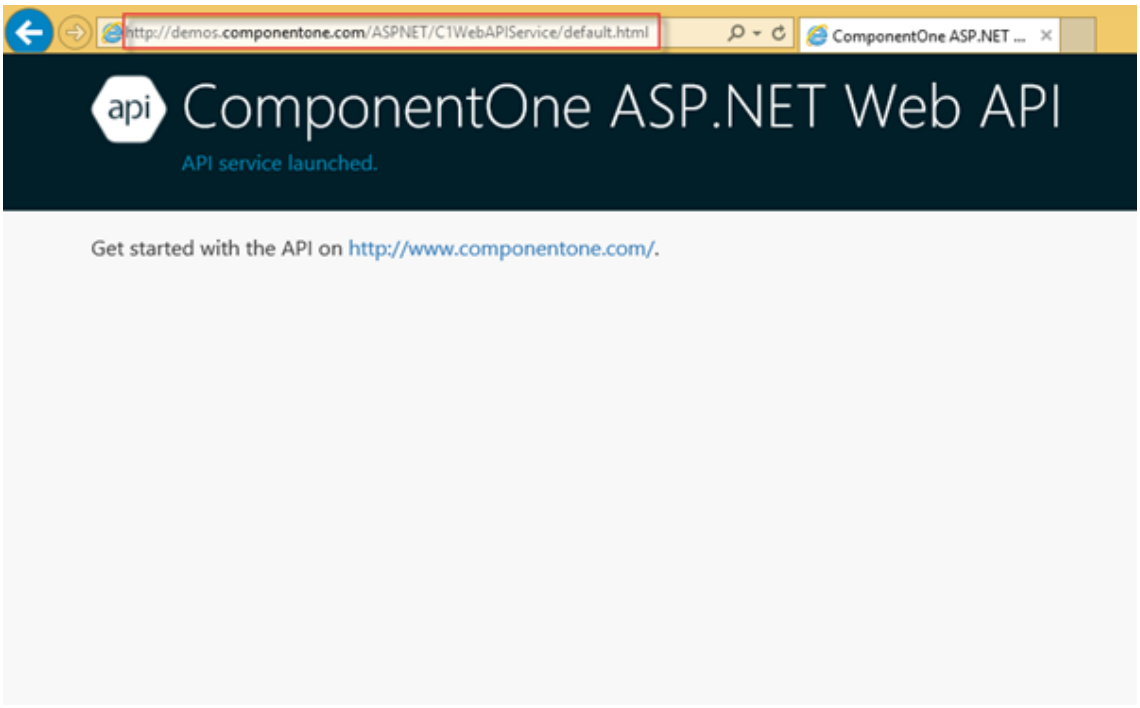
5. You can also access data from a configured remote or local storage. For more information on how to configure local data storage, see Configuring Storage, Configuring Server-side Data, and Configuring .NET collections.
6. Add runtime license to the application (for more information on how to add license to your WebAPI 3 application, refer to Licensing.
7. Run the application in Visual Studio.

In this new project, ComponentOne template adds the references C1.Web.Api.dll, C1.C1Excel.dll and other related assemblies.

Note: To secure your data, you can customize the **WEB API** project to configure security according to your requirements.

Your server-side application is successfully created. You can now use the generated URL of launched Web API service to make a call to the service project from your client application. You can use Web API services such as Report, Excel, Image and Barcode in the client application. For more information, see Services topic.

The following image shows a hosted Web API service application in browser.



In the example above, the service has been hosted at http://demos.componentone.com/ASPNET/webapi/default.html.

Once you have successfully hosted a **Web API URL** for Services, you can access and call these Web API service for exporting and importing excel files, excel images, generate excel from given data and template, and generating barcode from a given text. For more information on how to work with Web API Services, see Services topic.

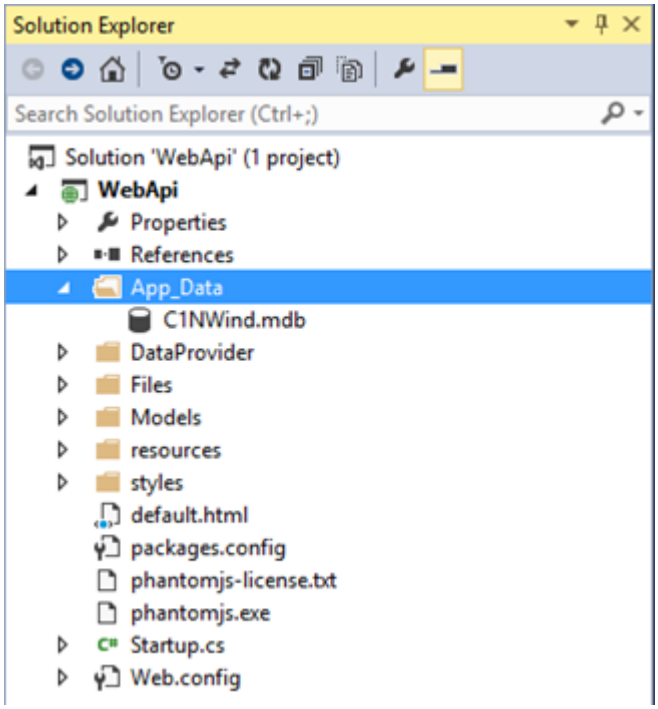
Configuring Server-side Data

You can also use data from a database file to generate, merge, and convert excel. You need to configure the connection string to use the required data from the database file. The following example uses a locally available mdb file named C1NWind as storage, and OleDb data provider. However, your database file could exist locally or reside on SQL server, and you may use any other data provider.

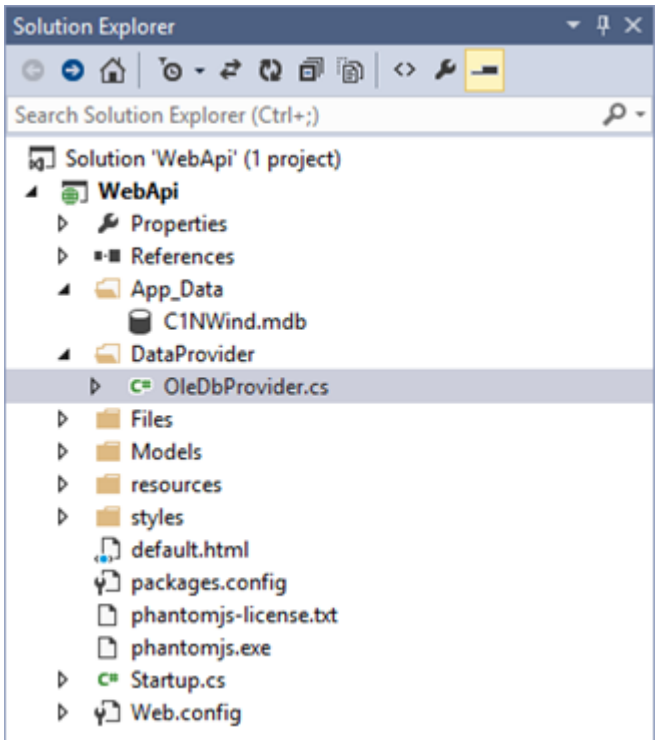
> **Note:** Once you have configured your server-side data, you can create client side application to call the services to use the data. For more information, see Web API Services topic.

Complete the below steps to configure the connection string for C1NWind.mdb database file available locally.

1. Create a new folder in your Web API service application, and add the database file in it. Here we add the file in App_Data folder.



2. Create an OleDbProvider.cs file in your application, and add the **OleDbProvider** class to it.



3. Add the following code in the **OleDbProvider** class.

C#

```
public class OleDbProvider: DataProvider {
  public OleDbProvider(string name, string connectionString) {
    Name = name;
    ConnectionString = connectionString;
  }
  public string Name {
    get;
```

```
    private set;
  }
  public string ConnectionString {
    get;
    private set;
  }
  public override bool Supports(string name) {
    return name.StartsWith(Name, StringComparison.OrdinalIgnoreCase);
  }
  public override IEnumerable GetObject(string name, NameValueCollection args) {
    var tableName = name.Substring(Name.Length).TrimStart('\\', '/');
    var selectCommand = string.Format("select * from {0}", tableName);
    var conn = new OleDbConnection(ConnectionString);
    var command = new OleDbCommand(selectCommand) {
      Connection = conn
    };
    conn.Open();
    return command.ExecuteReader(CommandBehavior.CloseConnection);
  }
}
```

4. Configure the connection string for the database file within **Configuration (IAppBuilder app)** method of **Startup** class, as shown below.

C#

```
var connectionString =
ConfigurationManager.ConnectionStrings["Nwind"].ConnectionString;
app.AddDataProvider(new OleDbProvider("Nwind", connectionString));
```

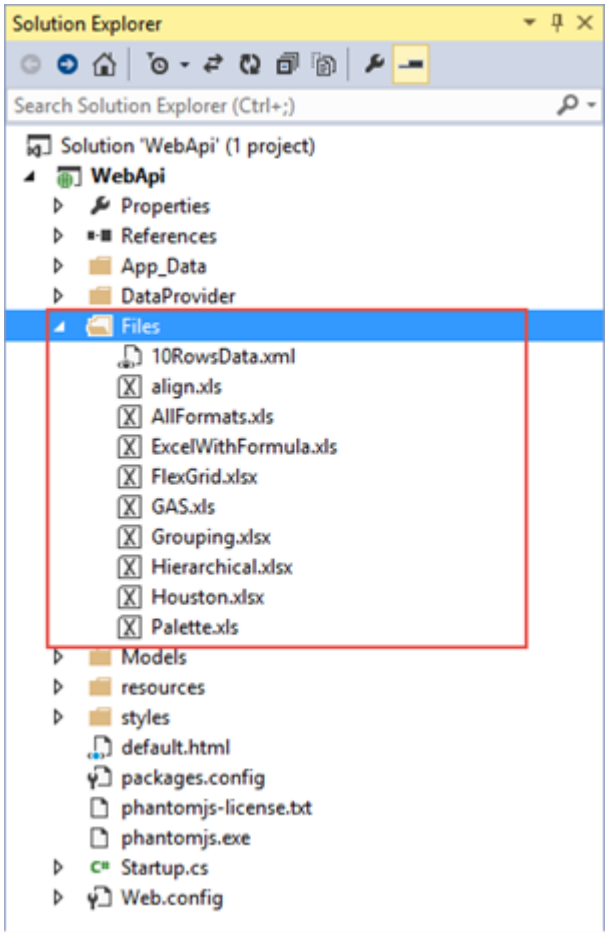
Configuring Storage

Once you have created the Web API service application, you can optionally configure **Storage** in it. Storage serves as a repository for data, on the server side or remote location. It provides data from database or data files to the client applications. The client applications use this data and consume the Rest API service through GET request, to generate, convert, and merge excel files in desired format. For more information, see Excel Services.

Client application fetches data/file from the configured storage once user makes a GET request. Your storage can be a remote or local storage. Remote storage could reside on cloud or on a different server than your configured host service. Whereas, local storage resides on the same server where host service is deployed. This section demonstrates how to create and configure a local storage folder within your service app. While, the following sections discuss configuring server side connection string and .NET collections to configure local storage.

Complete the following steps to configure local storage in Web API service created in Configuring Web API topic.

1. Create a folder named **Files**, in your service application.
2. Add the desired data files to this folder.



3. To configure this storage in your service, call the **GetFullRoot()** method and pass its value to a local variable **folder** in **Configuration (IAppBuilder app)** method within **Startup.cs** file of your Web API service project, as shown below.

C#

```
var folder = GetFullRoot("Files");
```

4. Define the GetFullRoot() method within **Startup** class, as shown below.

C#

```
private static string GetFullRoot(string root)
{
        var applicationBase =
AppDomain.CurrentDomain.SetupInformation.ApplicationBase;
        var fullRoot = Path.GetFullPath(Path.Combine(applicationBase, root));
        if (!fullRoot.EndsWith(Path.DirectorySeparatorChar.ToString(),
        StringComparison.Ordinal))
    {
    // When we do matches in GetFullPath, we want to only match full directory
names.
    fullRoot += Path.DirectorySeparatorChar;
    }
return fullRoot;
}
```

5. Add disk storage to the StorageProviderManager of your service app through AddDiskStorage() method in **Configuration** method within **Startup.cs**, as shown below.

C#

```
 app.UseStorageProviders()
.AddDiskStorage("fullRoot", GetFullRoot("Files"));
```

This method takes root name and full path of the storage. Give a desired name to your storage folder, for example "root" in this case.

Configuring .NET Collections

Data in .NET collections can also be used to generate, merge, and convert excel in a desired format. This section demonstrates how to configure .NET collection in Web API service application. The example implements IEnumerable interface to use data generated in models.

Note: Once you have configured .NET Collections in Web API service application, you can create client side application to call the services to use the .NET Collections. For more information, see Web API Services topic.

1. Create desired model classes within Models folder in your service application. For example, here we create **Sales.cs** and **CustomerOrder.cs** models.

Sales.cs

Example Title

```
public class Sale
    {
        public int ID { get; set; }
        public DateTime Start { get; set; }
        public DateTime End { get; set; }
        public string Country { get; set; }
        public string Product { get; set; }
        public string Color { get; set; }
        public double Amount { get; set; }
        public double Amount2 { get; set; }
        public double Discount { get; set; }
        public bool Active { get; set; }

        public MonthData[] Trends { get; set; }
        public int Rank { get; set; }
        public static IEnumerable <Sale> GetData(int total)
        {
            var countries = new[] { "US", "UK", "Canada", "Japan", "China",
"France", "German", "Italy", "Korea", "Australia" };
            var products = new[] { "Widget", "Gadget", "Doohickey" };
            var colors = new[] { "Black", "White", "Red", "Green", "Blue" };
            var rand = new Random(0);
            var dt = DateTime.Now;
            var list = Enumerable.Range(0, total).Select(i =>
            {
                var country = countries[rand.Next(0, countries.Length - 1)];
                var product = products[rand.Next(0, products.Length - 1)];
```

```
                var color = colors[rand.Next(0, colors.Length - 1)];
                var date = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60,
i % 60);

                return new Sale
                {
                    ID = i + 1,
                    Start = date,
                    End = date,
                    Country = country,
                    Product = product,
                    Color = color,
                    Amount = rand.NextDouble() * 10000 - 5000,
                    Amount2 = rand.NextDouble() * 10000 - 5000,
                    Discount = rand.NextDouble() / 4,
                    Active = (i % 4 == 0),
                    Trends = Enumerable.Range(0, 12).Select(x => new MonthData {
Month = x + 1, Data = rand.Next(0, 100) }).ToArray(),
                    Rank = rand.Next(1, 6)
                };
            });
            return list;
        }

        internal static dynamic GetCountries()
        {
            throw new NotImplementedException();
        }

        internal static dynamic GetProducts()
        {
            throw new NotImplementedException();
        }
    }

    public class MonthData
    {
        public int Month { get; set; }
        public double Data { get; set; }
    }
}
```

CustomerOrder.cs

C#

```
public partial class FlexGridController: Controller {
  public ActionResult Index() {
    var customers = db.Customers.Take(10).ToList();
    var model = customers.Select(c => new CustomerWithOrders {
      CustomerID = c.CustomerID,
```

```
            CompanyName = c.CompanyName,
            Country = c.Country,
            City = c.City,
            Orders = (db.Orders.Where(o => o.CustomerID == c.CustomerID).ToList())
        });
        return View(model);
    }
}
```

2. Configure the models within **Configuration (IAppBuilder app)** method of **Startup** class. This is achieved by adding dataset in the Data Provider Manager of your service application through AddItemsSource() method, as shown below.

C#

```
 app.UseDataProviders().AddItemsSource("Sales", () =>
Sale.GetData(10).ToList()).AddItemsSource("Orders", () =>
CustomerOrder.GetOrderData(20).ToList())
```

Configuring the Client Application

Client Application for Export and Import Services

This section demonstrates how to create a generic client application using MVC and Wijmo 5 controls, which can make a call to the Web API service. You can call Web API service through the client application for enabling export/import functionality. The client uses ComponentOne Web API Client JavaScript file to raise the export/import request for consuming Web API service. For more information on how to work with C1 Web API services, see Services topic.

Adding Control to Client Application

Complete the following steps to create a client application and add FlexGrid control to it.

MVC

1. Create an MVC5 application in Visual Studio.
2. Add a controller (for example **FlexGridController**) in **Controllers** folder. Include the following references.

C# copyCode

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Services_Excel.Models;
```

Note: Replace **Services_Excel** with the name of your application.

3. Replace the method Index() with the following method in the **FlexGridController.cs**.

C# copyCode

```
public ActionResult Index()
{
    return View(Sale.GetData(10));
}
```

4. Add a corresponding view (for example **Index.cshtml**) for the controller. Replace the default code of **Views\FlexGrid\Index.cshtml** with the code given below to initialize the FlexGrid control.

Razor copyCode

```
@(Html.C1().FlexGrid<Sale>().Id("flexGrid").Width("auto")
    .AutoGenerateColumns(false)
    .Bind(bl => bl.Bind(Model))
    .CssClass("grid")
    .IsReadOnly(true)
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ID"));
        bl.Add(cb => cb.Binding("Date"));
        bl.Add(cb => cb.Binding("Country"));
        bl.Add(cb => cb.Binding("Product"));
        bl.Add(cb => cb.Binding("Color"));
        bl.Add(cb => cb.Binding("Amount"));
    })
)
```

5. Add the following HTML code in the <body> tags.

Javascript

```
<button class="btn btn-primary" onclick="exportControlDemoControl()">Export</button>
```

6. Add the following JavaScript code.

Javascript

```
function exportControlDemoControl() {
```

```
        var exporter = new c1.mvc.grid.ExcelExporter(),
            control = wijmo.Control.getControl('#flexGrid');
        exporter.requestExport(control, 'http://demos.componentone.com/aspnet/webapi/api/export/excel', {
            type: xlsx
        });
    }
```

HTML

1. To create a new HTML page in your favorite text editor, add the following code and save the document with a .html extension.

HTML

```
<!DOCTYPE HTML>
<HTML>
<head>
</head>
<body>
</body>
</HTML>
```

2. Add links to the dependencies to your HTML page within the <head> tags.

> Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the Wijmo references within <head> tags of your HTML pages as shown below.

HTML

```
<!-- Wijmo references -->
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.grid.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add the following markup within the <body> tags to create the control.

HTML

```
<div id="flexGrid" style="width:auto"></div> <br/>
```

4. Add the following HTML code in the <body> tags.

Javascript

```
<button class="btn btn-primary" onclick="exportControlDemoControl()">Export</button>
```

5. Add the following JavaScript code.

Javascript

```
function exportControlDemoControl() {
        var exporter = new c1.mvc.grid.ExcelExporter(),
            control = wijmo.Control.getControl('#flexGrid');
        exporter.requestExport(control, 'http://demos.componentone.com/aspnet/webapi/api/export/excel', {
            type: xlsx
        });
    }
```

Adding Data to Client Application

Complete the following steps to populate data in the client application.

MVC

1. In the **Solution Explorer**, right click the folder **Models** and select **Add | Class**. The **Add New Item** dialog appears.
2. In the **Add New Item** dialog, set the name of the class (for example: Sale.cs).
3. Click **Add**.
4. Add the following code to the new model class.

Sale.cs copyCode

```
public class Sale
{
    public int ID { get; set; }
    public DateTime Date { get; set; }
    public string Country { get; set; }
    public string Product { get; set; }
    public string Color { get; set; }
    public double Amount { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "UK", "Canada", "Japan", "China", "France", "Germany", "Italy", "Korea", "Australia" };
    private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget" };

    /// <summary>
    /// Get the data.
    /// </summary>
    /// <param name="total"></param>
    /// <returns></returns>
    public static IEnumerable<Sale> GetData(int total)
    {
        var colors = new[] { "Black", "White", "Red", "Green", "Blue" };
        var rand = new Random(0);
        var dt = DateTime.Now;
        var list = Enumerable.Range(0, total).Select(i =>
        {
            var country = COUNTRIES[rand.Next(0, COUNTRIES.Count - 1)];
            var product = PRODUCTS[rand.Next(0, PRODUCTS.Count - 1)];
            var color = colors[rand.Next(0, colors.Length - 1)];
            var date = new DateTime(dt.Year, i % 12 + 1, 25);

            return new Sale
```

```
            {
                ID = i + 1,
                Date = date,
                Country = country,
                Product = product,
                Color = color,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2)
            };
        });
        return list;
    }

    public static List<string> GetCountries()
    {
        var countries = new List<string>();
        countries.AddRange(COUNTRIES);
        return countries;
    }

    public static List<string> GetProducts()
    {
        List<string> products = new List<string>();
        products.AddRange(PRODUCTS);
        return products;
    }
}
```

5. Save the application.

HTML

1. Within the <head> tag, below the references, add the following script to initialize the grid and generate data.

JavaScript

```
<script type="text/javascript">
$(document).ready(function () {
// create some random data
var countries = 'US,UK,Canada,Japan,China,France,Germany,Italy,Korea,Australia'.split(',');
var products = 'Widget,Gadget'.split(',');
var colors = 'Black,White,Red,Green,Blue'.split(',');
var today = new Date();
var dd = today.getDate();
var yyyy = today.getFullYear();
var numRows = 10;
var data = [];
for (var i = 0; i < numRows; i++) {
var date = (i%12+1)+'/'+dd+'/'+yyyy;
data.push({
id: i + 1,
date: date,
country: countries[Math.floor((Math.random() * 100) + 1) % 10],
product: products[Math.floor((Math.random() * 100) + 1) % 2],
color: colors[Math.floor((Math.random() * 100) + 1) % 5],
amount: Math.random() * 5000
});
}
// create CollectionView on the data (to get events)
var view = new wijmo.collections.CollectionView(data);
// initialize the grid
var grid = new wijmo.grid.FlexGrid('#flexGrid', {
columns: [{
binding: 'id',
header: 'ID'
}, {
binding: 'date',
header: 'Date'
}, {
binding: 'country',
header: 'Country'
}, {
binding: 'product',
header: 'Product'
}, {
binding: 'color',
header: 'Color'
}, {
binding: 'amount',
header: 'Amount'
}],
autoGenerateColumns: false,
itemsSource: view,
selectionMode: wijmo.grid.SelectionMode.Row
});
});
</script>
```

2. Save the application.

Back to Top

Client Application for REST Api Services

This section demonstrates how to create a generic client application, which makes a call to a RESTful Web API service. Web API Edition services based on REST can be consumed by client applications built on various platforms. You can call Web API service through this client application for generating and merging excel files, or generating barcode. For more information on how to work with C1 Web API services, see Services topic.

The examples in the following sections use HTML and WinForms applications to call the REST API services. Therefore, we will create these two applications in this section.

Let us consider, that the client applications built in this section will send a POST request to the endpoint (server) for generating excel files, and the clients that will send a GET request can be created on similar lines.

Adding Controls to Client Application

Complete the following steps to create the client application and add input controls in it.

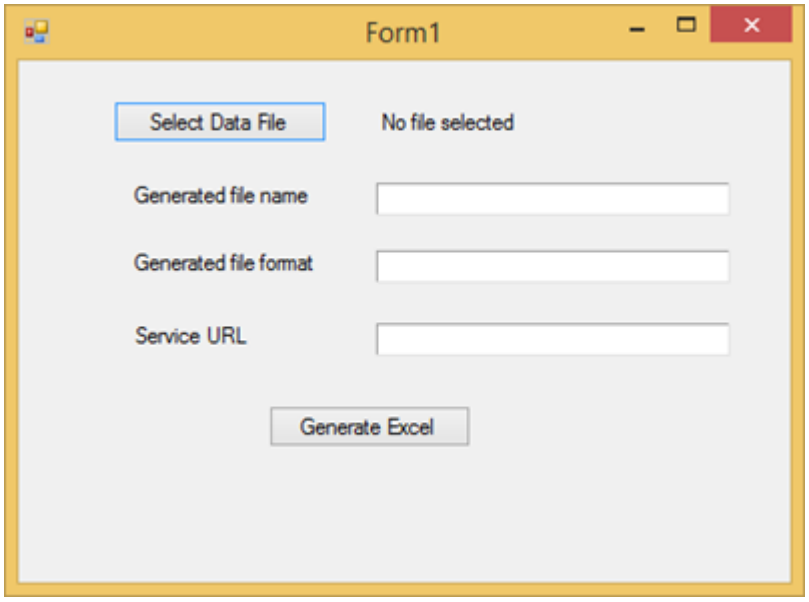
WinForms

1. Create a new WinForms project in Visual Studio. For more information, please refer Creating a .NET Project.
2. Include the following references.

C#

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.ComponentModel;
using System.Windows.Forms;
```

3. From the Toolbox add two C1Button controls, four C1Label controls, and three C1TextBox controls to the form.
4. Set the text properties of these input controls.
5. Click **Build | Build Solution** to build the project. Press **F5** to run the project. Your form will appear as shown below.



6. Press **Shift+F5** to stop debugging your application, and switch to Design view. From Toolbox, drag and drop OpenFileDialog component on the form. This component will appear in Component tray in Design view of your WinForms application.

The **OpenFileDialog** component will be used to display a dialog box for selecting data file, while making a request to Web API service.

HTML

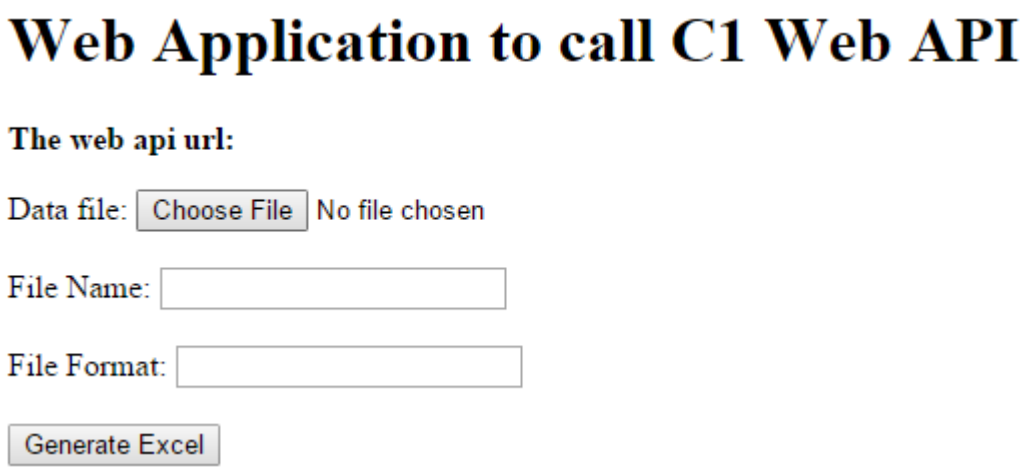
1. To create a new HTML page in your favorite text editor, add the following tags in the text editor and save the document with a .html extension.

```
HTML
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
</body>
</HTML>
```

2. Add the following markup within the <body> tags to create the controls.

```
HTML
<h1>Web Application to call C1 Web API</h1>
<p><b>The web api url:  </b></p>
<form>
        <label for="dataFile">Data file:</label>
        <input type="file" id="dataFile" name="dataFile" accept=""/>
        <br /><br />
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName" value=""/>
        <br /><br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value=""/>
        <br /><br />
        <input type="submit" value="Generate Excel"/>
</form>
```

3. Save the HTML file, and open it in a browser. Your HTML page will appear as shown below, containing the input controls.



Back to Top

Redistributable Files

Web API Edition is developed and published by GrapeCity, inc, you may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate controls. You may also distribute the following redistributable files, free of royalties, with any such application you develop; to the extent that they are used separately on a single CPU on the client/workstation side of the network.

Redistributable Files
For Web API 2.2 C1.Web.Api Nuget package C1.Web.Api.BarCode Nuget package C1.Web.Api.DataEngine Nuget package C1.Web.Api.Document Nuget package C1.Web.Api.Excel Nuget package C1.Web.Api.Image Nuget package C1.Web.Api.Pdf Nuget package C1.Web.Api.Report Nuget package C1.DataEngine NuGet Package C1.Excel NuGet Package C1.FlexReport NuGet Package C1.Document NuGet Package **For ASP.NET Core Web API** C1.AspNetCore.Api NuGet Package C1.AspNetCore.Api.Report NuGet Package C1.AspNetCore.Api.BarCode NuGet Package C1.AspNetCore.Api.Image NuGet Package C1.AspNetCore.Api.Excel NuGet Package C1.AspNetCore.Api.Pdf NuGet Package C1.AspNetCore.Api.Document NuGet Package C1.AspNetCore.Api.DataEngine NuGet Package C1.DataEngine NuGet Package C1.Excel NuGet Package C1.FlexReport NuGet Package C1.Document NuGet Package

Web API Edition Limitations

Following are few limitations of Web API services. Some of these will be eliminated in due course as product is improved.

Report Services

- Report Services does not support Map control.

Image Services

- The client appearance changed by interaction may not be exported as image. For example, FlexPie rotation

after click will not get exported.
- The appearance of controls if affected by theme, in certain cases, cannot be reflected on exported images.
- Data Label export does not support custom itemFormatter JS functions.
- A line series may not be seen on export, when it is overlapping the gridline.

Excel Services

Common

- PivotTable is not supported.
- VBA Macro support is limited, where only opaque copy is supported for xlsx.
- A chart object is not saved whenever an Excel file is loaded and saved, irrespective of the file format.
- Protect Structure/Windows is not supported. For details, please refer to Excel for .NET Limitations.

Import and Export

- Excel import and export having merged cells in it is unsupported in FlexGrid.
- Exported colors in Excel 97-2003 format may look different.
- Style import is not supported in FlexGrid.
- The selected cells (with highlighted fill) will be exported.
- Formats can only be imported to FlexGrid column-wide. The format of the last cell in a column will apply to cells of that whole column. This is limited by FlexGrid control.
- In MVC FlexGrid, custom column headers cannot be exported if disableServerRead is false and onlyCurrentPage is set to false.
- For import the collapsed state of group will be lost, and all the groups are expanded.
- Images in the excel file cannot be exported or imported.
- Icon set in the excel file cannot be exported or imported.

Generate Excel

- On generating excel to json string, and reading it by FlexGrid, the collapsed state of group will be lost and all the groups will be expanded.
- On generating excel file, having image in it, to json or xml string, the image won't be included in the generated string.

Merge Excel

- On merging excel files, having image, to json or xml string, the images won't be included in the merged string.
- Merging excel files with icon set in them, is unsupported.

Deployment

- NuGet cannot work if the Visual Studio option "save new projects on create" is unchecked. This option is unchecked by default in "Visual Basic" environment settings. Make sure to check this option if you use such settings. We strongly recommend that all users choose "Web" environment to develop.

About this Documentation

Acknowledgements

Contact Us

If you have any suggestions or ideas for new features or controls, please call us or write:

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 • USA
1.800.858.2739 | 412.681.4343
412.681.4384 (Fax)

http://www.componentone.com/

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/SuperPages/SupportServices/.

Some methods for obtaining technical support include:

- **Online Resources**
  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support**
  The online support service provides you direct access to our Technical Support staff via an online incident submission system. When you submit an incident, you immediately receive a response via e-mail confirming that the incident is created successfully. This email provides you with an Issue Reference ID. You will receive a response from ComponentOne via e-mail in 2 business days or less.
- **Product Forums**
  ComponentOne's product forums are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers or community engineers will be available on the forums to share insider tips and technique and answer users' questions. Note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
  ComponentOne documentation is available online for viewing. If you have suggestions on how we can improve our documentation, please send a feedback to the Documentation team. Note that the feedback sent to the Documentation team is for documentation related issues only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Security in WebAPI Service

ComponentOne WebAPI services are based on ASP.NET WebAPI Framework. This helps the service application to make use of features that are supported by ASP.NET WebAPI Framework. Security solutions available in WebAPI can be implemented in all the WebAPI service applications.

Web API Services provides different authentication types for users and client applications to authenticate with the data server. You can choose appropriate authentication type for your WebAPI services in order to achieve security level as per your organization standards. For more information about security in WebAPI, see Authentication and Authorization in ASP.NET Web API.

This topic explains security in Web API and how to implement security in Report Services. To apply security to your

Reports Services application, complete the following steps:

Step 1: Configure Report Services

Complete the following steps to configure Report Service Web API and to add custom JavaScript file to access folders.:

- **Create Report Service Web API application**
- **Create files action in controller and view page**

Create Report Service Web API application

Complete the following steps to configure Web API project:

1. Create a new report service application with **Individual User Accounts** using Visual Studio template. For more information about creating report service application, see Report Services using Visual Studio Web API template.

2. Create an authorization process that includes Login, Logout and User Registration. For more information, about authorization process, see Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API.

Back to Top

Create files action in controller and view page

Once you have completed the above two steps, you need add to files.js file to access the folders and reports that are specified in the report provider.

1. From the **Solution Explorer**, right-click the **Scripts** folder, and then add **files.js** Javascript file.
2. Add the following code to access the folders and reports in the report provider.

files.js

```
$(function () {
    setFolder(folder);
});

function folderRefresh() {
    setFolder(folder);
}

function folderUp() {
    folder = folder || '/';
    var names = folder.split('/');
    if (names.length > 0 && !names[names.length - 1]) {
        names.splice(names.length - 1, 1);
    }

    if (names.length > 0) {
        names.splice(names.length - 1, 1);
    }
```

```
    var newFolder = names.join('/');
    setFolder(newFolder);
}

function setFolder(value) {
    value = value || '/';
    if (value[value.length - 1] != '/'){
        value += '/';
    }

    folder = value;
    var $folder = $('#folder');
    $folder.text(value);

    $('#upBtn').prop('disabled', value == '/');

    var $catalogItems = $('#catalogItems');
    $catalogItems.html('Loading');

    wijmo.viewer.ReportViewer.getReports(reportApi, value).then(function (data)
{
        $catalogItems.html('');
        data = data || {};
        var dataItems = data.items || [];
        var items = [];
        for (var i = 0, length = dataItems.length; i < length; i++) {
            var item = dataItems[i];
            if (item.type === wijmo.viewer.CatalogItemType.File) {
                items = items.concat(item.items);
                continue;
            }

            items.push(item);
        }

        for (var i = 0, length = items.length; i < length; i++) {
            var item = items[i], name = item.name, isReport = item.type ===
wijmo.viewer.CatalogItemType.Report;
            if (isReport) {
                var parts = item.path.split('/');
                name = parts[parts.length - 2] + '/' + parts[parts.length - 1];
            }

            var $item = $('<div>')
                .data('catalogItem', item)
                .addClass('catalog-item')
                .on('dblclick', function () {
                    var curItem = $(this);
                    var curData = curItem.data('catalogItem');
                    if (curData.type === wijmo.viewer.CatalogItemType.Report) {
```

```
                        showReport(curData.path);
                        return;
                    }

                    setFolder(curData.path);
                }),
                icon = $('<div>').addClass('glyphicon')
                .addClass(isReport ? 'glyphicon-file' : 'glyphicon-folder-
close'),
                title = $('<div>').addClass('title').text(name);
            $item.append(icon).append(title);
            $catalogItems.append($item);
        }
    }).catch(function (xhr) {
        var message = xhr.status + ': ' + xhr.statusText;
        $catalogItems.html('<div class="error">' + message + '<br/>'
            + 'You don\'t have permission to access this folder. Please contact
your network administrator.</div>' );
    });
}

function showReport(path) {
    var url = reportPageUrl + '?path=' + path;
    gotoUrl(url);
}

function gotoUrl(url) {
    window.location.href = url;
}

var httpRequest = wijmo.httpRequest;
wijmo.httpRequest = function (url, settings) {
    settings = settings || {};
    var token = sessionStorage.getItem(tokenKey);
    if (token) {
        var requestHeaders = settings.requestHeaders || {};
        requestHeaders['Authorization'] = 'Bearer ' + token;
        settings.requestHeaders = requestHeaders;
    }

    httpRequest(url, settings);
};
```

3. Right-click the **Views** folder and add a view (Files.cshtml) to get the folders and reports in the report provider.

Files.cshtml

```
<div class="toolbar">
    <button id="upBtn" class="btn" onclick="folderUp()"><span class="glyphicon
glyphicon-arrow-up"></span><span>Up folder</span></button>
    <button id="refreshBtn" class="btn" onclick="folderRefresh()"><span
class="glyphicon glyphicon-refresh"></span><span>Refresh</span></button>
```

```
    <span class="address">
        <span class="glyphicon glyphicon-folder-open"></span>
        <span>Folder:</span>
        <span id="folder"></span>
    </span>
</div>
<div id="catalogItems" class="catalog-items">
</div>
@section scripts{<script src="~/Scripts/files.js"></script>}
```

Back to Top

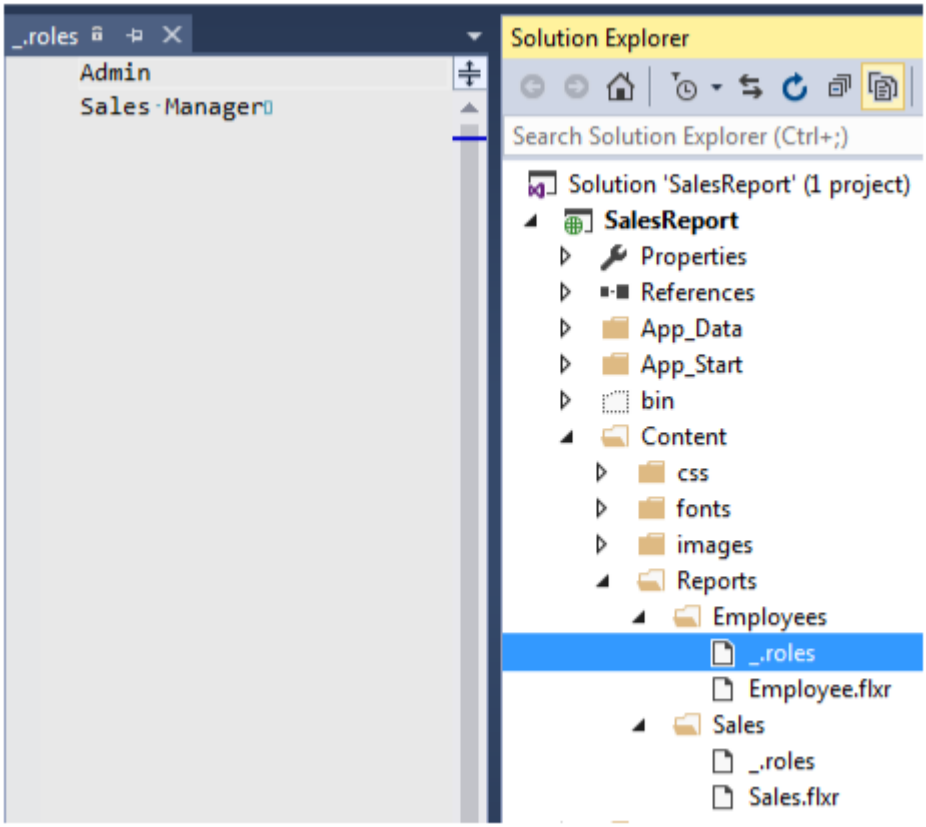
Step 2: Add Custom Authorization

Complete the following steps to create a customized authorization for controlling the file access permissions.

- **Configure folder access permissions**
- **Customize ReportController with the customized authorization attribute**
- **Customize IDirectRouteProvider for route attributes**
- **Resolve conflict of multiple ReportControllers**
- **Register HttpConfiguration in WebApiConfig class**
- **Build and Run the Project**

Configure folder access permission

In your application, you need to add files that define which role can access the folder, as shown in the image below. Once you have added the **_roles** file, you can read the roles information in authorization attribute.



To provide access to the folders based on the role assigned to the user, read the roles information in the authorization attribute. The authorization attribute identifies the user login and provides access to the files based on the role assigned to the users.

1. In the **Solution Explorer**, right click project and select **Add | Class**. The Add New Item dialog appears.
2. In the **Add New Item** dialog, set the name of the class (for example: StorageAuthorizeAttribute.cs).
3. Click **Add**. A new class is added to the application.
4. Add the following code inside StorageAuthorizeAttribute.cs file.

StorageAuthorizeAttribute.cs

```
using C1.Web.Api.Report;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Web.Http.Controllers;
using System.Web.Http.Filters;
using C1.Web.Api.Storage;

namespace SalesReport.Controllers
{
    internal class StorageAuthorizeAttribute : AuthorizationFilterAttribute
    {
        public override void OnAuthorization(HttpActionContext actionContext)
        {
            var principal =
actionContext.ControllerContext.RequestContext.Principal;
            if (principal == null || principal.Identity == null ||
!principal.Identity.IsAuthenticated)
            {
                Unauthorize(actionContext);
                return;
            }

            var values = actionContext.RequestContext.RouteData.Values;
            object pathObj;
            if (!values.TryGetValue("path", out pathObj))
            {
                return;
            }

            var path = (pathObj as string) ?? string.Empty;
            var defaultProvider = ReportProviderManager.Current.Get("") as
FlexReportProvider;
            if (defaultProvider == null)
            {
                return;
            }

            var roles = GetRoles(defaultProvider.Storage, path);
            if(!roles.Any())
```

```
            {
                return;
            }

            if (!roles.Any(r => principal.IsInRole(r)))
            {
                Unauthorize(actionContext);
            }
        }

        private static readonly object _locker = new object();
        private static readonly IDictionary<string, IEnumerable<string>>
_folderRoles =
            new Dictionary<string, IEnumerable<string>>
(StringComparer.OrdinalIgnoreCase);
        private static IEnumerable<string> GetRoles(IStorageProvider storage,
string path)
        {
            string folder = path;
            var fileStorage = storage.GetFileStorage(path);
            if (fileStorage.Exists)
            {
                var pathParts = path.Split('/');
                pathParts = pathParts.Take(pathParts.Length - 1).ToArray();
                folder = string.Join("/", pathParts);
            }

            lock (_locker)
            {
                IEnumerable<string> roles;
                if (_folderRoles.TryGetValue(folder, out roles))
                {
                    return roles;
                }

                var roleList = GetFolderRoles("", storage);
                var folderParts = folder.Split(new[] { '/' },
StringSplitOptions.RemoveEmptyEntries);
                var currentFolder = "";
                foreach (var part in folderParts)
                {
                    currentFolder += part;
                    var current = GetFolderRoles(currentFolder, storage);
                    if(current != null && current.Any())
                    {
                        roleList = current;
                    }

                    currentFolder += "/";
                }
```

```
                return roleList;
            }
        }

        private static IEnumerable<string> GetFolderRoles(string path,
IStorageProvider storage)
        {
            IEnumerable<string> roles;
            if (_folderRoles.TryGetValue(path, out roles))
            {
                return roles;
            }

            var roleList = new List<string>();
            var rolesFile = storage.GetFileStorage(path + "/_.roles");
            if(rolesFile.Exists)
            {
                using (var reader = new StreamReader(rolesFile.Read()))
                {
                    while (!reader.EndOfStream)
                    {
                        var line = reader.ReadLine();
                        if (!string.IsNullOrEmpty(line))
                        {
                            roleList.Add(line);
                        }
                    }
                }
            }

            _folderRoles.Add(path, roleList);
            return roleList;
        }

        private static void Unauthorize(HttpActionContext actionContext)
        {
            actionContext.Response = new
System.Net.Http.HttpResponseMessage(HttpStatusCode.Unauthorized);
        }
    }
}
```

Back to Top

Customize ReportController with the customized authorization attribute

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. Complete the following steps in the **Add Scaffold** dialog:
    1. Select **Empty MVC Controller** template.
    2. Set name of the controller (for example: `ReportController`).
    3. Click **Add**.

4. Add the following code inside ReportController.cs file.

ReportController.cs

```
using System.Web.Http;

namespace SalesReport.Controllers
{
    public class ReportController : C1.Web.Api.Report.ReportController
    {
        [StorageAuthorize]
        public override IHttpActionResult GetCatalogInfo(string path, bool
recursive = false)
        {
            return base.GetCatalogInfo(path, recursive);
        }
    }
}
```

Back to Top

Customize IDirectRouteProvider for route attributes

1. In the **Solution Explorer**, right click project and select **Add | Class**. The Add New Item dialog appears.
2. In the **Add New Item** dialog, set the name of the class (for example: CustomDirectRouteProvider.cs).
3. Click **Add**. A new class is added to the application.
4. Add the following code inside CustomDirectRouteProvider.cs file.

CustomDirectRouteProvider.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http.Controllers;
using System.Web.Http.Routing;

namespace SalesReport
{
    public class CustomDirectRouteProvider : DefaultDirectRouteProvider
    {
        protected override IReadOnlyList<IDirectRouteFactory>
GetActionRouteFactories(HttpActionDescriptor actionDescriptor)
        {
            // inherit route attributes decorated on base class controller's
actions
            return actionDescriptor.GetCustomAttributes<IDirectRouteFactory>
(true);
        }

        protected override string GetRoutePrefix(HttpControllerDescriptor
controllerDescriptor)
        {
            var prefix = base.GetRoutePrefix(controllerDescriptor);
            if (string.IsNullOrEmpty(prefix))
            {
                var prefixAttr =
```

```
controllerDescriptor.GetCustomAttributes<IRoutePrefix>(true).FirstOrDefault();
                if (prefixAttr != null)
                {
                    return prefixAttr.Prefix;
                }
            }

            return prefix;
        }
    }
}
```

Back to Top

Resolve conflict of multiple ReportControllers

There are two ReportControllers in this application. This code adds a customized **IHttpControllerTypeResolver** which helps the client application to identify the required ReportController at the time of execution.

1. In the **Solution Explorer**, right click project and select **Add | Class**. The Add New Item dialog appears.
2. In the **Add New Item** dialog, set the name of the class (for example: ReportsControllerTypeResolver.cs).
3. Click **Add**. A new class is added to the application.
4. Add the following code inside **ReportsControllerTypeResolver.cs** file.

ReportsControllerTypeResolver.cs

```
using System;
using System.Web.Http.Controllers;
using System.Web.Http.Dispatcher;

namespace SalesReport
{
    internal class ReportsControllerTypeResolver :
DefaultHttpControllerTypeResolver
    {
        public ReportsControllerTypeResolver() : base(IsControllerType)
        { }

        private static bool IsControllerType(Type t)
        {
            if (t != null && t.IsClass && (t.IsVisible && !t.IsAbstract) &&
typeof(IHttpController).IsAssignableFrom(t))
                return HasValidControllerName(t) && t !=
typeof(C1.Web.Api.Report.ReportController);
            return false;
        }

        private static bool HasValidControllerName(Type controllerType)
        {
            string str = DefaultHttpControllerSelector.ControllerSuffix;
            if (controllerType.Name.Length > str.Length)
                return controllerType.Name.EndsWith(str,
StringComparison.OrdinalIgnoreCase);
            return false;
```

```
        }
    }
}
```

Register HttpConfiguration in WebApiConfig class

Register the HttpConfiguration in WebApiConfig.cs file. Also, configure the Web API to use only bearer token authentication.

1. In the **Solution Explorer**, right click **AppData** folder and select **Add | Class**. The Add New Item dialog appears.
2. In the **Add New Item** dialog, set the name of the class (for example: WebApiConfig.cs).
3. Click **Add**. A new class is added to the application.
4. Add the following code inside **WebApiConfig.cs** file.

WebApiConfig.cs

```
using System.Web.Http;
using Microsoft.Owin.Security.OAuth;
using System.Web.Http.Dispatcher;

namespace SalesReport
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services
            // Configure Web API to use only bearer token authentication.
            config.SuppressDefaultHostAuthentication();
            config.Filters.Add(new
HostAuthenticationFilter(OAuthDefaults.AuthenticationType));

            // Web API routes
            config.MapHttpAttributeRoutes(new CustomDirectRouteProvider());

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            config.Services.Replace(typeof(IHttpControllerTypeResolver), new
ReportsControllerTypeResolver());
        }
    }
```

Back to Top

Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

Back to Top

Services

To use **Web API Services** you need to first create a service project in Visual Studio, and deploy it on IIS or self-host it. Client applications (MVC, HTML based, desktop applications or other generic applications) can then call these Web API services for exporting and importing excel files, exporting images (to desired format), generating excel from given data and template, and generating barcode from a given text.

ComponentOne Studio Web API Edition provides the following Web API Services.

Report Services
: Learn about how to work with Report Services such as Report List, Load Report, and Export in ComponentOne Studio Web API.

Data Engine Services
: Learn about how to work with Data Engine Services such as Data Source, Raw Data, Analysis, and Unique Value in ComponentOne Studio Web API.

PDF Services
: Learn about how to work with PDF Services such as Export, Supported Formats, and PDF Status in ComponentOne Studio Web API.

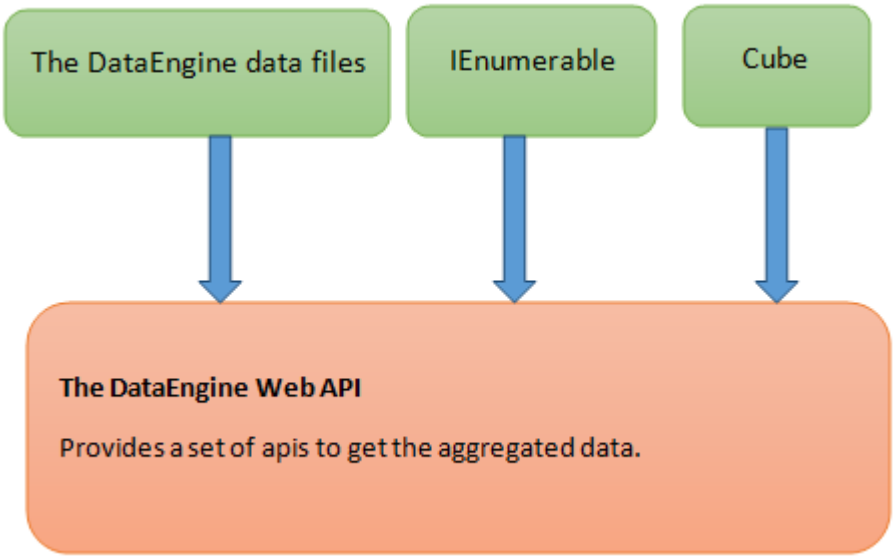
Excel Services
: Learn about how to work with Excel Services such as Export, Import, Generate Excel, and Merge Excel in ComponentOne Studio Web API.

Image Services
: Learn about how to export different controls to image using Image Services in ComponentOne Studio Web API. The service supports image export to PNG, JPG, BMP, TIFF, and GIFF formats.

Barcode Services
: Learn about how to generate scannable C1 supported barcode using Barcode Services in ComponentOne Studio Web API.

Data Engine Services

DataEngine provides a suite of APIs that allow the user to analyze data from multiple data sources, including SQL Server, other SQL based RDMS servers, NoSQL service, web service, structured files from file/network systems, and more. The aggregating data can be consumed by other controls or application.



DataEngine Web API uses a column-oriented data model which is widely used in many open source and commercial analytical databases and libraries. Data Engine can handle up to hundreds of millions records in a fraction of a

second. The aggregated data is fetched from the Web API, the client application is simple, and it only sends the corresponding query to the server with some format to fetch the data. For more information about Data Engine services, see Configuring Data Engine Services.

Configuring Data Engine Web API

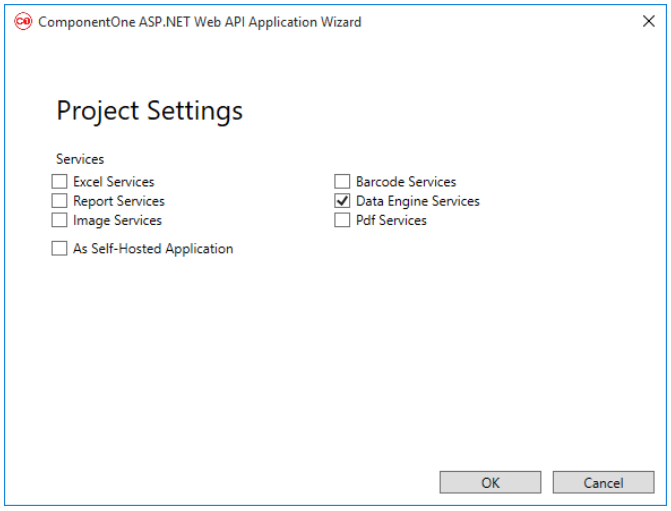
Data Engine WebApi using C1 Template

To add data to the DataEngine service, complete the following steps:

- **Step 1: Create a new WebAPI application**
- **Step 2: Create a new Model**
- **Step 3: Configure Startup.cs**
- **Step 4: Build and Run the Project**

Step1: Create a new WebAPI application

1. In Visual Studio, select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **C1 | Visual C# | Web | C1 Web API Application** to create a new C1 Web API Service application.
3. Set a **Name** and **Location** for your application, and then Click **OK.**
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select **Data engine services** checkbox.



5. Once you have selected the **Services** from the wizard, click **OK** to create a new **C1 Web API Service application**.

Back to Top

Step 2: Create a new Model

Create a new class inside the Models folder to create **Product** data source for the Olap control.

1. Add a new class to the folder **Models** (for example: `ProductData.cs`). For more information about how to add a new model, see Adding controls.
2. Add the following code to the model to define the data for Olap.

C#

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Web;

namespace DataEngineWebpi.Models
{
    public class ProductData
    {
        private static Random r = new Random();

        public int ID { get; set; }
        public string Product { get; set; }
        public string Country { get; set; }
        public DateTime Date { get; set; }
        public int Sales { get; set; }
        public int Downloads { get; set; }
        public bool Active { get; set; }
        public double Discount { get; set; }

        private static int randomInt(int max)
        {
            return (int)Math.Floor(r.NextDouble() * (max + 1));
        }

        public static IEnumerable<ProductData> GetData(int cnt)
        {
            string[] countries = "China,India,Russia,US,Germany,UK,Japan,Italy,Greece,Spain,Portugal".Split(',');
            string[] products = "Wijmo,Aoba,Xuni,Olap".Split(',');
            List<ProductData> result = new List<ProductData>();
            for (var i = 0; i < cnt; i++)
```

```
            {
                result.Add(new ProductData
                {
                    ID = i,
                    Product = products[randomInt(products.Length - 1)],
                    Country = countries[randomInt(countries.Length - 1)],
                    Date = new DateTime(2015, randomInt(5) + 1, randomInt(27) + 1),
                    Sales = randomInt(10000),
                    Downloads = randomInt(10000),
                    Active = randomInt(1) == 1 ? true : false,
                    Discount = r.NextDouble()
                });
            }
            return result;
        }
    }
}
```

Back to Top

Step3: Configure Startup.cs file

In the Startup.cs file, register the data source, that will be later accessed by the client application.

1. From the Solution Explorer, select and open **Startup.cs** file.
2. Replace the code inside **Startup1** class.

Example Title copyCode

```
using System;
using System.Threading.Tasks;
using Microsoft.Owin;
using Owin;
using System.IO;
using C1.DataEngine;
using C1.Web.Api;
using DataEngineWebpi.Models;

[assembly: OwinStartup(typeof(DataEngineWebpi.Startup1))]

namespace DataEngineWebpi
{
    public class Startup1
    {
        private static string DATAPATH = Path.Combine(System.Web.HttpRuntime.AppDomainAppPath, "Data");
        public void Configuration(IAppBuilder app)
        {
            app.UseDataEngineProviders()
                .AddDataEngine("complex10", () =>
                {
                    return ProductData.GetData(100000);
                })
                .AddDataEngine("complex50", () =>
                {
                    return ProductData.GetData(500000);
                })
                .AddDataEngine("complex100", () =>
                {
                    return ProductData.GetData(1000000);
                })
                .AddDataSource("dataset10", () => ProductData.GetData(100000).ToList())
                .AddDataSource("dataset50", () => ProductData.GetData(500000).ToList())
                .AddDataSource("dataset100", () => ProductData.GetData(1000000).ToList())
                .AddCube("cube",
                @"Data Source=http://ssrs.componentone.com/OLAP/msmdpump.dll;Provider=msolap;Initial Catalog=AdventureWorksDW2012Multidimensional",
                    "Adventure Works");
        }
    }
}
```

Once you have added the above code in Startup1.cs, you can register the DataEngine data and the memory data by the extended methods RegisterDataEngine and RegisterDataSet.

> **Note:** If you want your WebAPI server to **support cross domain requests**, you can add the following code in the **Configuration** method of Startup1.cs file.
>
> ```
> app.UseCors(CorsOptions.AllowAll);
> ```

Back to Top

Step 4: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

> Append the URL path (for example: http://localhost:1234/api/dataengine/complex/fields) in the address bar of the browser to see the output.

Back to Top

Data Engine WebApi using Visual Studio Template

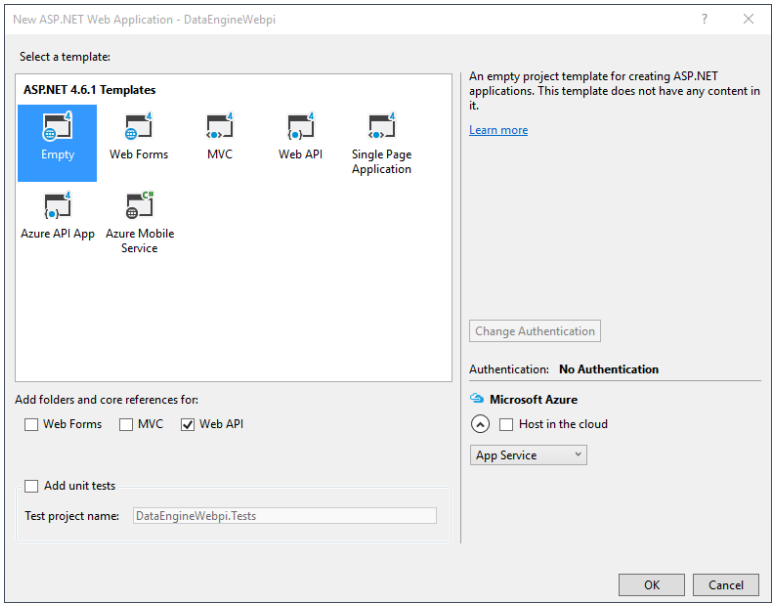
To use DataEngine WebApi using Visual Studio, complete the following steps:

- **Step 1: Create a WebApi Application**
- **Step 2: Install DataEngine Nuget package**
- **Step 3: Create a Model**
- **Step 4: Add Startup file for your application**
- **Step 5: Register data source**
- **Step 6: Configure Web.config**

- **Step 7: Build and Run the Project**

Step 1: Create a new WebApi Application

1. In Visual Studio, Select **File | New | Project.**
2. Under installed templates, select **Visual C# | Web | ASP.NET Web Application (.NET Framework)**. Make sure your application **.NET Framework version is 4.5** or above.
3. Set a **Name** and **Location** for your application. Click **OK** to open New ASP.NET Web Application (.NET Framework) dialog.
4. In the **New ASP.NET Project (.NET Framework)** dialog, select the **Empty** template and check the **Web API** option.

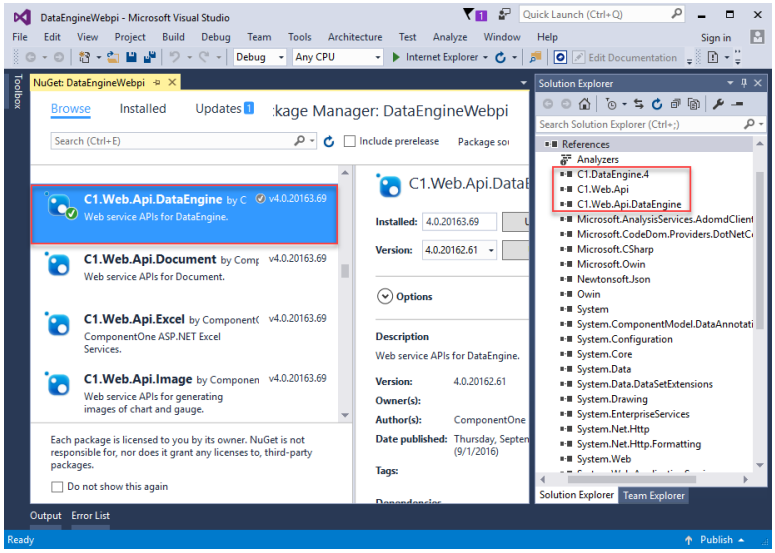


5. Click **OK** to create WebApi Application.

Back to Top

Step 2: Install C1.Web.Api.DataEngine Nuget package

1. Add the C1.Web.Api.DataEngine reference to the project.
2. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**.



3. In **NuGet Package Manager**, select **GrapeCity** as the Package source. Search for **C1.Web.Api.DataEngine** package, and click **Install**.

> **Note:**
> - To execute **Startup1.cs** class alongwith your application, install **"Microsoft.Owin.Host.SystemWeb"** Nuget package in your Visual Studio application.
> - **`"C1.Web.Api.DataEngine"`** gets added under the "dependencies" within project.json file of your project once you restore the packages.

Back to Top

Step 3: Create ProductData.cs Model

Create a new class inside the Models folder to create data source for the Olap control.

1. Add a new class to the folder **Models** (for example: `ProductData.cs`). For more information about how to add a new model, see Adding controls.
2. Add the following code to the model to define the data for Olap.

C#

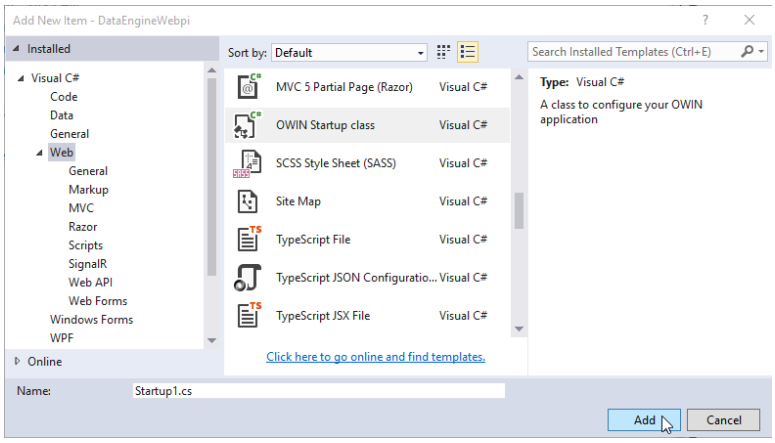
```
using System;
using System.Collections.Generic;
using System.Data;
```

```
using System.Linq;
using System.Web;

namespace DataEngineWebpi.Models
{
    public class ProductData
    {
        private static Random r = new Random();

        public int ID { get; set; }
        public string Product { get; set; }
        public string Country { get; set; }
        public DateTime Date { get; set; }
        public int Sales { get; set; }
        public int Downloads { get; set; }
        public bool Active { get; set; }
        public double Discount { get; set; }

        private static int randomInt(int max)
        {
            return (int)Math.Floor(r.NextDouble() * (max + 1));
        }

        public static IEnumerable<ProductData> GetData(int cnt)
        {
            string[] countries = "China,India,Russia,US,Germany,UK,Japan,Italy,Greece,Spain,Portugal".Split(',');
            string[] products = "Wijmo,Aoba,Xuni,Olap".Split(',');
            List<ProductData> result = new List<ProductData>();
            for (var i = 0; i < cnt; i++)
            {
                result.Add(new ProductData
                {
                    ID = i,
                    Product = products[randomInt(products.Length - 1)],
                    Country = countries[randomInt(countries.Length - 1)],
                    Date = new DateTime(2015, randomInt(5) + 1, randomInt(27) + 1),
                    Sales = randomInt(10000),
                    Downloads = randomInt(10000),
                    Active = randomInt(1) == 1 ? true : false,
                    Discount = r.NextDouble()
                });
            }
            return result;
        }
    }
}
```

Back to Top

Step 4: Add Startup.cs for your application

1. In **Solution Explorer**, select a target project.
2. On the Project menu, click **Add New Item** option.
3. In the Add New Item dialog, select **Web** and then select **OWIN Startup class** template from the list on right.



4. Click **Add**. Startup1.cs file is added to the application.

Step 5: Register data source in Startup1.cs

1. Select and Open **Startup1.cs** file from Solution Explorer.
2. Replace the code inside Startup1 class.

Startup.cs copyCode

```
using System;
using System.Threading.Tasks;
using Microsoft.Owin;
using Owin;
using System.IO;
using C1.DataEngine;
using C1.Web.Api;
using DataEngineWebpi.Models;

[assembly: OwinStartup(typeof(DataEngineWebpi.Startup1))]

namespace DataEngineWebpi
{
    public class Startup1
    {
```

```
        private static string DATAPATH = Path.Combine(System.Web.HttpRuntime.AppDomainAppPath, "Data");
        public void Configuration(IAppBuilder app)
        {
            app.UseDataEngineProviders()
                .AddDataEngine("complex10", () =>
                {
                    return ProductData.GetData(100000);
                })
                .AddDataEngine("complex50", () =>
                {
                    return ProductData.GetData(500000);
                })
                .AddDataEngine("complex100", () =>
                {
                    return ProductData.GetData(1000000);
                })
                .AddDataSource("dataset10", () => ProductData.GetData(100000).ToList())
                .AddDataSource("dataset50", () => ProductData.GetData(500000).ToList())
                .AddDataSource("dataset100", () => ProductData.GetData(1000000).ToList())
                .AddCube("cube",
                @"Data Source=http://ssrs.componentone.com/OLAP/msmdpump.dll;Provider=msolap;Initial Catalog=AdventureWorksDW2012Multidimensional",
                    "Adventure Works");
        }
    }
}
```

Once you have added the above code in Startup1.cs, you can register the DataEngine data and the memory data by the extended methods RegisterDataEngine and RegisterDataSet.

> **Note:** If you want your WebAPI server to **support cross domain requests**, you can add the following code in the **Configuration** method of Startup1.cs file.
>
> *app.UseCors(CorsOptions.AllowAll);*

Back to Top

Step 6: Configure Web.config

Open the **Web.config** file and ensure that the **WebDAVModule** and the **WebDAV** handler are removed from <system.webServer> as shown below:

Web.config

```
<system.webServer>
<modules>
      <remove name="WebDAVModule" />
    </modules>
<handlers>
      <remove name="WebDAV" />
      .........
    </handlers>
</system.webServer>
```

Back to Top

Step 7: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

> Append the URL path (for example: http://localhost:1234/api/dataengine/complex/fields) in the address bar of the browser to see the output.

Back to Top

Data Engine SSAS Service

Complete the following steps to add data to Data Engine using SSAS service:

- **Step1: Create a new WebAPI application**
- **Step 2: Configure Startup.cs**
- **Step 3: Build and Run the Project**

Step1: Create a new WebAPI application

1. In Visual Studio, select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **C1 | Visual C# | Web | C1 Web API Application** to create a new C1 Web API Service application.
3. Set a **Name** and **Location** for your application, and then Click **OK.**
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select **Data engine services** option.
5. Once you have selected the **Services** from the wizard, click **OK** to create a new **C1 Web API Service application**. In this new project, ComponentOne template adds the references C1.Web.Api.dll, C1.DataEngine.dll, and C1.WebApi.DataEngine.dll.

Configure Startup.cs file

1. From the Solution Explorer, select and open **Startup.cs** file. Here we are setting up the SSAS service URL to access the cube data.
2. In the **Startup.cs** file, add the following code inside the Startup class. In this step we are configuring the SSAS service URL for the application.

   The following table describes the SSAS connection string.

Property	Description	Example
Data Source	Specifies the server instance. This property is required for all connections. Valid values include the network name or IP address of the server, local or localhost for local connections, or the name of a local cube (.cub) file.	Data source=AW-SRV01.corp.Adventure-Works.com
Initial Catalog or Catalog	Specifies the name of the Analysis Services database to connect to. The database must be deployed on Analysis Services, and you must have permission to connect to it.	Initial catalog=AdventureWorks2016
Provider	Valid values include MSOLAP.<version>, where <version> is either 4, 5, 6 or 7.	Provider=MSOLAP.7 is used for connections that require the SQL Server 2016

Startup.cs

```
public class Startup
    {
        private readonly HttpConfiguration config =
GlobalConfiguration.Configuration;
        public void Configuration(IAppBuilder app)
        {
               app.UseDataEngineProviders()
               .AddCube("cube", @"Data
Source=http://ssrs.componentone.com/OLAP/msmdpump.dll;Provider=msolap;Initial
Catalog=AdventureWorksDW2012Multidimensional", "Adventure Works");

            // CORS supports
           app.UseCors(CorsOptions.AllowAll);

            // Web Api
            RegisterRoutes(config);
            app.UseWebApi(config);
        }
```

Step 3: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

> Append the URL path (for example: http://localhost:1234/api/dataengine) in the address bar of the browser to see the output.

Back to Top

DataEngine Client Application

Consuming Data Engine Service

This section demonstrates how to call the Web API service through a client application and fetch the data that is stored in Data Engine services. The client sends a GET request to the service, which returns a response stream. The response stream is then saved in the JSON format.

Get Fields

In the following example, we use "**complex**" as an example. It is the registered key for the DataEngine data source. The service URL takes in the Data Source name, location of dataset or collection, and the corresponding fields that are associated with the dataset.

Call the Service

Use the following javascript to call the Web API service, and fetch data from the service application.

Index.html

```
<script>
    $('#action-button').click(function() {
        $.ajax({
            type: 'GET',
            url: 'http://localhost:7383/api/dataengine/complex/fields',
            cache: false,
            dataType: 'json',
            processData: false,
            success: function (data, tx, xhr) {
                // the field list and total row count will be obtained
from result.data.
                // the data format is like the following
                        /*{
                        "data": [
                                {"binding":"Active","dataType":3},
                                {"binding":"Country","dataType":1},
                                {"binding":"Date","dataType":4},
                                {"binding":"Discount","dataType":2},
                                {"binding":"Downloads","dataType":2},
                                {"binding":"ID","dataType":2},
                                {"binding":"Product","dataType":1},
                                {"binding":"Sales","dataType":2}
                                ]
                         }*/
      }
    })
});
</script>
```

Back to Top

Aggregated Data

This section demonstrates how to call the Web API service through a client application and fetch the aggregated data for the sales of the products in the countries. The client sends a POST Ajax request to the service, which returns a response stream. The response stream is then saved in the JSON format.

In the following example, we use "**complex**" as an example. It is the registered key for the DataEngine data source. The service URL takes in the Data Source name, location of dataset or collection, and the corresponding fields that are associated with the dataset.

Call the Service

Use the following javascript to call the Web API service to create an analysis from the service application.

Index.html

```
<script>
$.ajax({

      type: 'POST',

      url: 'http://server/api/dataengine/complex/analyses',

      data: {

        view: '{"fields":[{"binding":"Active","dataType":3},
{"binding":"Country","dataType":1},{"binding":"Date","dataType":4},

        {
          "binding": "Discount",
          "dataType": 2
        },
        {
          "binding": "Downloads",
          "dataType": 2
        },
        {
          "binding": "ID",
          "dataType": 2
        },

        {
          "binding": "Product",
          "dataType": 1
        },
        {
          "binding": "Sales",
          "dataType": 2
        }],

      rowFields: {
```

```
        items: ["Product"]
      },

      columnFields: {
        items: ["Country"]
      },

      valueFields: {
        items: ["Sales"]
      }
    }
    '

  },

  dataType: 'json',

  cache: false,

  processData: false,

  success: function(data, status, xhr) {

    // the following information will be obtained from data.

    /*{

    "data": {

    "token":"93f00724-3877-41ea-80ff-bbeae96f5e36",

    "status":{

    “executingStatus”: ”Executing”,

    “progress”: // the analysis progress

    },

    “result”: //the result data,

    }

    }*/

  }

});
</script>
```

Back to Top

Manage Data Sources

DataEngine services provide API to analyze the data from multiple data sources, including SQL server, SSAS server and structured files from the file system. This topic will help you determine a suitable method to setup the DataEngine in your application. You can use the following methods to configure DataEngine, according to your project requirements.

> **Note**: In order to specify the data source easily in your Web API application, you can map the data source settings into keys. Make sure the key is unique for each data source settings in your application.

AddDataEngine Methods

When you are working with complex or large data in your application, we suggest you to use `AddDataEngine` and `AddDataEngine<T>` methods. These methods generate memory mapped files to save the fields and result data, which makes them the fastest methods for large data processing. For more information about DataEngine methods, see Data Engine WebApi using C1 Template.

Method	Description	Elements
AddDataEngine(string name, DbCommand command, DbConnection connection = null, string workspace = null, string tableName = null)	Adds a DataEngine data.	• **name**: Unique value to describe the data source. • **command**: ADO.NET command object for retrieving data from the database. If connection is null, the user should make sure the connection in command is open before calling this method. • **connection**: An open ADO.NET connection object for connecting to the database or set to null. • **workspace**: Path in the server's file system where DataEngine data is saved in files. If it is set to null or not set, the default workspace is used. • **tableName**: The name of the DataEngine table. If it is set to null or not set, the value of the name parameter is used.
AddDataEngine<T> (string name, Func<IEnumerable<T> dataGetter, string workspace = null, string tableName = null)	Adds DataEngine data from an arbitrary IEnumerable.	• **name**: Unique value to describe the data source. • **dataGetter**: Function to get the arbitrary IEnumerable. • **workspace**: Path in the server's file system where DataEngine data is saved in files. If it is set to null or not set, the default workspace is used. • **tableName**: The name of the DataEngine table. If it is set to null or not set, the value of the name parameter is used.
AddDataEngine(string name, string workspace, string tableName)	Adds a DataEngine data which data files already exist.	• **name**: Unique value used to describe the data source. • **workspace**: Path in the server's file system where DataEngine data is saved in files. If it is set to null or not set, the default workspace is used. • **tableName**: The name of the DataEngine table. If it is set to null or not set, the value of the name parameter is used.

AddDataSource Methods

When you are working with data sources and do not wish to generate the data files on the API server, we suggest you to use `AddDataSource` and `AddDataSource<T>` methods. These methods do not generate any cache files to save the data, therefore the aggregated result data is recalculated for every analysis. For more information about DataSource methods, see Data Engine WebApi using C1 Template.

Method	Description	Elements
AddDataSource(string name, Func<IEnumerable> dataSourceGetter)	Adds an in-memory data using a function.	• **name**: Unique value to describe the data source. • **datasourceGetter**: Function used to return the in-memory data.
AddDataSource(string name, IEnumerable datasource)	Adds an in-memory data.	• **name**: Unique value to describe the data source. • **datasource**: The in-memory data.

AddCube Method

When you are working with cube data from any SQL Server Analysis Service, we suggest you to use the `AddCube` method. This method does not generate any data files in your application, therefore for any analysis, a MDX query is sent to the server and the aggregated data is returned back from the server. Performance of this method mostly depends on the SSAS server. For more information about AddCube method, see Data Engine WebApi using SSAS Service.

Method	Description	Elements
AddCube(string name, string connectionString, string cubeName)	Adds a cube data.	• **name**: Unique value to describe the data source. • **connectionString**: String that specifies information about a data source and the means of connecting to it. • **cubeName**: Name of the cube data source.

Data Source Customization

While working with DataEngine in your Web API application, you might want to customize the data source properties. DataEngine provides the following ways to customize your data source.

Customize the maximum value

User can customize the maximum value count of the aggregated result data. Once the aggregated data is generated, user can use the SetDataEngineAnalysisResultMaxCount property to specify whether the aggregated result data is valid for serialization. In case the count exceeds the maximum value, an exception will be thrown.

The user can increase the maximum value or change the view definition to solve the exception. There is no limitation by default.

Example Code
`IAppBuilder.SetDataEngineAnalysisResultMaxCount(int count)`

Set the default workspace path

In DataEngine services, the default workspace path is “application base path + /data”. The user can use the following method to customize the default workspace path:

Example Code
`IAppBuilder.SetDefaultWorkspacePath (string path)`

Fields

Fields service provides API to get all the fields from the database. You can access this service from your client application to view data from many different data sources that are hosted on Data Engine service.

Fields Service Request Schema

To view data from a specific data source and get the fields information, you need to use GET method. Data source information specified in the request URL, as:

GET:`http://<host>[:port]/api/dataengine/{datasourcekey}/fields`

Data source information can be specified in the request URL, as follows:

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/dataengine/complex10/fields
Status Code: 200
```

URL Parameters

Field service URL accepts **datasourcekey** as paramter. You need to specify the data source key information to access the data. Data source key is used to specify the data source from where you want to get the field information. For more information on Data Engine Services, see WebApiExplorer demo.

Response Messages

HTTP Status Code	Reason
200	A field array is returned.
404	{datasource} is invalid.

Raw Data

Raw Data service provides API to access the raw data of the data source. This service is not supported when you are connected to cube data. Your client application sends an HTTP request to the service application for using the Raw Data API to get the raw data of the data source.

Raw Data Service Request Schema

To view raw data from a data source, you need to use GET method. Data source information specified in the request URL, as:

GET: http://<host>[:port]/api/dataengine/{datasourcekey}/?[skip=n&top=m]

Data source information can be specified in the request URL, as follows

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/dataengine/complex10?skip=1&top=10
Status Code: 200
```

URL Parameters

Raw Data service include following parameters.

Parameter	Description
datasourcekey	Used to specify the data source from where you want to get the field information.
Skip	A positive integer value to specify the start index of the whole raw data. If not set, then returns the entries from the first one.
Top	A positive integer value to specify the count of the returned raw data. If not set, then returns all entries starting with the skip value.

Response Messages

HTTP Status Code	Reason
200	An IRawData object is returned to specify the raw data result information. It includes two items: an array for the raw data and a number value for the total count of all the raw data without paging.
404	{datasource} is invalid.

For more information on Data Engine Services, see WebApiExplorer demo.

Analyses

Analyses service provides API to create an analysis according to the specified data source key. Your client application sends an HTTP request to the service application for using the Analysis API to create an analysis with data source.

Analysis Service Request Schema

To create an analysis with data source, you need to use POST method. Data source information specified in the request URL, as:

POST: http://<host>[:port]/api/dataengine/{datasourcekey}/analyses

Data source information can be specified in the request URL, as follows

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/dataengine/complex100/analyses
Status Code: 201
```

URL Parameters

Analysis service includes the following parameters.

Parameter	Description
datasourcekey	Used to specify the data source you want to analyze.

View Definition	Used to specify the view definition.
	For example:
	{
	fields:[
	{"binding":"Active","dataType":3},
	{"binding":"Country","dataType":1},
	{"binding":"Date","dataType":4},
	{"binding":"Discount","dataType":2},
	{"binding":"Downloads","dataType":2},
	{"binding":"ID","dataType":2},
	{"binding":"Product","dataType":1},
	{"binding":"Sales","dataType":2}
	],
	rowFields:{items:["Product"]},
	columnFields:{items:["Country"]},
	valueFields:{items:["Sales"]}
	}

Response Messages

HTTP Status Code	Reason
201	An IAnalysis object is returned to specify the analysis result information. IAnalysis object includes the status and the result data. When the executingStatus is “Completed”, the result data is an array which stands for the aggregated result data. Otherwise, the result data is always null. Location in response header shows the url to access the analysis instance.
404	{datasource} is invalid.

For more information on Data Engine Services, see WebApiExplorer demo.

Detail Data

Detail Data service provides API to get a list of objects in the raw data source that define the content of a specific cell in the output. This service is not supported when you are connected to cube data. Your client application sends an HTTP request to the service application for using the Detail Data API to get the detail data of the specified cell.

Detail Data Service Request Schema

To view detail data from a data source, you need to use POST method. Data source information specified in the request URL, as:

POST: `http://<host>[:port]/api/dataengine/{datasourcekey}`

Data source information can be specified in the request URL, as follows

```
Request URL: http://testdemos.componentone.com/aspnet/webapi/api/dataengine/complex10
Status Code: 200
```

URL Parameters

Raw Data service includes the following parameters.

Parameter	Description
datasourcekey	Used to specify the data source from where you want to get the field information.

View Definition	Used to specify the view definition. Type is IViewDefinition. For example:
	{
	fields:[
	{"binding":"Active","dataType":3},
	{"binding":"Country","dataType":1},
	{"binding":"Date","dataType":4},
	{"binding":"Discount","dataType":2},
	{"binding":"Downloads","dataType":2},
	{"binding":"ID","dataType":2},
	{"binding":"Product","dataType":1},
	{"binding":"Sales","dataType":2}
	],
	rowFields:{items:["Product"]},
	columnFields:{
	
	items:["Country"]},
	valueFields:{items:["Sales"]}
	}

Key	Used to specify the cell using the corresponding value. It is an array object. Firstly, you need add the values of the fields in rowFields in order into keys. Then add the values of the fields in columnFields in order. If some field value is null, JUST add null into array and DO NOT remove it. You should keep the count of the keys array same as the count of the fields in rowFields and columnFields. For example: ["Aoba", "China"]
Skip	A positive integer value to specify the start index of the whole raw data. If not set, then returns the entries from the first one.
Top	A positive integer value to specify the count of the returned raw data. If not set, then returns all entries starting with the skip value.

Response Messages

HTTP Status Code	Reason
200	An IRawData object is returned to specify the detail result information. It includes two parts: an array for the raw data of the cell and the total count of the raw data without paging.

404	{datasource} is invalid.

For more information on Data Engine Services, see WebApiExplorer demo.

Unique Values

Unique value service provides API to get the unique values of a field specified by`{fieldname}`. This service is not supported when you are connected to cube data. Your client application sends an HTTP request to the service application for using the Detail Data API to get the unique values of a field.

Unique Value Service Request Schema

To get the unique values of a field specified by {fieldname}, you need to use POST method. Data source information specified in the request URL, as:

POST: `http://<host>[:port]/api/dataengine/{datasourcekey}/fields/{fieldname}/uniquevalues`

Data source information can be specified in the request URL, as follows

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/dataengine/complex10/fields/Product/uniquevalues
Status Code: 200
```

URL Parameters

Unique value service includes the following parameters.

Parameter	Description
datasourcekey	Used to specify the data source from where you want to get the field information.
fieldname	Used to specify the field by the header which you want to get the unique values.

View Definition	Used to specify the view definition. For example:
	{
	fields:[
	{"binding":"Active","dataType":3},
	{"binding":"Country","dataType":1},
	{"binding":"Date","dataType":4},
	{"binding":"Discount","dataType":2},
	{"binding":"Downloads","dataType":2},
	{"binding":"ID","dataType":2},
	{"binding":"Product","dataType":1},
	{"binding":"Sales","dataType":2}
	],
	rowFields:{items:["Product"]},
	columnFields:{items:["Country"]},
	valueFields:{items:["Sales"]}
	}

Response Messages

HTTP Status Code	Reason
200	An array is returned to specify the unique values of a field.

404	{datasource} is invalid.

For more information on Data Engine Services, see WebApiExplorer demo.

Status

Status service provides API to get the status of the specified token. Your client application sends an HTTP request to the service application for using the Status API to get the status of the specified token.

Status Service Request Schema

To get the status of the specified token, you need to use GET method. Token information can be specified in the request URL, as:

GET: `http://<host>[:port]/api/dataengine/{datasourcekey}/analyses/{token}/status`

URL Parameters

Parameter	Description
datasourcekey	Used to specify the data source from where you want to get the token information.
token	Used to specify the analysis instance. Token value can be obtained from the 3rd service.

Response Messages

HTTP Status Code	Reason
200	An IStatus object is returned to specify the status. When the status is "Completed", it means the query has been executed successfully and data is prepared for the output. Once the Status API is returned successfully, you can get the data by using the **Result Data** service. If the status is still “Executing”, you can send this request repeatedly until the status is changed to other values: Completed, Cleared or Exception. If the status is “Exception”, it means some error occurs during the executing. For example, the token provided in this request doesn’t exist, an exception would be returned as response.
404	{datasource} or {token} is invalid.

Analysis Service Request Schema

Analysis service provides API to calculate the analysis data according to the specified token. Your client application sends an HTTP request to the service application for using the Analysis API.

To get the analysis data by the specified token, you need to use GET method. Token information can be specified in the request URL, as:

GET: `http://<host>[:port]/api/dataengine/{datasourcekey}/analyses/{token}`

Response Messages

HTTP Status Code	Reason
200	An IAnalysis object is returned to specify the analysis result information.
404	{datasource} or {token} is invalid.

Result Data Request Schema

Result Data service provides API to get the result data by the specified token. Your client application sends an HTTP request to the service application for using the Result Data API.

To get the result data by the specified token, you need to use GET method. Token information can be specified in the request URL, as:

GET: `http://<host>[:port]/api/dataengine/{datasourcekey}/analyses/{token}/result`

Response Messages

HTTP Status Code	Reason
200	An array object is returned to specify the result data. After calling this service to obtain the data prepared, the token will be removed. You cannot use it in other requests then. Otherwise, an exception would be returned.
404	{datasource} or {token} is invalid.

Clear Token Service Request Schema

Clear Token service provides API to clear the token when it is not needed. Your client application sends an HTTP request to the service application for using the Clear Token API to clear the token when it is not needed.

To get the result data by the specified token, you need to use DELETE method. Token information can be specified in the request URL, as:

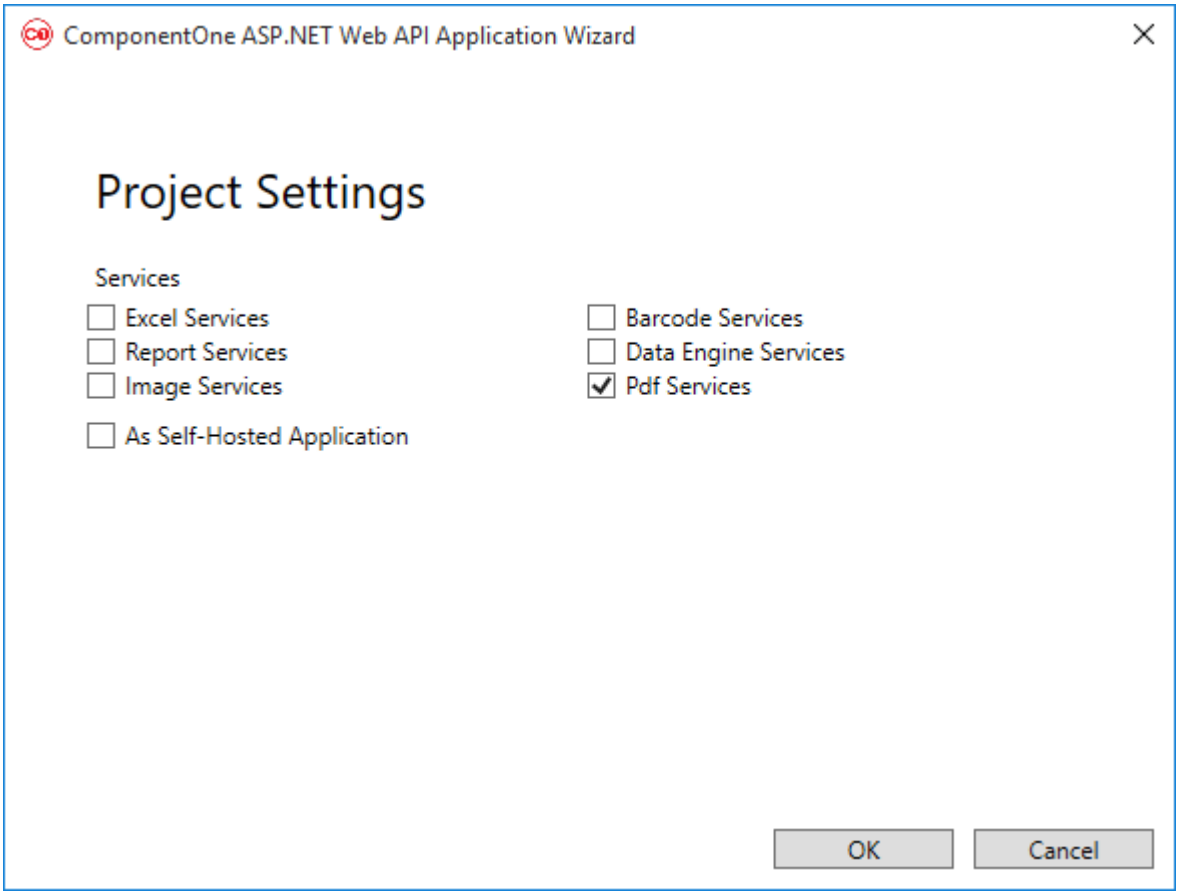
DELETE: `http://<host>[:port]/api/dataengine/{datasourcekey}/analyses/{token}`

Response Messages

HTTP Status Code	Reason
204	No content for response.
404	{datasource} or {token} is invalid.

PDFDocument Services

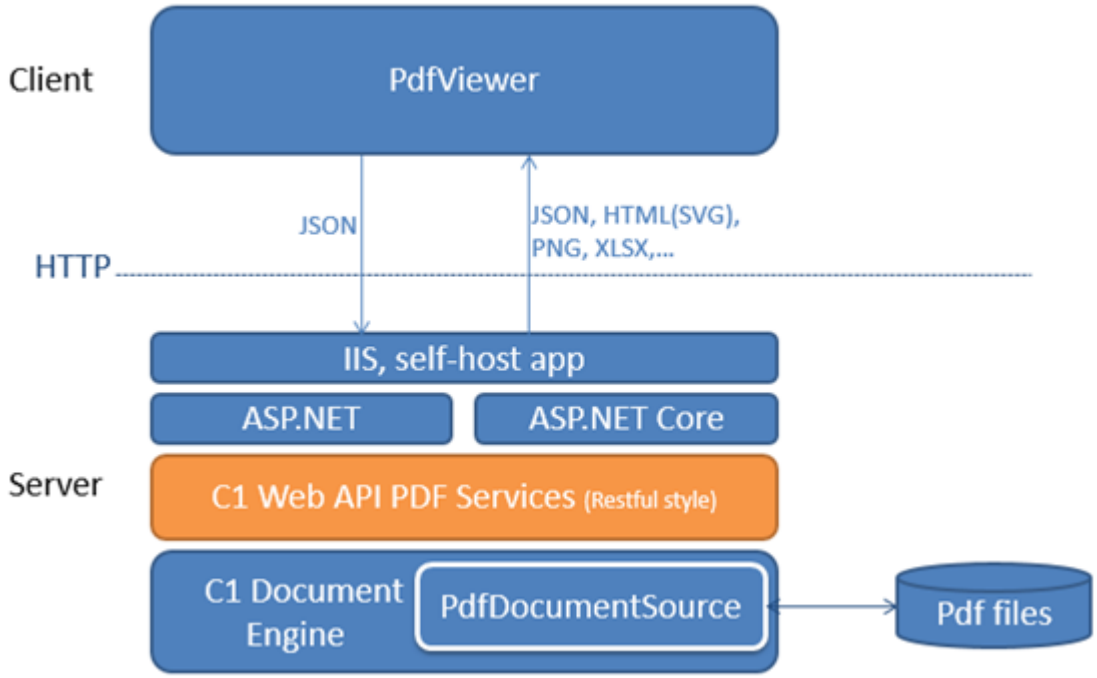
Web API Studio Edition introduces PDFDocument Services that enables you to build HTTP services, which can be consumed by a variety of clients for viewing, loading and caching PDF documents. These are REST based API services, which communicate with HTML 5 PDFViewer control to display the PDF documents on the web.



Available as Visual Studio template, C1 Web API enables you to create PDFDocument service on Visual Studio. Client applications send a request to the PDFDocument service applications to load or export PDF files. The service supports exporting your reports to Word, Excel, RTF, OpenXML, and Image. For more information on PDF Services, see WebApiExplorer demo.

Configure PDF Web API Service

C1 Web API PDFDocument Services enable you to build HTTP services, which can be consumed by a variety of clients for viewing, loading and caching PDF files. These are REST based API services, which communicate with the HTML 5 PDFViewer control to display the pdf file content on the web.

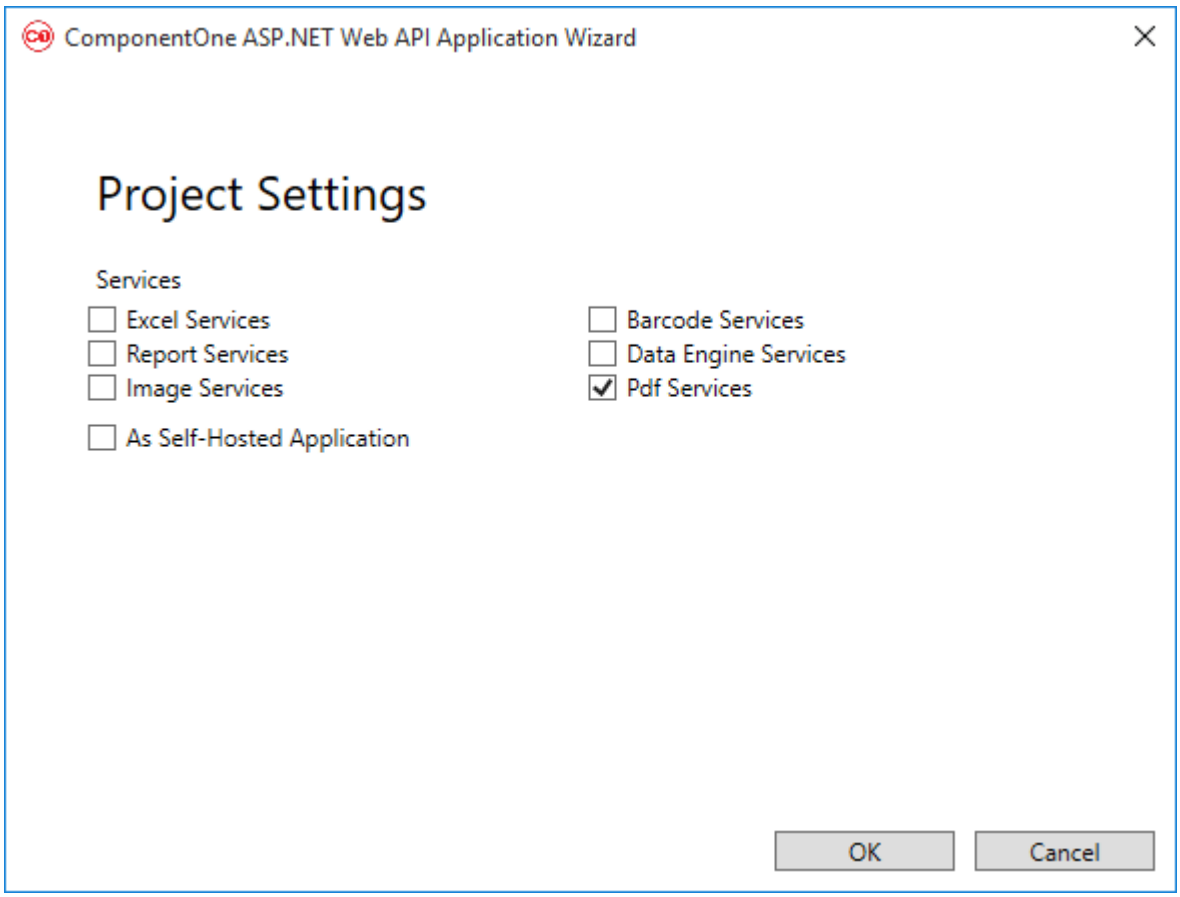


Complete the following steps to setup PDF services.

- **Step1: Create a new WebAPI application**
- **Step 2: Configure Startup.cs**
- **Step 3: Build and Run the Project**

Step1: Create a new WebAPI application

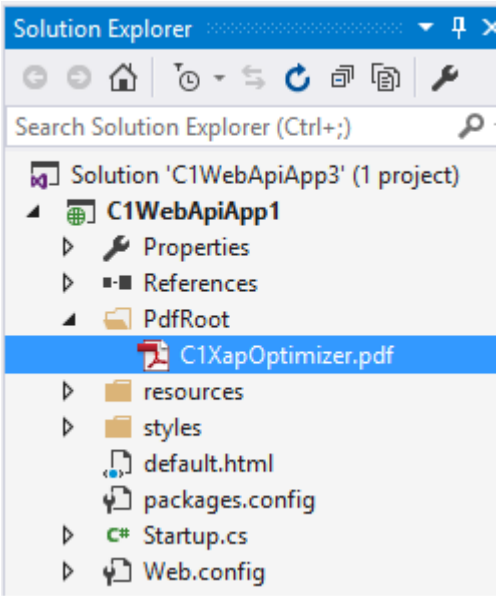
1. In Visual Studio, select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **Visual C# | Web | C1 Web API Application** to create a new C1 Web API Service application.
3. Set a **Name** and **Location** for your application, and then Click **OK.**
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select **Pdf services** option.



5. Once you have selected the **Services** from the wizard, click **OK** to create a new **C1 Web API Service application**.

Configure Startup.cs file

1. Create a folder named **PdfRoot**, in your service application.
2. Add the desired **PDF files** to this folder.



3. From the Solution Explorer, select and open **Startup.cs** file.
4. In the **Startup.cs** file, add disk storage in **Configure** method of **Startup**.

Startup.cs

```
app.UseStorageProviders()
.AddDiskStorage ("PdfRoot",
System.IO.Path.Combine(System.Web.HttpRuntime.AppDomainAppPath, "PdfRoot"));
```

Step 3: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

> Append the URL path (for example: http://localhost:1234/api/pdf/) in the address bar of the browser to see the output.

Back to Top

Info

PDF Info service gets the information related to the PDF file, with the specified PDF path. The PDF document will be cached. The client application sends an HTTP request to the service application for using PDF Info WebAPI.

PDF Info Service Request Schema

To obtain the information related to a specific PDF file using the PDF path, you need to use GET method. PDF file path is specified in the request URL as:

GET: `http://<host>[:port]/api/pdf/{pdfpath}/$pdf`

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/pdf/PdfRoot/C1XapOptimizer.pdf/$pdf
Status Code: 200
```

pdfpath - The pdf file that storage manager can recognize.

Response Messages

HTTP Status Code	Reason
200	Json string contains IExecutionInfo type object.
404	The pdf path does not exist.

Export

PDF Export services enables you to export your PDF documents to different file formats. The PDF document layout and style along with the data gets exported to the specified format.

PDF Export service uses Export API to enable end users to render and export a specified PDF document to a desired format. You can provide the PDF file path, exportFileName and exportOptions in the service URL. The client application sends an HTTP request to the service application for using Export API to export the PDF document to the supported formats.

[GET] Export Service Request Schema

To get the supported export formats for a PDF document through client, you need to use GET method. PDF document path and PDF document name are specified in the request URL. Your client application sends an HTTP request message to the service, as:

```
GET: http://<host>[:port]/api/pdf/{pdfpath}/$pdf/export?[exportFileName]&
[exportoptions]
```

pdfpath - The full path of PDF file.

exportFileName - The exported file name.

exportOptions - IExportOptions

URL Parameters

The GET request URL for PDF Export service accepts PDF document **pdfpath**, **file name**, and **exportOptions** (supported formats) as parameters.

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The pdf path does not exist.
406	The format name is not acceptable.

[POST] Export Service Request Schema

To render and export the PDF file to the specified export filter with options, you need to use POST method. PDF file path is specified in the request URL. Your client application sends an HTTP request message to the service, as:

```
POST: http://<host>[:port]/api/pdf/{pdfpath}/$pdf/export
```

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The pdf path does not exist.
406	The format name is not acceptable.

For more information on PDF Services, see WebApiExplorer demo.

Supported Formats

SupportedFormats API enables the end users to get the supported export formats for a PDF document. Your client application sends an HTTP request to the service application for using GetPdfSupportedFormats API to obtain the information about the supported export formats for a PDF document.

Supported Formats Service Request Schema

To get the supported export formats for a PDF document through client, you need to use GET method. PDF document file path is specified in the request URL, as:

```
GET: http://<host>[:port]/api/pdf/{pdfpath}/$pdf/supportedformats
```

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/pdf/PdfRoot/C1XapOptimizer.pdf/$pdf/supportedformats
Status Code: 200
```

pdfpath - The full path of the pdf file.

URL Parameters

Supported formats service URL accepts pdf file path as parameter.

Response Messages

HTTP Status Code	Reason
200	Json string contains a collection of IExportDescription.
404	The pdf path does not exist.

For more information on PDF Services, see WebApiExplorer demo.

Features

PDFDocument service provides different features that are supported with the PDF file format. Your client application sends an HTTP request to the service application for using Features API to obtain the features supported by the PDF document.

PDF Features Service Request Schema

To get the features that are supported by the pdf document source, you need to use GET method. PDF file path is specified in the request URL, as:

GET:`http://<host>[:port]/api/pdf/{pdfpath}/$pdf/features`

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/pdf/PdfRoot/C1XapOptimizer.pdf/$pdf/features
Status Code: 200
```

pdfpath - The pdf file that storage manager can recognize.

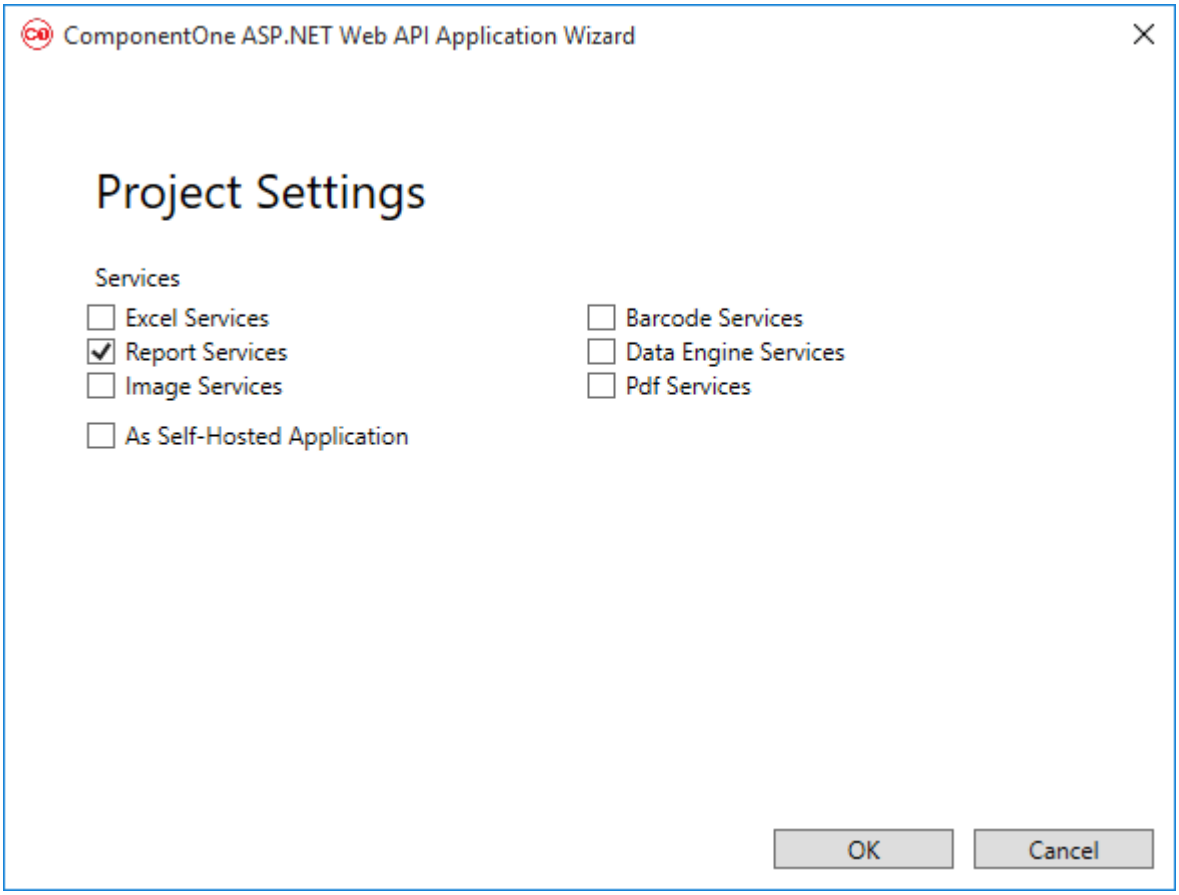
Response Messages

HTTP Status Code	Reason
200	Json string contains IDocumentFeatures type object.
404	The pdf path does not exist.

For more information on PDF Services, see WebApiExplorer demo.

Report Services

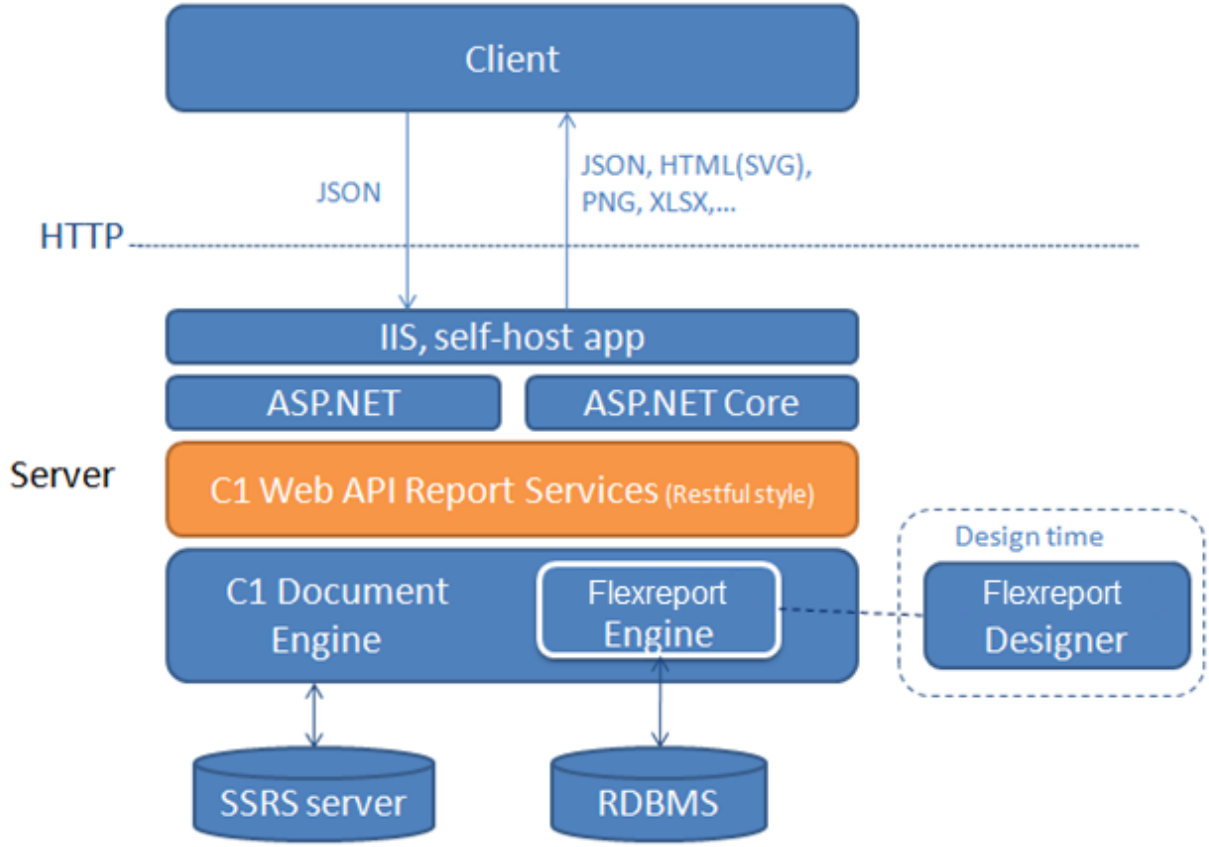
Web API Studio Edition provides Report Services that enables you to build HTTP services, which can be consumed by a variety of clients for viewing, loading and caching FlexReports and SSRS reports. These are REST based API services, which communicate with HTML 5 ReportViewer control to display the report content on the web. You can view a variety of reports in the HTML5 based ReportViewer control, mainly FlexReport and SSRS reports.



Available as Visual Studio template, C1 Web API enables you to create reporting services on Visual Studio. Client applications send a request to the report service applications to load reports, export reports, and allow users to use parameters in the report. The service supports exporting your reports to PDF, Excel, HTML, RTF, OpenXML, and Image.

Configuring FlexReports Web API

FlexReport is a comprehensive reporting tool that provides complete reporting solution - from building complex reports to previewing, exporting, and printing. With a rich object model, and modern user interfaces in its previewing control and designer application. Reporting is significant to business decision making and knowledge management.



ComponentOne Studio Web API Edition enables the users to create report services that enables you to build HTTP services, which can be used for different type of clients for viewing, loading and caching reports. These are REST based API services, which communicate with the **ReportViewer** (HTML 5 based Report Viewer) to display the report content on the web.

Note: To work with FlexReport control, the minimal server configuration requirement is Windows Server 2008 R2.

C1 Web API Report Services can load FlexReport file (.flxr) created using FlexReportDesigner application. For more information, see FlexReport for WinForms. You can create a FlexReport Web API Report Services application using the following two ways.

- Using C1 Web API Template
- Using Standard Visual Studio Web API Template

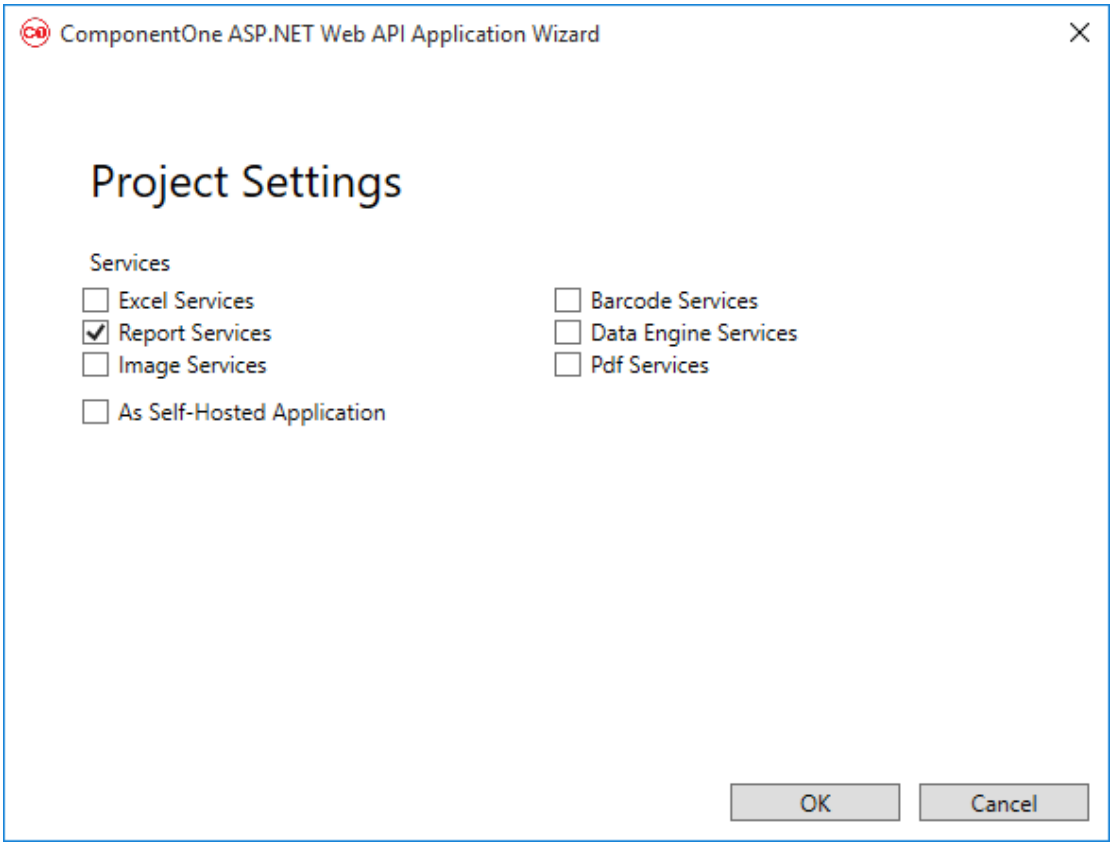
Using ComponentOne Web API Edition Template

If you have installed Web API Edition from C1Studio installer, then it is easy to create a project pre-configured with FlexReport Web API. Use the C1 Web API Template to create a new project named FlexreportWebAPI and follow the steps below:

- **Step 1: Create a new WebAPI application**
- **Step 2: Add Report files to Web API Application**
- **Step 3: Set Report's Root Location**
- **Step 4: Deploy FlexReport Web API Service**

Step1: Create a new WebAPI application

1. In Visual Studio, select **File | New | Project** to create a new Web API Service Project.
2. Under installed templates, select **Visual C# | Web | C1 Web API Application** to create a new C1 Web API Service application.
3. Set a **Name** and **Location** for your application, and then Click **OK.**
4. In the **ComponentOne ASP.NET Web API Application Wizard**, select **Report services** option.



5. Once you have selected the **Services** from the wizard, click **OK** to create a new **C1 Web API Service application**.

Step 2: Add Report files to Web API Application

Complete the following steps to add report files to your application.

1. Add a folder named **Files** to your application.
2. Add the FlexReport's report definition file to it.
   If the report is using any local database (such as an MDB or an MDF file), then add the database to the **App_Data** folder of your application. However, make sure that the connection string of the reports are pointing to the **App_Data** folder accordingly.

```
<DataSources>
  <DataSource>
    <Name>Main</Name>
    <DataProvider>OLEDB</DataProvider>
    <ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|C1Demo.mdb;Persist Security Info=False</ConnectionString>
    <RecordSource>Employees</RecordSource>
    <RecordSourceType>TableDirect</RecordSourceType>
  </DataSource>
</DataSources>
```

> **Note:** Make sure you add **Microsoft.Owin.Cors** Nuget package in your application, for app.UseCors();

Back to Top

Step 3: Set Report's Root Location

1. In the **Startup.cs** file, add the following code inside **Configuration** method of the **Startup class**.

Startup.cs

```
 app.UseCors(CorsOptions.AllowAll);
 app.UseReportProviders()
.AddFlexReportDiskStorage("ReportsRoot", GetFullRoot("Files"))
```

This code registers the folder/location where the Report files will be stored, in this case it is the "Files" folder.

2. Add the **GetFullRoot** function inside **Startup** class.

Startup.cs

```
private static string GetFullRoot(string root)
{
        var applicationBase =
AppDomain.CurrentDomain.SetupInformation.ApplicationBase;
        var fullRoot = Path.GetFullPath(Path.Combine(applicationBase, root));
        if (!fullRoot.EndsWith(Path.DirectorySeparatorChar.ToString(),
StringComparison.Ordinal))
        {
                fullRoot += Path.DirectorySeparatorChar;
        }
        return fullRoot;
}
```

Back to Top

Step 4: Deploy Report Web API Service

1. Compile the project.
2. Deploy the project to IIS.

The Web API URL, for service hosted on local IIS, will be http://localhost/FlexReportwebAPI/api/report**.**

> **Note:** Once you have successfully created **Web API URL** for Report Services, you can access and view reports stored in the service using FlexViewer for MVC and Wijmo Viewer. For more information on how to view reports, see Viewing Reports in FlexViewer.

Back to Top

Using Standard Visual Studio Web API Template

Complete the following steps to configure FlexReport Web API using standard Visual Studio Template for Web API:

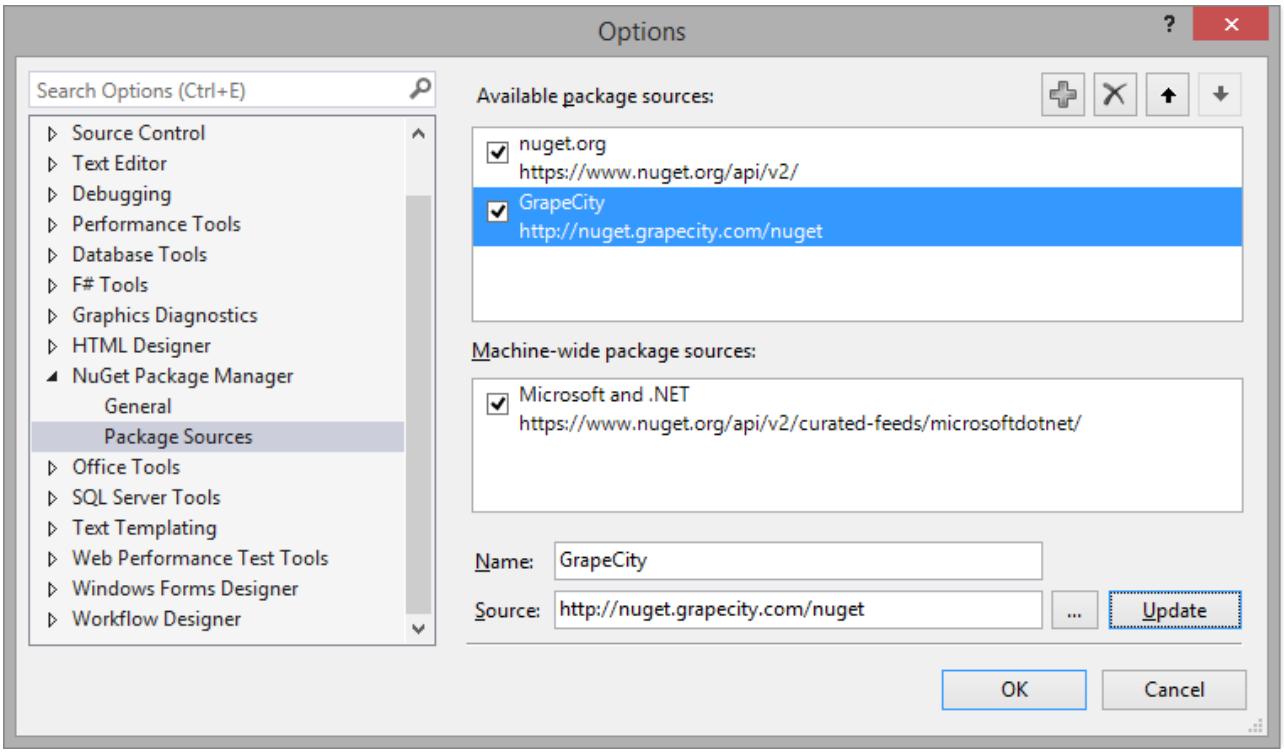
- **Step 1: Configure Web API Project**
- **Step 2: Add Report files to the Project**
- **Step 3: Set Report's Root Location and Use C1 Web APIs**
- **Step 4: Deploy FlexReport Web API Service**

Step 1: Configure Web API Project

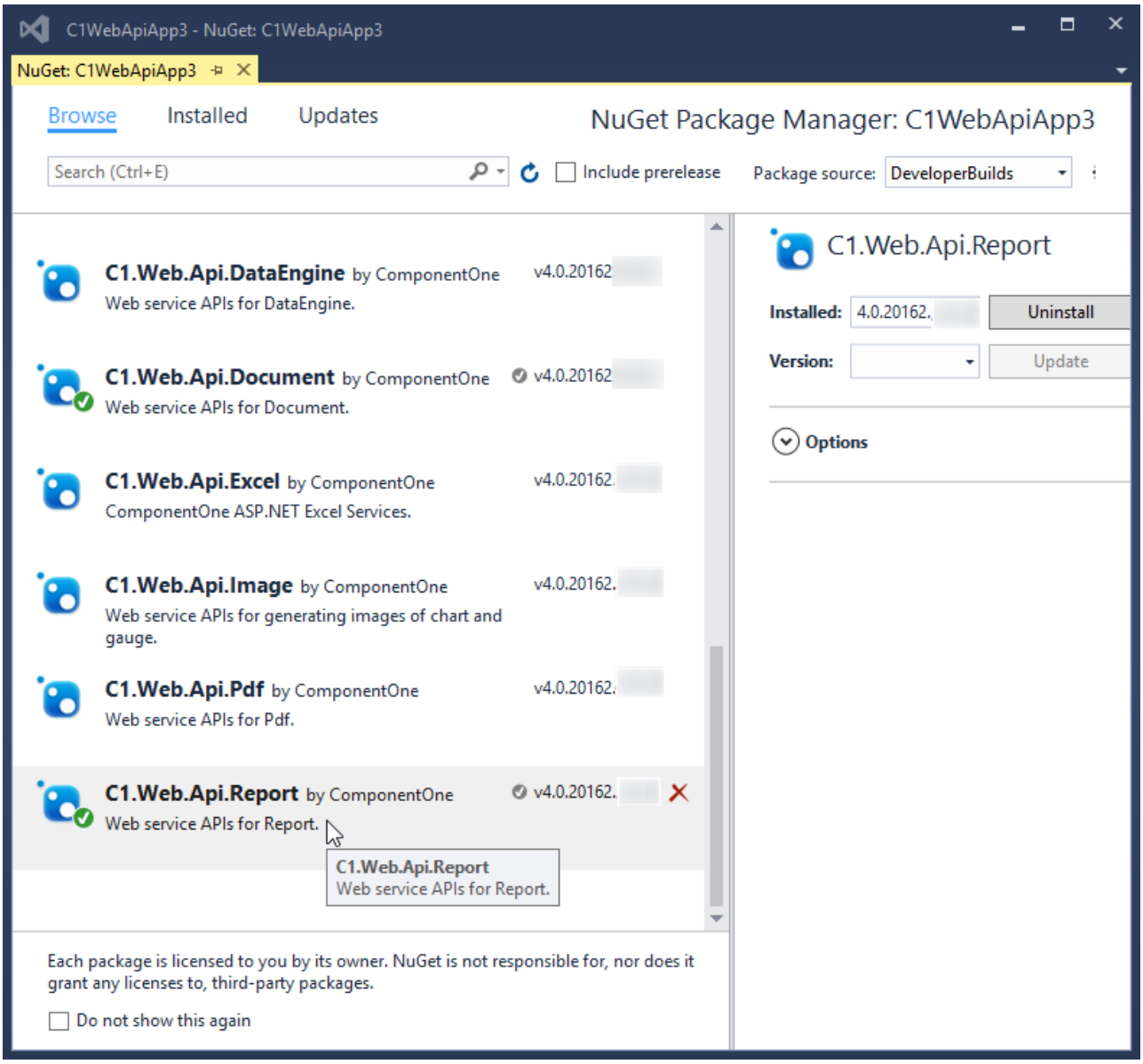
Complete the following steps to configure Web API project:

1. Create a new ASP.NET Web API Project.
2. Add the FlexReport Web API package from GrapeCity NuGet.

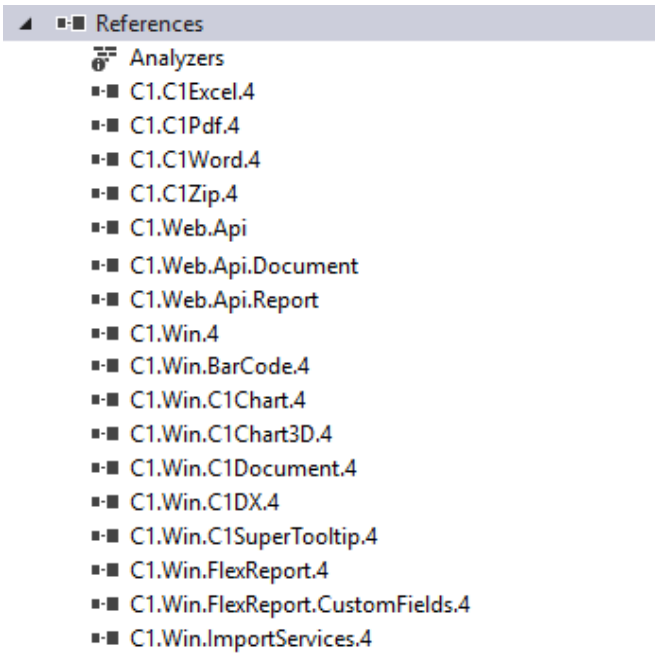
   If you have installed MVC Edition, the GrapeCity NuGet source path gets already set inside Visual Studio.



Alternatively, you can manually add the source path from **NuGet Package Manager** | **Package Sources** option available in **Options** dialog box, which appears on selecting **Tool** | **NuGet Package Manager** | **Package Manager Settings**, as discussed in Configuring NuGet Package Sources topic.
In the NuGet package manager the Report Service is listed as shown in the following image:



FlexReport Web API adds the following references to the project:



3. License your Web API application. Create a **licenses.licx** file within Properties folder of your application and add the following code in it:

licenses.licx

```
C1.Web.Api.LicenseDetector, C1.Web.Api

C1.Web.Api.Report.LicenseDetector, C1.Web.Api.Report
```

Back to Top

Step 2: Add Report files to the Project

1. Create a folder named **Files** in your application.
2. Add the FlexReport Definition file to it.

   If the report is using any local database (such as MDB or MDF files), then add the database file to the **App_Data** folder of your project. However, make sure that the connection string of the reports are pointing to the App_Data folder accordingly.

```
<DataSources>
  <DataSource>
    <Name>Main</Name>
    <DataProvider>OLEDB</DataProvider>
    <ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|C1Demo.mdb;Persist Security Info=False</ConnectionString>
    <RecordSource>Employees</RecordSource>
    <RecordSourceType>TableDirect</RecordSourceType>
  </DataSource>
</DataSources>
```

Back to Top

Step 3: Set Report's Root Location and Use C1 Web APIs

1. Open the **Startup.cs** file and add the following code to Configuration method:

Startup.cs

```
public void Configuration(IAppBuilder app)
{
    app.UseCors(CorsOptions.AllowAll);
    var folder = GetFullRoot("Files");
```

```
    app.AddDiskStorage("root", folder);
    ConfigureAuth(app);
}
```

Note: Make sure you add **Microsoft.Owin.Cors** Nuget package in your application, for app.UseCors();

2. Add the following GetFullRoot function in the startup class.

Startup.cs

```
private static string GetFullRoot(string root) {
var applicationBase = AppDomain.CurrentDomain.SetupInformation.ApplicationBase;
var fullRoot = Path.GetFullPath(Path.Combine(applicationBase, root));
if (!fullRoot.EndsWith(Path.DirectorySeparatorChar.ToString(),
StringComparison.Ordinal)) {
    fullRoot += Path.DirectorySeparatorChar;
  }
  return fullRoot;
}
```

3. Open Web.Config and add the following entry under handlers inside system.webServer Node.

Web.config

```
<add name="ExtensionlessUrlHandler-Integrated-4.0" path="api/*" verb="*"
type="System.Web.Handlers.TransferRequestHandler"
preCondition="integratedMode,runtimeVersionv4.0" />
```

4. In the Web.config file, ensure that the **WebDAVModule** and the **WebDAV** handler are removed from <system.webServer> as shown below:

Web.config

```
<system.webServer>
<modules>
      <remove name="WebDAVModule" />
    </modules>
<handlers>
      <remove name="WebDAV" />
      .........
    </handlers>
</system.webServer>
```

Back to Top

Step 4: Deploy Report Web API Service

1. Compile the project.
2. Deploy the project to IIS.

The Web API URL, for service hosted on local IIS, will be http://localhost/FlexReportwebAPI/api/report**.**

Note: Once you have successfully created **Web API URL** for Report Services, you can access and view reports stored in the service using FlexViewer for MVC and Wijmo Viewer. For more information on how to view reports, see Viewing Reports in FlexViewer.

Register Report Provider

ComponentOne Studio Web API Edition provides support for SSRS reports for the WebApi services. To work with

different report types in C1 WebApi, you need to specify the report provider information in the service request URL. Before you specify the provider information in the Service URL, you need to register the report provider in your Visual Studio application.

To locate the real path of the report, you need to register a report provider which provides the root path of the report. The report provider should be registered by using the **ReportProviderManager class** in the Startup.Configuration section of your application.

Steps to Register Report Provider

For FlexReport

1. From the Solution Explorer window in Visual Studio, select and open the **Startup.cs** file.

2. In the Startup.cs file, under the **Configuration** section, you can use **ReportProviderManager.AddFlexReportStorage()** method and **ReportProviderManager.AddFlexReportDiskStorage()** method to add the storage information which contains report definitions (.flxr or .xml).

   Example

   ```
   app.UseReportProviders()
   .AddFlexReportDiskStorage("Root", @"../reports");
   ```

For SSRS Reports

1. Form the Solution Explorer window in Visual Studio, select and open the **Startup.cs** file.

2. In the Startup.cs file, under the **Configuration** section, you can use **ReportProviderManager.AddSsrsReportHost()** method to add the SSRS server URL configuration in your application.

   Example

   ```
   app.UseReportProviders()
   .AddSsrsReportHost("local", http://demo.ssrsreports.com/ReportServer);
   ```

Report Service

Report service supports C1Report, FlexReport, and SSRS Report.

URL Description for FlexReport and SSRS reports

Following table shows the difference between the URL pattern for FlexReports and SSRS reports

URL Element	FlexReports	SSRS Reports
Provider	Specifies the storage path or the root path which contains the reports definition files.	Specifies the Web service URL of the report server where the report definition is stored.
Folder Path	The relative path of the folder that contains report definitions. FlexReport definition file (.flxr or .xml) is also considered as a folder, since one report definition file contains multiple	Web service URL provides the folder location where the report definition is

	reports.	stored.
Report Name	The report name defined in the FlexReport definition file. For example: "Simple List"	The report file name in the specified folder. For example: "Employee_Sales_Summary"
Example Report Path	ReportsRoot/report/CommonTasks.flxr/Simple List	C1SSRS/AdventureWorks /Employee_Sales_Summary

Report Info

Report info service gets the information related to the report for the specified report path. Your client application sends an HTTP request to the service application for using Report info WebAPI.

Report Information Service Request Schema

To obtain the information related to a specific report using the report path, you need to use GET method. Report file path is specified in the request URL as:

```
GET: http://<host>[:port]/api/report/{reportpath}/$report
```

```
Request URL: http://demos.componentone.com/ASPNET/c1webapi/4.0.20163.79/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$report
Status Code: 200
```

Response Messages

HTTP Status Code	Reason
200	Json string contains IReportInfo type object.
404	The report path does not exist.

Catalog

Catalog service uses GET method to enable the end users to fetch the catalog information from the specified folder path. Client application sends an HTTP request to the service application for using the **Catalog** API.

Catalog Service Request Schema

To obtain the catalog information from a specified folder path, you need to use GET method. Report folder path is specified in the request URL, as:

GET:`http://<host>[:port]/api/report/{folderpath}?[recursive]`

Response is a Json string that contains ICatalogItem object which describes the folder or report.

recursive - Recursive is a request parameter. It is set to **False** by default. When you set the recursive to **True,** the service returns the entire list of child items below the specified item.

URL Parameters

The Catalog service URL primarily accepts two parameters **folderpath** and **reportpath**. In case of FlexReport files, you need to specify the report file name, which is recognized by the storage manager, to access the list of reports.

Parameters Information

Parameters information service uses Parameters API to enable users to get the parameters description for specified report and use this information to initialize parameter panel. Your client application sends an HTTP request to the service application for using Parameters API.

Parameters Information Service Request Schema

To obtain the information of all the parameters defined in a report through client, you need to use GET method. Report file path and report name are specified in the request URL, as:

```
GET: http://<host>[:port]/api/report/{reportpath}/$report/parameters
```

The following illustration depicts a request URL with parameters to get a report's parameter description.

```
Request URL: http://testdemos.componentone.com/aspnet/webapi/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$report/parameters
Status Code: 200
```

URL Parameters

Parameters service URL primarily accepts **report path**. You need to specify the full report file path in the Request Url.

Response Messages

HTTP Status Code	Reason
200	Json string containing collection of IParameter.
404	The report path does not exist.

Parameter Service Request Schema

To get the parameter with specified name defined in the report definition, you need to use GET method. Report file path and report name are specified in the request URL, as:

```
GET:http://<host>[:port]/api/report/{reportpath}/$report/parameters/{parametername}
```

URL Parameters

Parameters service URL primarily accepts two parameters **report path** and **parameter name**. You need to specify the report file name and the parameter name in the Request Url.

Response Messages

HTTP Status Code	Reason
200	Json string containing IParameter type object.
404	The report path or parameter name does not exist.

For more information on Report Services, see WebApiExplorer demo.

Export

Export your report data to different file formats any time and anywhere. The report layout and style along with the data gets exported to the specified format.

Export service uses **Export** API to enable end users to render and export a specified report to a desired format on the fly. You can also set different options such as parameters and pageSettings in the service URL. Your client application sends an HTTP request to the service application for using **Export** API to export a report to the supported formats.

[GET] Export Service Request Schema

To render and export the report to the specified export filter with options and the specified page settings or/and parameters, you can use GET method. Report file path, exportFileName, exportOptions, parameters, and pageSettings are specified in the request URL. Your client application sends an HTTP request message to the service, as:

GET: `http://<host>[:port]/api/report/{reportpath}/$report/export?[exportFileName]&[parameters]&[pageSettings]&[exportoptions]`

The following illustration depicts a request URL with parameters to export a report.

URL Parameters

Export Service URL includes the following parameters.

Parameter	Description
reportpath	Specify the full report path.
exportFileName	Specify a file name for the exported file.
exportOptions	List the available options using IExportOptions.
parameters	Specify the parameters.
pageSettings	Specify the page settings using IPageSettings.

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The report path or format name does not exist.
406	The format name is not acceptable.

[POST] Export Service Request Schema

To render and export the report to the specified export filter with options and the specified page settings or/and parameters, you can use POST method. Report file path is specified in the request URL. Your client application sends an HTTP request message to the service, as:

POST: `http://<host>[:port]/api/report/{reportpath}/$report/export`

Post Data includes the following options.

- exportFileName
- exportOptions
- parameters
- pageSettings

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The report path or format name does not exist.
406	The format name is not acceptable.

For more information on Report Services, see WebApiExplorer demo.

Supported Formats

Supported Format uses SupportedFormats API to enable end users to get the supported export formats for a report. Your client application sends an HTTP request to the service application for using SupportedFormats API to obtain the information about the supported export formats for a report.

Supported Formats Service Request Schema

To get the supported export formats for a report through client, you need to use GET method. Report file path and report name are specified in the request URL, as:

```
GET: http://<host>[:port]/api/report/{reportpath}/$report/supportedformats
```

The following illustration depicts a request URL with parameters to load a report from storage.

```
Request URL: http://testdemos.componentone.com/aspnet/webapi/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$report/supportedformats
Status Code: 200
```

URL Parameters

Supported Formats URL primarily accepts **Report Path** as parameter. You need to specify the report file path in the Request Url. For more information on Report Services, see WebApiExplorer demo.

Response Messages

HTTP Status Code	Reason
200	Json string contains a collection of IExportDescription.
404	The report path or format name does not exist.

Supported Format Service Request Schema

To get the supported export format for the specified report, you need to use the GET method. Report file path and report name are specified in the request URL, as:

```
GET:http://<host>[:port]/api/report/{reportpath}/$report/supportedformats/{formatname}
```

URL Parameters

Supported format service URL accepts two parameters **Report Path** and **Format Name**. You need to specify the report file name and the format name in the Request Url.

Response Messages

HTTP Status Code	Reason

200	Json string contains IExportDescription type object.
404	The report path does not exist.

For more information on Report Services, see WebApiExplorer demo.

Report Instances

Report services provides various APIs that help you to work with cached report instance. These are RESTful API services, which enable you to load, render, export, view parameters, and update page settings of a report. It can communicate with the ReportViewer control to display the report content on the web. The service URL includes the **$instances** keyword to access the cached information of a report.

$instances - Indicates functions for the report runtime.

You can make use of the following Web API services that are based on **$instances** keyword.

HTTP Methods	Description
POST: /api/report/{reportpath}/$instances	Gets all cached report instances for the specified report path.
GET: /api/report/{reportpath}/$instances/{instanceid}	Gets the info of cached report instance with the specified instance id.
POST: /api/report/{folder path}/{report name}/$instances/{instance id}/render	Renders the report instance with the specified instance id.
GET: /api/report/{reportpath}/$instances/{instanceid}/status	Gets the status of the cached instance.
DELETE: /api/report/{folder path}/{report name}/$instances/{instance id}	Delete the instance.
GET: /api/report/{reportpath}/$instances/{instanceid}/parameters	Gets all parameters in the cached instance.
PUT: /api/report/{folder path}/{report name}/$instances/{instance id}/parameters	Updates all parameter values in the report instance with specified instance id.
PATCH: /api/report/{folder path}/{report name}/$instances/{instance id}/parameters	Updates specified parameter values in the report instance with specified instance id.
GET: /api/report/{reportpath}/$instances/{instanceid}/parameters/{parametername}	Gets info of the parameter with specified name in the cached instance.
PUT: /api/report/{folder path}/{report name}/$instances/{instance id}/parameters/{parameterName}	Updates the value of parameter with specified name in the report instance with specified instance if.
GET: /api/report/{reportpath}/$instances/{instanceid}/pagesettings	Gets the current page settings in the cached instance.
PUT: /api/report/{folder path}/{report name}/$instances/{instance id}/pagesettings	Updates all page settings properties in the report instance

	with specified instance id.
PATCH: /api/report/{folder path}/{report name}/$instances/{instance id}/pagesettings	Updates the specified page settings properties in the report instance with specified instance id.
GET: /api/report/{reportpath}/$instances/{instanceid}/outlines	Gets the outlines of the report with specified in the cached instance.
GET: /api/report/{reportpath}/$instances/{instanceid}/bookmarks/{bookmarkid}	Gets info of the bookmark with specified bookmark id in the cached instance.
GET: /api/report/{reportpath}/$instances/{instanceid}/search?[text]&[matchCase]&[wholeWord]	Gets the search result in the cached instance.
GET: /api/report/{reportpath}/$instances/{instanceid}/export?[options]	Exports the cached instance to specified format.
POST: /api/report/{reportpath}/$instances/{instanceid}/export	Exports the cached instance to specified format.
GET: /api/report/{reportpath}/$instances/{instanceid}/supportedformats/{formatname}	Gets supported export format width the specified name for the report.
GET: /api/report/{folder path}/{report name}/$instances/{instance id}/supportedformats	Gets all export formats supported by the report instance with specified instance id.
GET: /api/report/{reportpath}/$instances/{instanceid}/features	Gets the features supported by the report instance.

Instance

Report services provide GET, POST, UPDATE, and DELETE methods to work with report instances. Report instances help you to create a new report instance, get information of the report instance and render the report instance. Your client application sends an HTTP request to the service application for using the report instances API to get the information for a particular report instance.

[POST] Create a new instance

To create a new report instance from the specified report, you need to use **POST** method. Report folder path and report name are specified in the request URL, as follows:

```
POST: http://<host>[:port]/api/report/{folder path}/{report name}/$instances
```

Example URL
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$instances`

Parameters

Parameter	Description
Folder Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Report Name	Specify the report name that is defined in the FlexReport file, or under the SSRS folder.

Report Parameters	Check the response of action parameters. For example: pCategory=1&... If the parameter is multi-value, set the parameters multiple times. For example: pCategory=1&pCustomers=3&...
Report Page Settings	Set the page settings. Example: `paperSize=custom&height=10in&width=20.5cm`

Response Messages

HTTP Status Code	Reason
201	JSON string contains IReportInstanceInfo type object, for the new created instance.
404	The report path does not exist.

Back to Top

[GET] Get report information

To get information of a report instance with the specified instance id, you need to use **GET** method. Report full path and Instance id are specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}
```

Example URL
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot /FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$instances`

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains IReportInstanceInfo type object.
404	The report path or instance id does not exist.

Back to Top

[POST] Render a report

To render a report instance with the specified instance id, you need to use **POST** method. Report full path and instance id are specified in the request URL, as follows:

```
POST: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}/render
```

Example URL
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/Simple List$instances/21c2fa46-0a75-4708-acf2-454668eac4cb/render`

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report.

	For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Report Parameters	Check the response of action parameters. For example: parameters.pCategory=1 If the parameter is multi-value, set the parameters multiple times. For example: pCategory=1&pCustomers=3&...
Report Page Settings	Set the page settings. For example: pageSettings.paperSize=custom&pageSettings.height=10in&pageSettings.width=20.5cm
Action String	Specify the action string. Action string is only valid for SSRS reports. We recommend you to use AdventureWorks (SSRS)/Product Line Sales report. For example: actionString=Sort;33iT0;Ascending

Response Messages

HTTP Status Code	Reason
200	(Render completed) JSON string contains IReportStatus type object.
202	(On Rendering) The render action is accepted.
404	The report path or instance id does not exist.

Back to Top

[GET] Get status of a report

To get the status of the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id are specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/status
```

Example URL
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/20f8e7c7-ea84-4961-81f7-5b8ff662f588/status

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains IReportStatus type object.
404	The report path or instance id does not exist.

Back to Top

[DELETE] Delete an instance

To delete the instance, you need to use **DELETE** method. Report full path and instance id are specified in the request URL, as follows:

```
DELETE: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}
```

Example URL
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List$instances/21c2fa46-0a75-4708-acf2-454668eac4cb

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
204	Successfully deleted the instance.
404	The report path or instance id does not exist.

Back to Top

For more information about Report Instances. see WebApiExplorer Demo.

Parameters

Report instance helps you to fetch the parameter descriptions for a specified report, update the value of all the parameters in the report, update a specific parameter in the report and fetch information of a specific parameter using a parameter name. You can create a client application to send an HTTP request to the service application for using the report services.

[GET] Parameters Service Request Schema

To get the parameters in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/parameters
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/parameters`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of IParameter.
404	The report path does not exist.

Back to Top

[PUT] Parameters Service Request Schema

To update all the parameter values in the report instance with the specified instance id, you need to use **PUT** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
PUT: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/parameters
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/parameters`	

Parameters

Parameter	Description

Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Report Parameters	Check the response of action parameters. Example: pCategory=1&... If the parameter is multi-value, set the parameters multiple times. Example: pCategory=1&pCustomers=3&...

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of IParameter interface. New parameters with validation.
404	The report path or instance id does not exist.

Back to Top

[PATCH] Parameters Service Request Schema

To update a specific parameter value in the report instance with the specified instance id, you need to use **PATCH** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
PATCH: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/parameters
```

Example URL — Copy Code

```
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks
/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/parameters
```

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Report Parameters	Check the response of action parameters. Example: pCategory=1&... If the parameter is multi-value, set the parameters multiple times. Example: pCategory=1&pCustomers=3&...

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of IParameter interface. New parameters with validation.
404	The report path or instance id does not exist.

Back to Top

[GET] Parameters Service Request Schema

To get the parameter using a specific parameter name in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/parameters/{parameterName}
```

Example URL — Copy Code

```
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks
/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/20f8e7c7-ea84-4961-81f7-5b8ff662f588/parameters/pCategory
```

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Parameter Name	Specify the parameter name. For example: pCategory

Response Messages

HTTP Status Code	Reason
200	JSON string contains IParameter type object.

404	The report path or instance id does not exist.

Back to Top

[PUT] Parameters Service Request Schema

To get the parameter using a specific parameter name in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
PUT: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/parameters/{parameterName}
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/20f8e7c7-ea84-4961-81f7-5b8ff662f588/parameters/pCategory`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Parameter Name	Specify the parameter name. For example: pCategory
Parameter Values	Specify the report parameter values.

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of IParameter interface. New parameters with validation.
404	The report path or instance id does not exist.

For more information about Report Instances. see WebApiExplorer Demo.

Page Settings

Report instance helps you to get the current page settings of a report, update all the page setting properties, and update a specific page setting property in a report. You can create a client application to send an HTTP request to the service application for using the report services.

[GET] PageSettings Service Request Schema

To get the current page settings in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/pagesettings
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/pagesettings`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains IPageSettings type object.
404	The report path or instance id does not exist.

Back to Top

[PUT] PageSettings Service Request Schema

To update all the page settings properties in the report instance with the specified instance id, you need to use **PUT** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
PUT: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/pagesettings
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/pagesettings`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Report Page Settings	Set the page settings. Example: `paperSize=custom&height=10in&width=20.5cm`

Response Messages

HTTP Status Code	Reason
200	JSON string contains IPageSettings type object, which is the new page settings. Use the default value if one property is not set.
404	The report path or instance id does not exist.

Back to Top

[PATCH] PageSettings Service Request Schema

To update a specific page setting property in the report instance with the specified instance id, you need to use **PATCH** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
PUT: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/pagesettings
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks /FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/pagesettings`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Report Page Settings	Set the page settings. Example: `paperSize=custom&height=10in&width=20.5cm`

Response Messages

HTTP Status Code	Reason
200	JSON string contains IPageSettings type object, which is the new page settings.
404	The report path or instance id does not exist.

For more information about Report Instances. see WebApiExplorer Demo.

Supported Formats

Report instance helps you to fetch all the supported formats for your report and gets a specific export format with the specified format name supported by the report. You can create a client application to send an HTTP request to the service application for using the report services.

[GET] SupportedFormats Service Request Schema

To get all the supported export formats, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/supportedformats
```

Example URL	Copy Code
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/supportedformats	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of IExportDescription.
404	The report path or instance id does not exist.

Back to Top

[GET] SupportedFormats Service Request Schema

To get the export format with a specific name supported by the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/supportedformats/{format name}
```

Example URL	Copy Code
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/supportedformats/html	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Supported Format Name	Specify the supported format name. For example: html

Response Messages

HTTP Status Code	Reason
200	JSON string contains IExportDescription type object.
404	The report path or instance id or format name does not exist.

For more information about Report Instances. see WebApiExplorer Demo.

Features

Report instance allows you to fetch all the different features that are supported in your report. You can create a client application to send an HTTP request to the service application for using the report services.

[GET]Features Service Request Schema

To get the features supported by the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}/features
```

Example URL
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/MultiValue DataBound Parameters$instances/9e20a2d1-e371-44c2-87f2-bc65700172eb/features

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains IDocumentFeatures type object.
404	The report path or instance id does not exist.

For more information about Report Instances. see WebApiExplorer Demo.

Export

Report instance allows you to export your report to different file formats. The report layout and style along with the data gets exported to the specified format. Export service enables the end user to render and export a specified report to desired format using the specified instance id. You can create a client application to send an HTTP request to the service application for using the report services.

[GET] Export Service Request Schema

To export the report instance with report filter and option using the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}/export
```

Example URL
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$instances/21c2fa46-0a75-4708-acf2-454668eac4cb/export?format=html&exportFileName=Simple List`

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Format	Specify the supported format. For example: HTML
Export File Name	Specify a suitable name for the exported file.
Export Options	Specify the export options. For example: outputRange=1&...

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The report path or instance id does not exist or the request page range is not rendered.
406	The format name is not acceptable/supported.

Back to Top

[POST] Export Service Request Schema

To export the report instance with report filter and option using the specified instance id, you need to use **POST** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
POST: http://<host>[:port]/api/report/{folder path}/{report name}/$instances/{instance id}/export
```

Example URL
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List/$instances/21c2fa46-0a75-4708-acf2-454668eac4cb/export`

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/MultiValue DataBound Parameters
Instance Id	Specify the instance id of the executing report.
Format	Specify the supported format. For example: HTML
Export File Name	Specify a suitable name for the exported file.
Export Options	Specify the export options. For example: outputRange=1&...

Response Messages

HTTP Status Code	Reason
200	The exported file stream.
404	The report path or instance id does not exist or the request page range is not rendered.
406	The format name is not acceptable/supported.

For more information about Report Instances. see WebApiExplorer Demo.

Back to Top

Bookmarks and Search

Report instance allows you get information related to a specific bookmark using the bookmark name and also provides search functionality to get the search results of the report. You can create a client application to send an HTTP request to the service application for using the report services.

[GET] Bookmarks Service Request Schema

To get the bookmark using a specific bookmark name in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as follows:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/bookmarks/{bookmark name}
```

Example URL	Copy Code
`http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr/Simple List$instances/21c2fa46-0a75-4708-acf2-454668eac4cb/bookmarks/bookmark1`	

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List

Instance Id	Specify the instance id of the executing report.
Bookmark Name	Specify the bookmark name.

Response Messages

HTTP Status Code	Reason
200	JSON string contains IDocumentPosition type object.
404	The report path or instance id or bookmark id does not exist.

Back to Top

[GET] Search Service Request Schema

To get the search result of the report with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/search
```

Example URL — Copy Code

```
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks
/FlexCommonTasks.flxr/Simple List$instances/21c2fa46-0a75-4708-acf2-454668eac4cb
/search?text=Steven Buchanan&matchCase=true&wholeWord=true
```

Parameters

Parameter	Description
Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Instance Id	Specify the instance id of the executing report.
Search Text	Specify a value which is used to search.
Match Case	Boolean value indicates if search the value with case sensitive.
Whole Word	Boolean value indicates if search the value with matching a whole word

Response Messages

HTTP Status Code	Reason
200	JSON string contains a collection of ISearchResult type objects.
404	The report path or instance id does not exist.

Back to Top

[GET] Outlines Service Request Schema

To get all the outlines in the report instance with the specified instance id, you need to use **GET** method. Report full path and instance id of executing report is specified in the request URL, as:

```
GET: http://<host>[:port]/api/report/{folder path}/{report name}$instances/{instance id}/outlines
```

Example URL — Copy Code

```
http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report/ReportsRoot/FlexCommonTasks
/FlexCommonTasks.flxr/Simple List$instances/21c2fa46-0a75-4708-acf2-454668eac4cb/outlines
```

Parameters

Parameter	Description

Report Full Path	Specify the full path of the executing report. For example: FlexCommonTasks.flxr/Simple List
Instance Id	Specify the instance id of the executing report.

Response Messages

HTTP Status Code	Reason
200	JSON string contains collection of IOutlineNode type objects.
404	The report path or instance id does not exist.

For more information about Report Instances. see WebApiExplorer Demo.

Excel Services

Web API Studio Edition provides excel services for- import and export of MVC and Wijmo 5 controls, and REST API services to generate and merge excel files.

Inside Web API Edition, you will find HTTP request messages which provide export and import functionalities for Excel files. Also, GET and POST methods aid in generating and merging excel files. Available as Visual Studio template, Web API Edition enables you to create Web API service on Visual Studio. This service can then be consumed by client applications to export/import FlexGrid and Excel data, and generate and merge excel files.

Export Service

Export FlexGrid data to excel any time and anywhere with Web API Edition Export services. Its layout and style along with the data gets exported to the specified format. This means FlexGrid with merged cells or headers, formatted text, and grouped data is exported as is. You can include or exclude column headers of grid data while exporting to excel format.

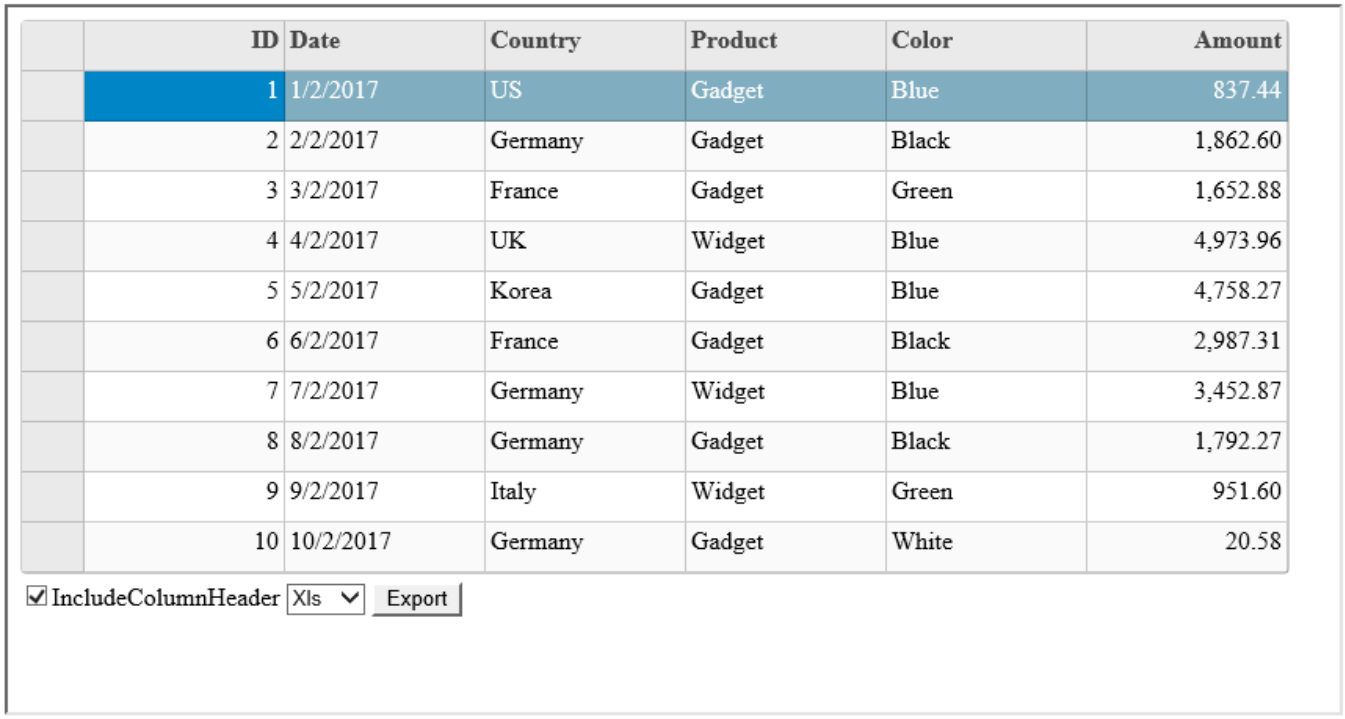
Export FlexGrid to Excel

This section shows how to call the Web API service project through a client application and add export function for exporting FlexGrid data as an excel.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the FlexGrid appears after completing the steps above:



	ID	Date	Country	Product	Color	Amount
	1	1/2/2017	US	Gadget	Blue	837.44
	2	2/2/2017	Germany	Gadget	Black	1,862.60
	3	3/2/2017	France	Gadget	Green	1,652.88
	4	4/2/2017	UK	Widget	Blue	4,973.96
	5	5/2/2017	Korea	Gadget	Blue	4,758.27
	6	6/2/2017	France	Gadget	Black	2,987.31
	7	7/2/2017	Germany	Widget	Blue	3,452.87
	8	8/2/2017	Germany	Gadget	Black	1,792.27
	9	9/2/2017	Italy	Widget	Green	951.60
	10	10/2/2017	Germany	Gadget	White	20.58

Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For further details refer to C1 Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views|MVCFlexGrid|Index.cshtml** to export FlexGrid data to excel.

Index.cshtml copyCode

```
<script type="text/javascript">
    function exportFlex() {
    var exporter = new c1.mvc.grid.ExcelExporter();
    fileType = document.getElementById("mySelect").value;
    var gridcontrol = wijmo.Control.getControl("#flexGrid");
    exporter.requestExport(gridcontrol,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/excel", {
        fileName: "exportFlexGrid",
        type: fileType,
        });
    }
</script>
```

HTML

Add the following markup in <script> tags to export FlexGrid data to excel.

JavaScript

```
<script type="text/javascript">
        function exportFlex() {
        var exporter = new wijmo.grid.ExcelExporter();
        var grid = wijmo.Control.getControl("#TheFlexGrid");
        exporter.requestExport(grid,
                "http://demos.componentone.com/ASPNET/WebAPI/api/export/excel",
{
        fileName: "export",
        type: wijmo.ExportFileType.Xls,
        });
}
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.grid.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<div>
    <select id="mySelect">
        <option selected>Xls</option>
        <option>Xlsx</option>
        <option>Csv</option>
    </select>
<button onclick="exportFlex()" title="Export">Export</button>
</div>
```

HTML

Add the following markup within the <body> tags to create button for export function.

HTML

```
<button onclick="exportFlex()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the data in the control to Excel format.

> Explore detailed demo samples of export service for MVC and Wijmo5 FlexGrid control in:
>
> - MVCFlexGrid Live Demo
> - Wijmo5FlexGrid Live Demo

Back to Top

Import Service

Easily import excel data into FlexGrid control. The service supports importing excel data from xlsx, xls, and csv file formats. The service provides flexibility to import excel data with or without column header. For excel files having formula, calculated field for FlexGrid is imported.

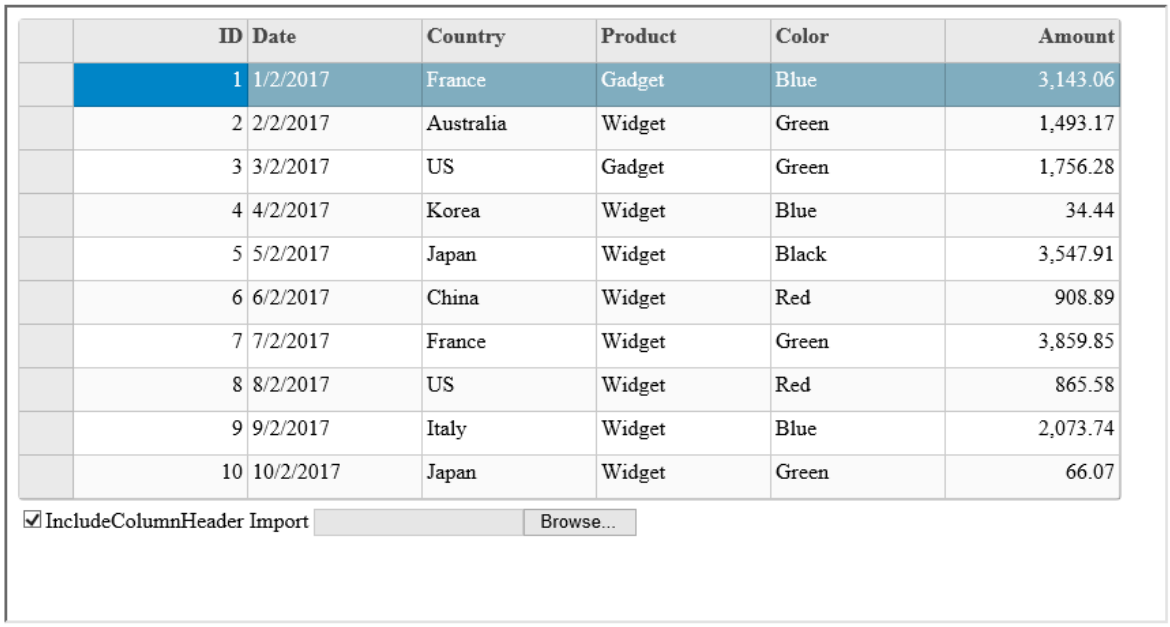
Import Excel into FlexGrid

This section shows how to call the Web API service project through client application and add import function to enable importing Excel data into FlexGrid control.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the FlexGrid appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For further details refer to C1 Web API Client JavaScript) to the project.
2. Add import functionality, using Client JavaScript Helper, which will be triggered on button click event.

MVC

Add the following code to **Views|MVCFlexGrid|Index.cshtml** to import excel data into FlexGrid control.

Index.cshtml copyCode

```
<script type="text/javascript">
    var gridcontrol;
    c1.mvc.Utils.documentReady(function () {
        gridcontrol = wijmo.Control.getControl('#flexGrid')
    });
    function importFlex() {
    var file = document.getElementById("fileInput").files[0];
    var importer = new wijmo.grid.ExcelImporter();
    importer.requestImport(gridcontrol,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/import/excel", file, true)
    }
</script>
```

HTML

Add the following markup in <script> tags to import excel data into FlexGrid control.

JavaScript

```
<script>
var grid;
$(document).ready(function() {
  grid = wijmo.Control.getControl('#TheGrid')
});
function importFlex() {
```

```
  var file = document.getElementById("fileInput").files[0];
  var importer = new wijmo.grid.ExcelImporter();
  importer.requestImport(grid,
  "http://demos.componentone.com/ASPNET/WebAPI/api/import/excel",
  file, true)
}
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.grid.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Provide an input control, for example button, to initiate import request in client application.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for import functionality.

Index.cshtml copyCode

```
<div>
    <span>Import</span>
    <input type="file" value="Import" id="fileInput" class="upload" onchange="importFlex()" />
</div>
```

HTML

Add the following markup within the <body> tags to add button for import functionality.

HTML

```
<div id="TheGrid" style="width:auto"></div> <br />
<input type="file" id="fileInput" class="upload" onchange="importFlex()" />
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

A button to select file to be imported appears along with the control, in the output. Use this button to import data from the desired file into the control.

> Explore detailed demo samples of import service for MVC and Wijmo5 FlexGrid control in:
>
> - MVCFlexGrid Live Demo
> - Wijmo5FlexGrid Live Demo

Back to Top

Generate Excel Service

Generate excel files from given xml, json and dataset/collection in no time with REST based APIs. Additionally, you can easily convert between various workbook formats by consuming REST API service. Source the data- to generate excel and convert between excel format- from remote or local storage, or upload the data from client.

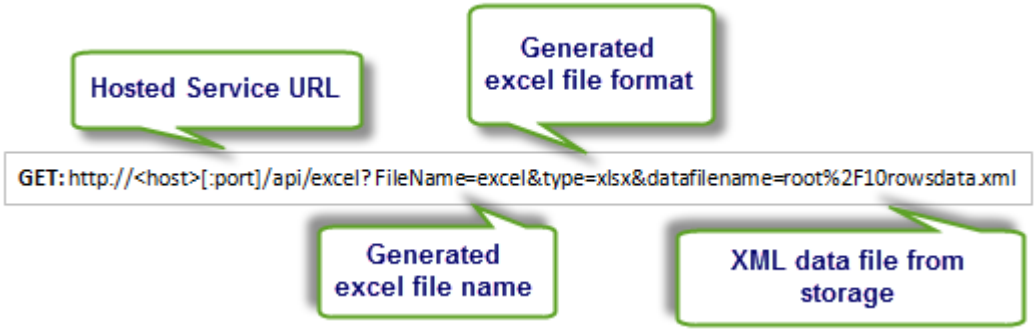
Your client application sends an HTTP request to the Web API service application. This request uses GET and POST methods to seek a response from the service. While, GET method retrieves the intended information from the resource specified, POST method submits the data to the resource.

Generate Excel Request Schema

To generate excel workbook in the desired format from the data that is present in a storage (local or remote), you need to use GET method. In this case, storage location along with file format of generated excel is specified in the request URL, as: **GET: http://[:port]/api/excel?FileName=<>&type=<>&datafilename=<>**

GET Request Schema to Generate Excel from xml

The following illustration depicts a request URL with parameters to generate excel in a desired format, from **xml** data file available in storage.



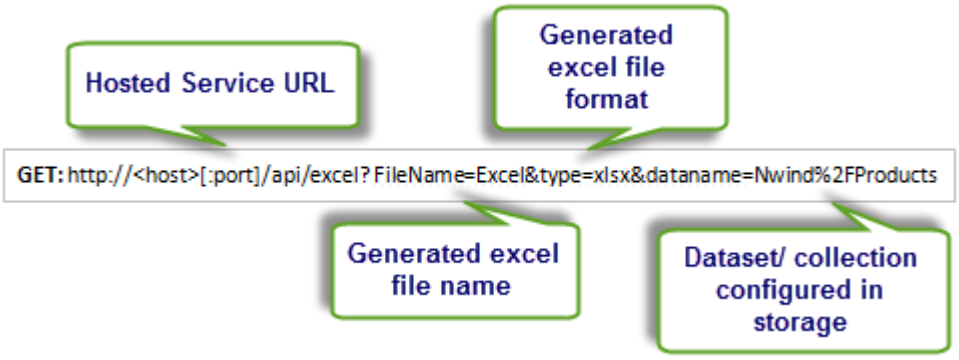
The following table elaborates request URL parameters (to generate excel from xml data file in storage) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the excel, generated through the service.
DataFileName	data file name that storage manager recognizes	xml data file that is available in storage.

GET Request Schema to Generate Excel from Dataset/Collection

The following illustration depicts a request URL with parameters to generate excel in a desired format, from

dataset/collection configured in storage.

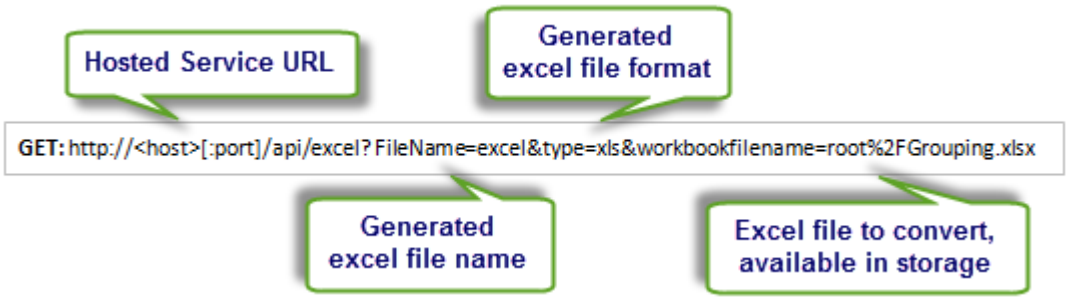


The following table elaborates request URL parameters (to generate excel from dataset/collection configured in storage) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the excel, generated through the service.
DataName	data name that data provider recognizes	Dataset or data collection that is configured in storage.

GET Request Schema to Convert Excel Format

The following illustration depicts a request URL with parameters to **convert excel file** available in storage to a desired format.



The following table elaborates request URL parameters (to convert excel file from storage to a desired format) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the converted excel.
WorkBookFileName	excel file name that storage manager recognizes	Excel file to convert, that is available in storage.

POST Request Schema to Generate Excel from xml Posted from Client

POST method is used if the data does not reside in storage, and is provided through client. Here, the parameters of query string are sent in the HTTP message body of the POST request instead of the URL. The request URL for POST appears as: **POST: http://[:port]/api/excel**

The following table elaborates query parameters for POST request (to generate excel from **xml** data posted from client) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the excel, generated through the service.
DataFile	xml	xml data to be uploaded from client, its content is collection-like (a root element with multiple same elements as its children).

POST Request Schema to Generate Excel from json Posted from Client

The following table elaborates query parameters for POST request (to generate excel from **json** data posted from client) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the excel, generated through the service.
Data	json	json data to be uploaded from client.

POST Request Schema to Convert Excel Format

The following table elaborates query parameters for POST request (to **convert** excel posted from client to a desired format) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	json, xlsx, xls, csv, xml	File format of the converted excel.
WorkbookFile	xls, xlsx, csv	Excel file to convert, to be uploaded from client.

Here, users need not specify the query parameters in the request URL. The following topics (Generate Excel from XML Posted from Client, Generate Excel from JSON Data Posted from Client, and Convert Workbook Formats using Data Posted from Client) discuss how the parameters of query string are sent in the HTTP message body in POST request.

Generate Excel from XML in Storage

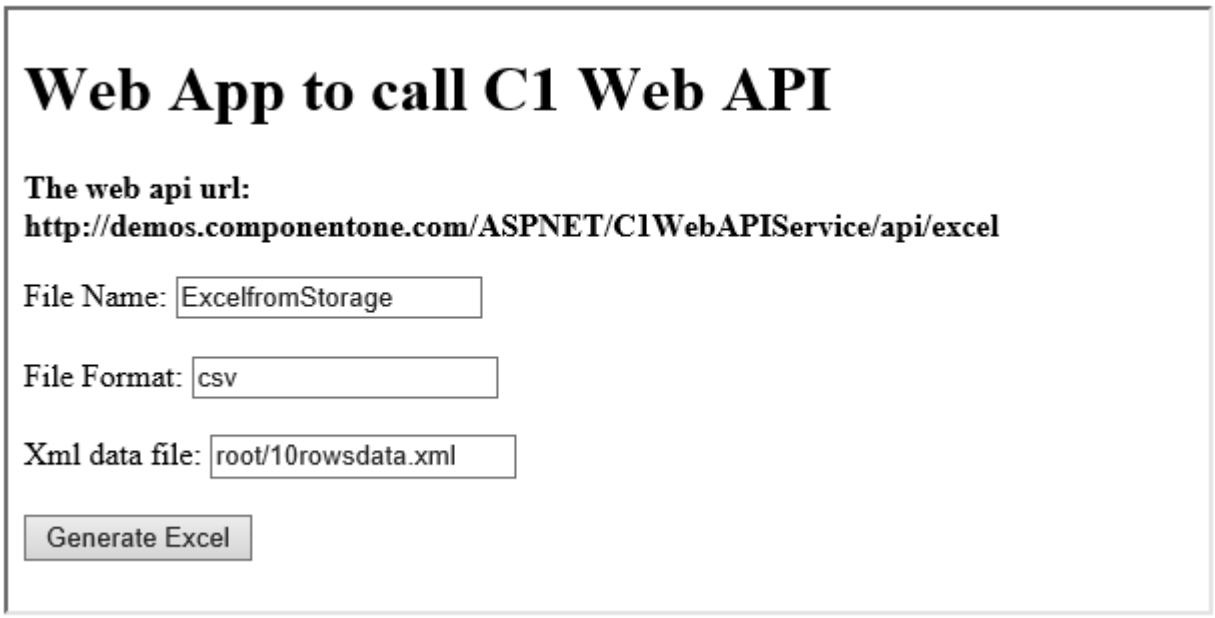
This section demonstrates how to call the Web API service through a client application and generate excel file from the XML string present in storage (remote storage or storage at the same server).

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a GET request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, the service URL takes name and location of XML data file (present in storage) in **DataFileName** parameter and the desired file format, **csv**, in **Type** parameter. The specified XML data file named 10RowsData.xml resides in root folder of the hosted service.

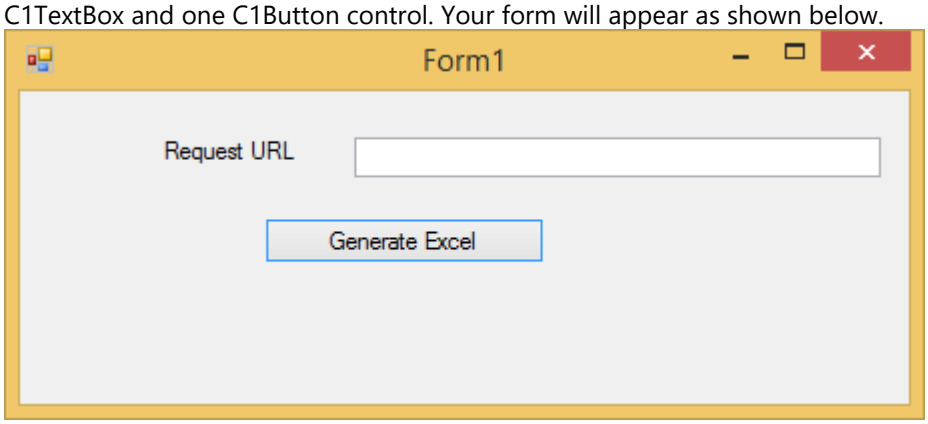


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add one C1Label, one C1TextBox and one C1Button control. Your form will appear as shown below.



2. Define a method (for example: GetExcel**()**) in form class of your WinForms application, to call the service application, as shown below.

C#

```
public void GetExcel() {
  var apiURL = string.IsNullOrEmpty(C1TextBox1.Text) ?
 "http://demos.componentone.com/ASPNET/WebAPI/api/excel?FileName=excel
&type=csv&datafilename=root%2F10rowsdata.xml" : C1TextBox1.Text;
  WebRequest request = WebRequest.Create(apiURL);
  WebResponse response = request.GetResponse();
```

```
  var fileStream = File.Create("D:\\ExcelfromStorage.csv");
  //The file format specified here should be same as that specified in the
request url
  response.GetResponseStream().CopyTo(fileStream);
}
```

3. Call the GetExcel() method on button-click event of **Generate Excel** button.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="GET">
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName"
value="ExcelfromStorage" />
        <br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="csv" />
        <br />
        <label for="datafilename">Xml data file:</label>
        <input type="text" id="datafilename" name="datafilename"
value="root/10rowsdata.xml" />
        <input type="submit" value="Generate Excel"/>
</form>
```

Note that, for GET request we set **method** attribute of <form> tag to GET, and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to generate the excel from XML data file, present in storage, to the desired excel format.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Provide the service URL, along with the query string containing appropriate parameters, in the textbox corresponding to Request URL field.
- Click the **Generate Excel** button. The generated excel data stream will get downloaded at the location specified within the GetExcel() method.

HTML Application

- Save your HTML file and open it in a browser.
- Set the appropriate parameters for the desired excel file format, and click **Generate Excel** button.

Explore detailed demo samples of REST API service to generate excel from XML data file available in storage at:

- Generate Excel Live Demo

Back to Top

Generate Excel from Data Sources in Storage

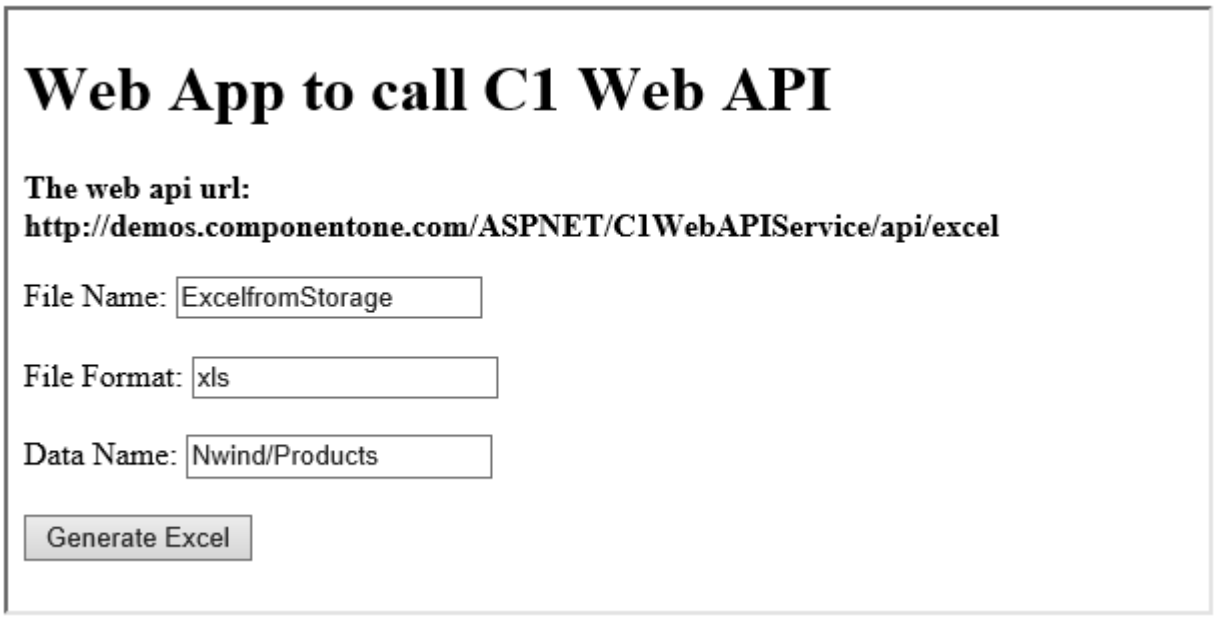
This section demonstrates how to call the Web API service through a client application and generate excel file from dataset or .NET collection present in storage (remote storage or storage at the same server).

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a GET request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, the service URL takes name and location of dataset/ collection (present in storage) in **DataName** parameter and the desired file format, xls, in **Type** parameter. The specified dataset, named Products, resides in Nwind folder of the hosted service.

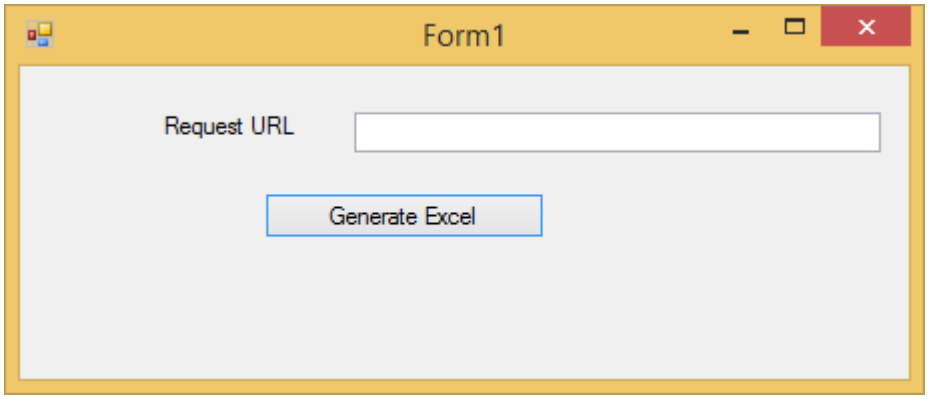


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add one C1Label, one C1TextBox and one C1Button control. Your form will appear as shown below.



2. Define a method (for example: GetExcel**()**) in form class of your WinForms application, to call the service application, as shown below.

C#

```
public void GetExcel() {
  var apiURL = string.IsNullOrEmpty(C1TextBox1.Text) ?
"http://demos.componentone.com/ASPNET/WebAPI/api/excel?
  FileName = excel & type = xls & dataname = Nwind % 2 FProducts " :
C1TextBox1.Text;
  WebRequest request = WebRequest.Create(apiURL);
  WebResponse response = request.GetResponse();
  var fileStream = File.Create("D:\\ExcelfromStorage.xls");
  //The file format specified here should be same as that specified in the
  request url
  response.GetResponseStream().CopyTo(fileStream);
}
```

3. Call the GetExcel() method on button-click event of **Generate Excel** button.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="GET">
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName"
value="ExcelfromStorage" />
        <br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="xls" />
        <br />
        <label for="DataName">Data Name:</label>
        <input type="text" id="DataName" name="DataName" value="Nwind/Products"
/>
        <input type="submit" value="Generate Excel"/>
</form>
```

Note that, for GET request we set **method** attribute of <form> tag to GET, and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to

generate the excel from dataset, present in the storage, to the desired excel format.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Provide the service URL, along with the query string containing appropriate parameters, in the textbox corresponding to Request URL field.
- Click the **Generate Excel** button. The generated excel data stream will get downloaded at the location specified within the GetExcel() method.

HTML Application

- Save your HTML file and open it in a browser.
- Set the appropriate parameters for the desired excel file format, and click **Generate Excel** button.

> Explore detailed demo samples of REST API service to generate excel from data sources available in storage at:
>
> - Generate Excel Live Demo

Back to Top

Convert Workbook Formats using Data from Storage

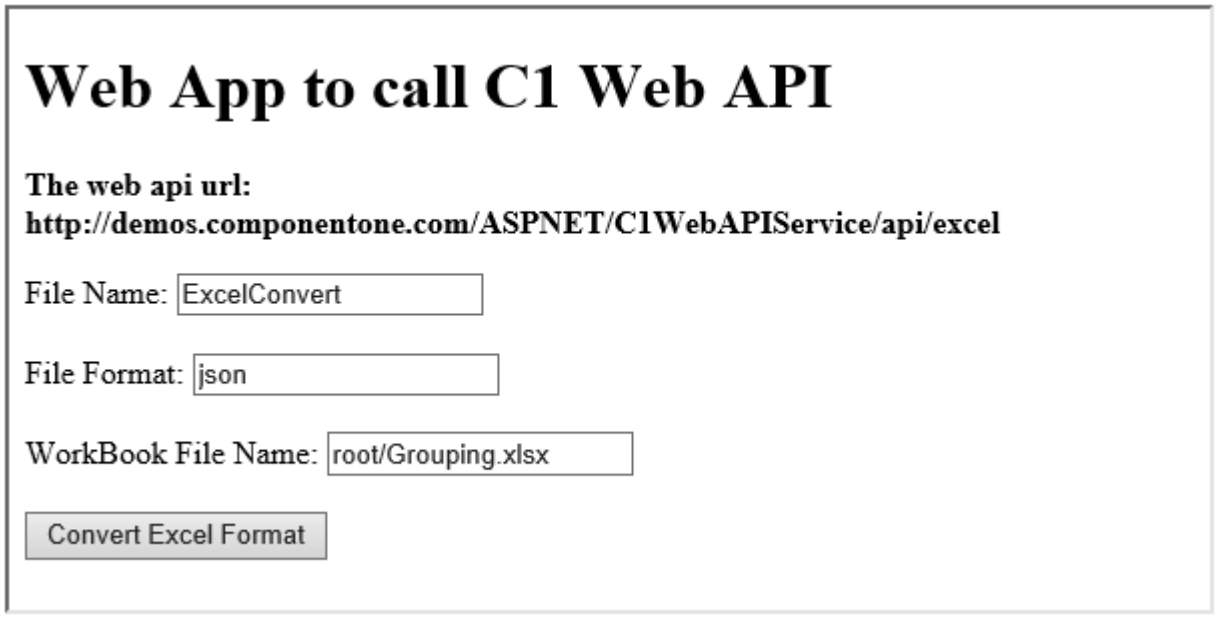
This section demonstrates how to call the Web API service through a client application to convert the excel file available in storage (remote storage or storage at the same server) to a different file format.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a GET request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, the service URL takes name and location of excel workbook (present in storage) in **WorkBookFileName** parameter and the desired file format, json, in **Type** parameter. The specified excel workbook, results.xlsx, resides in root folder of the hosted service.

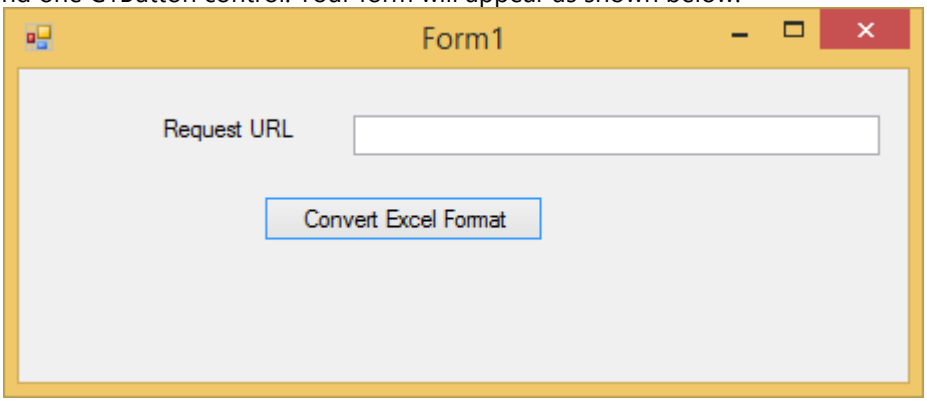


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add one C1Label, one C1TextBox and one C1Button control. Your form will appear as shown below.



2. Define a method (for example: ConvertExcel**()**) in form class of your WinForms application, to call the service application, as shown below.

C#

```
public void ConvertExcel() {
  var apiURL = string.IsNullOrEmpty(C1TextBox1.Text) ?   "http:
//demos.componentone.com/ASPNET/WebAPI/api/excel?
FileName=excel&type=json&workbookfilename=root%2Grouping.xlsx" :
C1TextBox1.Text;
  WebRequest request = WebRequest.Create(apiURL);
  WebResponse response = request.GetResponse();
  var fileStream = File.Create("D:\\ExcelConvert.json");
  response.GetResponseStream().CopyTo(fileStream);
}
```

3. Call the ConvertExcel() method on button-click event of **Convert Excel Format** button.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="GET">
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName" value="ExcelConvert" />
        <br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="json" />
        <br />
        <label for="WorkBookFileName">WorkBook File Name:</label>
        <input type="text" id="WorkBookFileName" name="WorkBookFileName"
value="root/Grouping.xlsx" />
        <input type="submit" value="Convert Excel Format"/>
</form>
```

Note that, for GET request we set **method** attribute of <form> tag to GET, and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to generate the excel in the desired format from the excel workbook, present in the storage.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Provide the service URL, along with the query string containing appropriate parameters, in the textbox corresponding to Request URL field.
- Click the **Convert Excel Format** button. The generated excel data stream will get downloaded at the location specified within the ConvertExcel() method.

HTML Application

- Save your HTML file and open it in a browser.
- Set the appropriate parameters for the desired excel file format, and click **Convert Excel Format** button.

> Explore detailed demo samples of REST API service to convert workbook formats at:
>
> - Convert Excel Format Live Demo

Back to Top

Generate Excel from XML Posted from Client

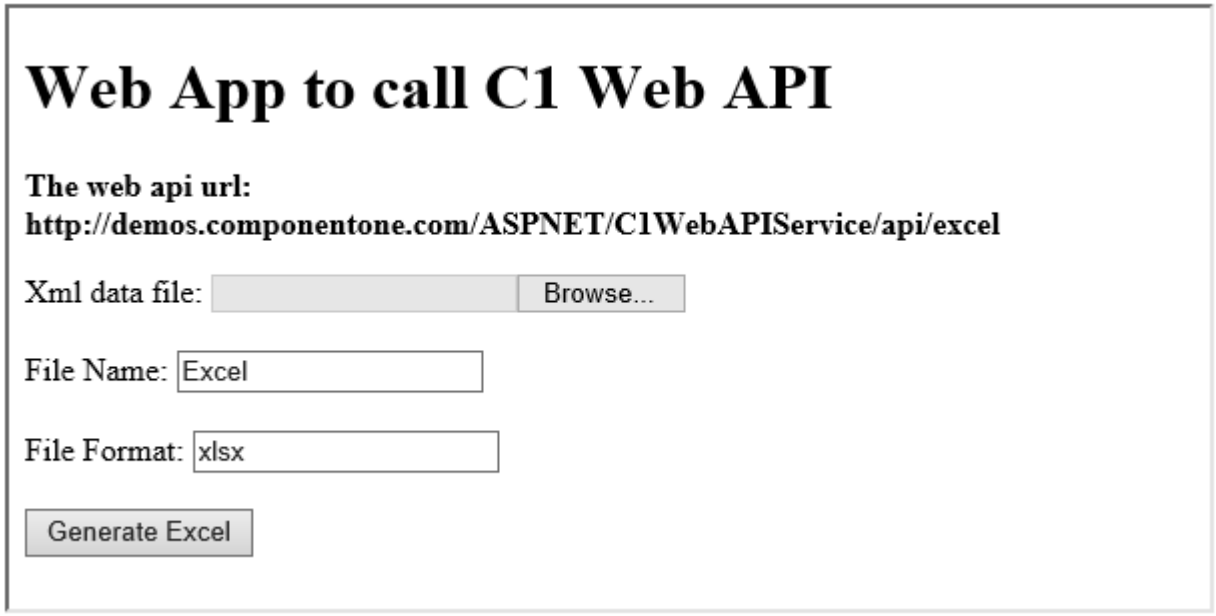
This section demonstrates how to call the Web API service through a client application and generate excel from XML data file, posted from client.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a POST request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, user provides XML data file through the client applications. Moreover, the end user needs to specify the service URL, name and appropriate file format of generated excel file through client project. It is so, because the parameters or query string for generating excel are sent in the HTTP message body of the POST request and not in the request URL.

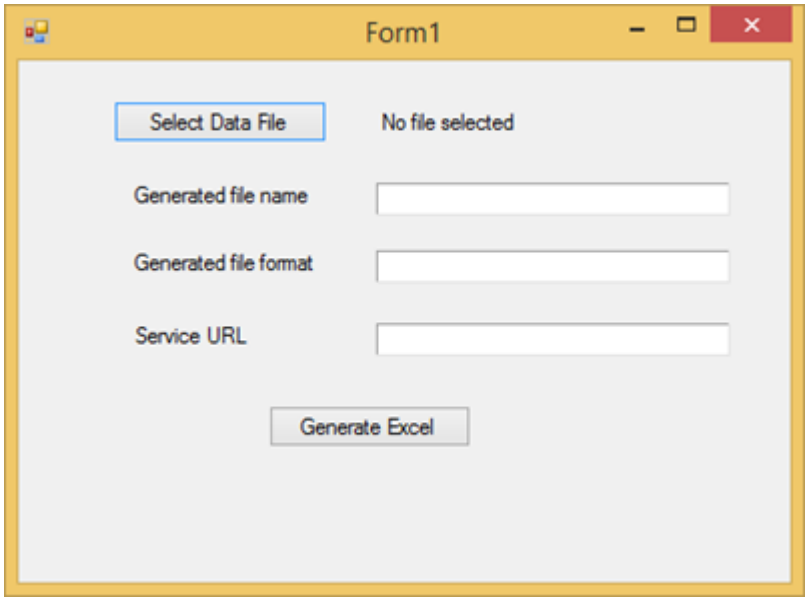


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add four C1Label, three C1TextBox, two C1Button controls and an OpneFileDialog component. Set their text properties so that your form appears as shown below.



2. Add the following code on button-click event of "Select Data File" button.

C#

```
private void button1_Click(object sender, EventArgs e) {
  var result = openFileDialog1.ShowDialog();
  if (result == DialogResult.OK || result == DialogResult.Yes) {
    _filePath = openFileDialog1.FileName;
    if (!string.IsNullOrEmpty(_filePath)) {
      label1.Text = _filePath;
    }
  }
}
```

3. Add the following code on button-click event of "Generate Excel" button.

C#

```
private void button2_Click(object sender, EventArgs e) {
  if (string.IsNullOrEmpty(_filePath)) {
    MessageBox.Show("Invalid response.");
    return;
  }
  using(var client = new HttpClient())
  using(var formData = new MultipartFormDataContent())
  using(var fileStream = File.OpenRead(_filePath)) {
    var fileName = string.IsNullOrEmpty(C1TextBox1.Text) ? "test" :
textBox1.Text;
    var fileFormat = string.IsNullOrEmpty(C1TextBox2.Text) ? "xlsx" :
textBox2.Text;
    formData.Add(new StringContent(fileName), "FileName");
    formData.Add(new StringContent(fileFormat), "FileFormat");
    formData.Add(new StreamContent(fileStream), "DataFile",
Path.GetFileName(_filePath));
    var response = client.PostAsync(C1TextBox3.Text, formData).Result;
    if (!response.IsSuccessStatusCode) {
      MessageBox.Show("Invalid response.");
```

```
      return;
    }
    var tempPath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString());
    if (!Directory.Exists(tempPath)) {
      Directory.CreateDirectory(tempPath);
    }
    var tempFilePath = Path.Combine(tempPath, string.Format("{0}.{1}", fileName,
fileFormat));
    using(var newFile = File.Create(tempFilePath)) {
      response.Content.ReadAsStreamAsync().Result.CopyTo(newFile);
    }

    Process.Start(tempFilePath);
  }
}
```

Note that for POST request, **formData** is used, which is an instance of System.Net.Http.MultipartFormDataContent class. Through code, we add HTTP content to formData and pass it while sending asynchronous POST request to the specified uri.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="POST" enctype="multipart/form-data">
        <label for="dataFile">Xml data file:</label>
        <input type="file" id="dataFile" name="dataFile" accept=".xml" />
        <br/><br/>
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName" value="test"/>
        <br /><br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="xlsx" />
        <br /><br />
        <input type="submit" value="POST"/>
</form>
```

Note that, for POST request we set **method** attribute of <form> tag to POST, its **enctype** attribute to "multipart/form-data" and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to generate the excel file from XML data file posted from client.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

- Click **Select Data File** button to select and post XML data file.
- Specify the name and appropriate file format of the generated excel, in respective text boxes.
- Provide the service URL for POST request (http://demos.componentone.com/ASPNET/C1WebAPIService/api/excel) in respective text box, and click the **Generate Excel** button. The generated excel data file will open.

HTML Application

- **Save** your HTML file and open it in a browser.
- Select and post XML data file.
- Set name and appropriate file format for generated excel, and click **Generate Excel** button.

> Explore detailed demo samples of REST API service to generate excel from XML data file at:
>
> - Generate Excel from XML Live Demo

Back to Top

Generate Excel from JSON Data Posted from Client

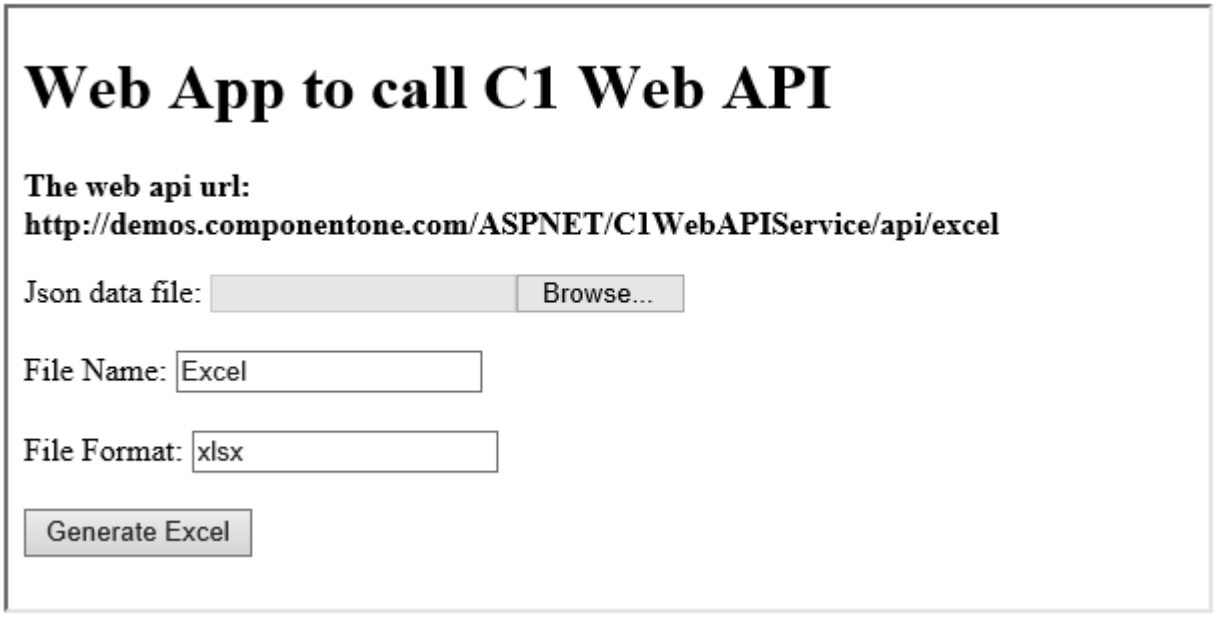
This section demonstrates how to call the Web API service through a client application and generate excel from JSON data file, posted from client.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a POST request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, user provides JSON data file through the client applications. Moreover, the end user needs to specify the service URL, name and appropriate file format of generated excel file through client project. It is so, because the parameters or query string for generating excel are sent in the HTTP message body of the POST request and not in the request URL.

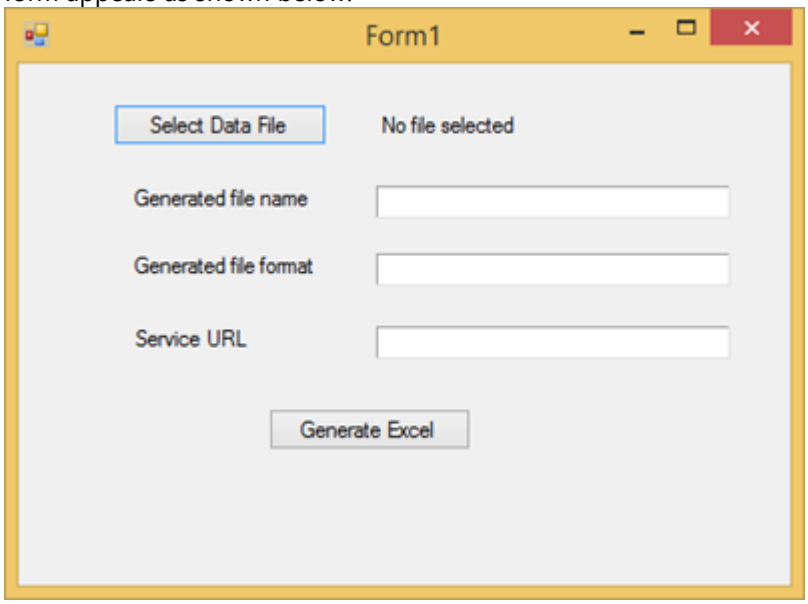


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add four C1Label, three C1TextBox, two C1Button controls and an OpneFileDialog component. Set their text properties so that your form appears as shown below.



2. Add the following code on button-click event of "Select Data File" button.

C#

```
private void button1_Click(object sender, EventArgs e) {
  var result = openFileDialog1.ShowDialog();
  if (result == DialogResult.OK || result == DialogResult.Yes) {
    _filePath = openFileDialog1.FileName;
    if (!string.IsNullOrEmpty(_filePath)) {
      label1.Text = _filePath;
    }
  }
}
```

3. Add the following code on button-click event of "Generate Excel" button.

C#

```
private void button2_Click(object sender, EventArgs e) {
  if (string.IsNullOrEmpty(_filePath)) {
    MessageBox.Show("Invalid response.");
    return;
  }
  using(var client = new HttpClient())
  using(var formData = new MultipartFormDataContent())
  using(var fileStream = File.OpenRead(_filePath)) {
    var fileName = string.IsNullOrEmpty(C1TextBox1.Text) ? "test" :
textBox1.Text;
    var fileFormat = string.IsNullOrEmpty(C1TextBox2.Text) ? "xlsx" :
textBox2.Text;
```

```
    formData.Add(new StringContent(fileName), "FileName");
    formData.Add(new StringContent(fileFormat), "FileFormat");
    formData.Add(new StreamContent(fileStream), "DataFile",
Path.GetFileName(_filePath));
    var response = client.PostAsync(C1TextBox3.Text, formData).Result;
    if (!response.IsSuccessStatusCode) {
      MessageBox.Show("Invalid response.");
      return;
    }
    var tempPath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString());
    if (!Directory.Exists(tempPath)) {
      Directory.CreateDirectory(tempPath);
    }
    var tempFilePath = Path.Combine(tempPath, string.Format("{0}.{1}", fileName,
fileFormat));
    using(var newFile = File.Create(tempFilePath)) {
      response.Content.ReadAsStreamAsync().Result.CopyTo(newFile);
    }

    Process.Start(tempFilePath);
  }
}
```

Note that for POST request, **formData** is used, which is an instance of System.Net.Http.MultipartFormDataContent class. Through code, we add HTTP content to formData and pass it while sending asynchronous POST request to the specified uri.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="POST" enctype="multipart/form-data">
        <label for="data">Json data file:</label>
        <input type="file" id="data" name="data" accept=".json" />
        <br/><br/>
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName" value="test"/>
        <br /><br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="xlsx" />
        <br /><br />
        <input type="submit" value="Generate Excel"/>
</form>
```

Note that, for POST request we set **method** attribute of <form> tag to POST, its **enctype** attribute to "multipart/form-data" and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to generate the excel file from JSON data file posted from client.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Click **Select Data File** button, to select and post JSON data file.
- Specify the name and appropriate file format of the generated excel, in respective text boxes.
- Provide the service URL for POST request (http://demos.componentone.com/ASPNET/WebAPI/api/excel) in respective text box, and click the **Generate Excel** button. The generated excel data file will open.

HTML Application

- **Save** your HTML file and open it in a browser.
- Select and post JSON data file.
- Set name and appropriate file format for generated excel, and click **Generate Excel** button.

> Explore detailed demo samples of REST API service to generate excel from JSON data file at:
>
> - Generate Excel from JSON Live Demo

Back to Top

Convert Workbook Formats using Data Posted from Client

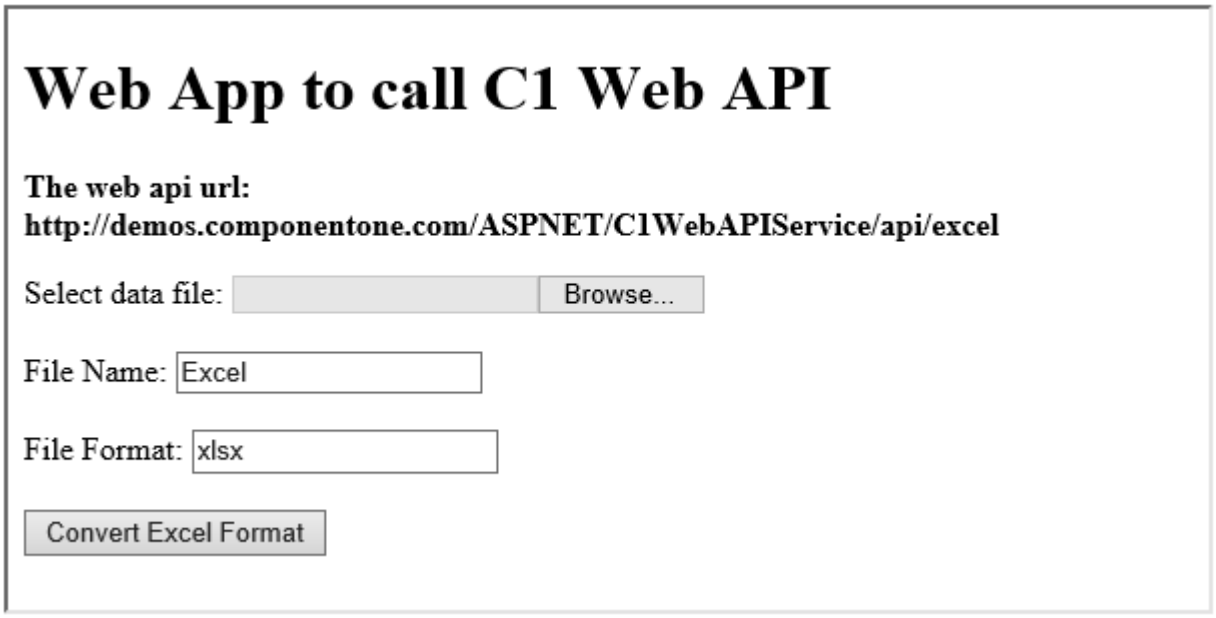
This section demonstrates how to call the Web API service through a client application and convert between excel file formats, posted from client.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a POST request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, user provides excel data file through the client applications. Moreover, the end user needs to specify the service URL, name and appropriate file format of generated excel file through client project. It is so, because the parameters or query string for converting excel formats are sent in the HTTP message body of the POST request and not in the request URL.



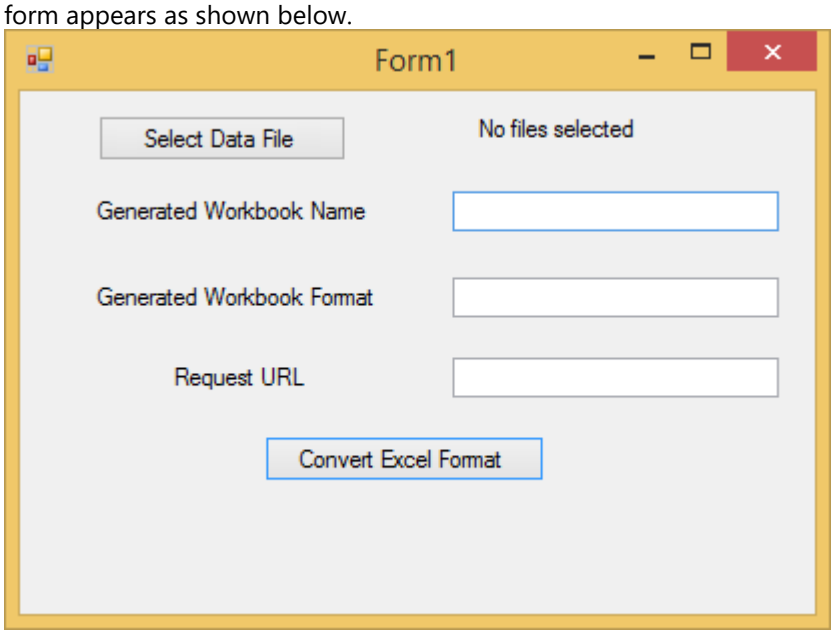
Note: Service can convert xls, xlsx, and csv files posted from the client.

Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add four C1Label, three C1TextBox, two C1Button controls and an OpneFileDialog component. Set their text properties so that your form appears as shown below.



2. Add the following code on button-click event of "Select Data File" button.

C#

```
private void button1_Click(object sender, EventArgs e) {
  var result = openFileDialog1.ShowDialog();
```

```
  if (result == DialogResult.OK || result == DialogResult.Yes) {
    _filePath = openFileDialog1.FileName;
    if (!string.IsNullOrEmpty(_filePath)) {
      label1.Text = _filePath;
    }
  }
}
```

3. Add the following code on button-click event of "Convert Excel Format" button.

C#

```
private void button2_Click(object sender, EventArgs e) {
  if (string.IsNullOrEmpty(_filePath)) {
    MessageBox.Show("Invalid response.");
    return;
  }
  using(var client = new HttpClient())
  using(var formData = new MultipartFormDataContent())
  using(var fileStream = File.OpenRead(_filePath)) {
    var fileName = string.IsNullOrEmpty(C1TextBox1.Text) ? "test" :
textBox1.Text;
    var fileFormat = string.IsNullOrEmpty(C1TextBox2.Text) ? "xlsx" :
textBox2.Text;
    formData.Add(new StringContent(fileName), "FileName");
    formData.Add(new StringContent(fileFormat), "FileFormat");
    formData.Add(new StreamContent(fileStream), "DataFile",
Path.GetFileName(_filePath));
    var response = client.PostAsync(C1TextBox3.Text, formData).Result;
    if (!response.IsSuccessStatusCode) {
      MessageBox.Show("Invalid response.");
      return;
    }
    var tempPath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString());
    if (!Directory.Exists(tempPath)) {
      Directory.CreateDirectory(tempPath);
    }
    var tempFilePath = Path.Combine(tempPath, string.Format("{0}.{1}", fileName,
fileFormat));
    using(var newFile = File.Create(tempFilePath)) {
      response.Content.ReadAsStreamAsync().Result.CopyTo(newFile);
    }
    Process.Start(tempFilePath);
  }
}
```

Note that for POST request, **formData** is used, which is an instance of System.Net.Http.MultipartFormDataContent class. Through code, we add HTTP content to formData and pass it while sending asynchronous POST request to the specified uri.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="POST" enctype="multipart/form-data">
        <label for="WorkbookFile">Select data file:</label>
        <input type="file" id="WorkbookFile" name="WorkbookFile" />
        <br/><br/>
        <label for="FileName">File Name:</label>
        <input type="text" id="FileName" name="FileName" value="Excel"/>
        <br /><br />
        <label for="Type">File Format:</label>
        <input type="text" id="Type" name="type" value="xlsx" />
        <br /><br />
        <input type="submit" value="Convert Excel Format"/>
</form>
```

Note that, for POST request we set **method** attribute of <form> tag to POST, its **enctype** attribute to "multipart/form-data" and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to convert the file format of excel file posted from client.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Click **Select Data File** button, to select and post excel file (xls, xlsx, or csv) to convert.
- Specify the name and appropriate file format of the generated excel, in respective text boxes.
- Provide the service URL for POST request (http://demos.componentone.com/ASPNET/WebAPI/api/excel) in respective text box, and click the **Convert Excel Format** button. The generated excel data file will open.

HTML Application

- **Save** your HTML file and open it in a browser.
- Select and post excel data file.
- Set name and appropriate file format for generated excel, and click **Convert Excel Format** button.

Explore detailed demo samples of REST API service to convert between excel file formats at:

- Convert Excel Format Live Demo

Back to Top

Generate Excel from Workbook Posted from Client

This section demonstrates how to call the Web API service through a client application and generate excel from workbook, posted from client.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a POST request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

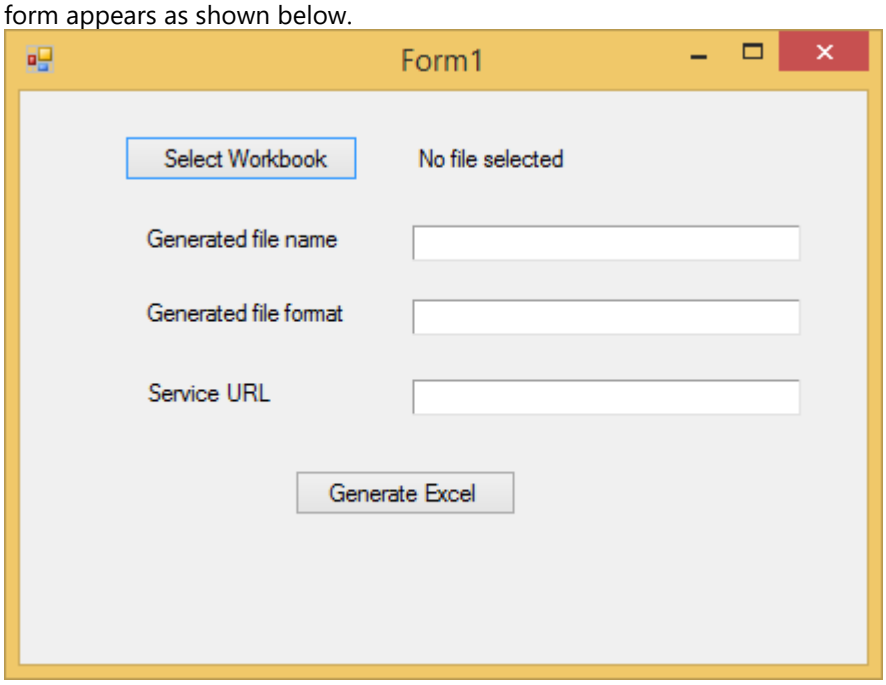
In the following example, user provides workbook through the client applications. Moreover, the end user needs to specify the service URL, name and appropriate file format of generated excel file through client project. It is so, because the parameters or query string for generating excel are sent in the HTTP message body of the POST request and not in the request URL.

Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add four C1Label, three C1TextBox, two C1Button controls and an OpneFileDialog component. Set their text properties so that your form appears as shown below.



2. Add the following code on button-click event of "Select Workbook" button.

C#

```
private void button1_Click(object sender, EventArgs e)        {            var
result = openFileDialog1.ShowDialog();            if (result == DialogResult.OK
|| result == DialogResult.Yes)            {                _filePath =
openFileDialog1.FileName;                if (!string.IsNullOrEmpty(_filePath))
{                    label1.Text = _filePath;                }            }
}
```

3. Add the following code on button-click event of "Generate Excel" button.

C#

```
private void button2_Click(object sender, EventArgs e)        {            if
(string.IsNullOrEmpty(_filePath))            {
```

```
MessageBox.Show("Invalid response.");                return;            }

            using (var client = new HttpClient())            using (var formData
= new MultipartFormDataContent())            using (var fileStream =
File.OpenRead(_filePath))            {                var fileName =
string.IsNullOrEmpty(C1TextBox1.Text) ? "test" : textBox1.Text;
var fileFormat = string.IsNullOrEmpty(C1TextBox2.Text) ? "xlsx" : textBox2.Text;
//formData.Add(new StringContent(fileName), "FileName");
//formData.Add(new StringContent(fileFormat), "FileFormat");
formData.Add(new StreamContent(fileStream), "DataFile",
Path.GetFileName(_filePath));                var response =
client.PostAsync(C1TextBox3.Text, formData).Result;                if
(!response.IsSuccessStatusCode)                {
MessageBox.Show("Invalid response.");                    return;
}                var tempPath = Path.Combine(Path.GetTempPath(),
Guid.NewGuid().ToString());                if (!Directory.Exists(tempPath))
{                    Directory.CreateDirectory(tempPath);                }

                var tempFilePath = Path.Combine(tempPath, string.Format("{0}.
{1}", fileName, fileFormat));                using (var newFile =
File.Create(tempFilePath))                {
response.Content.ReadAsStreamAsync().Result.CopyTo(newFile);                }

                Process.Start(tempFilePath);            }

        }
```

Note that for POST request, **formData** is used, which is an instance of System.Net.Http.MultipartFormDataContent class. Through code, we add HTTP content to formData and pass it while sending asynchronous POST request to the specified uri.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel"
method="POST" enctype="multipart/form-data">            <label
for="dataFile">Xml data file:</label>            <input type="file"
id="dataFile" name="dataFile" accept=".xml" />            <br/>
<label for="fileName">File Name:</label>            <input type="text"
id="fileName" name="fileName" value="test"/>            <br />            <label
for="fileFormat">File Format:</label>            <input type="text"
id="fileFormat" name="type" value="xlsx" />            <br />            <input
type="submit" value="POST"/>        </form>
```

Note that, for POST request we set **method** attribute of <form> tag to POST, its **enctype** attribute to "multipart/form-data" and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to generate the excel file from XML data file posted from client.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Click **Select Data File** button, to select and post workbook.
- Specify the name and appropriate file format of the generated excel, in respective text boxes.
- Provide the service URL for POST request (http://demos.componentone.com/ASPNET/WebAPI/api/excel) in respective text box, and click the **Generate Excel** button. The generated excel data file will open.

HTML Application

- **Save** your HTML file and open it in a browser.
- Select and post workbook.
- Set name and appropriate file format for generated excel, and click **Generate Excel** button.

> Explore detailed demo samples of REST API service to generate excel from workbook at:
>
> - Generate Excel from workbook Live Demo

Back to Top

Merge Excel Service

Merge multiple excel files into workbook with REST Api service provided by Web API Edition. The service merges multiple excel files into a supported workbook format- json, xlsx, xls, and xml.

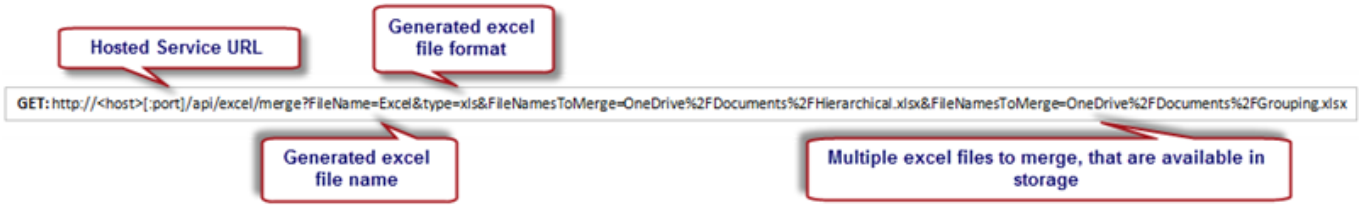
To merge multiple excel files, use GET method if the files to merge reside on a configured storage, else use POST method if you provide the files (to merge) from client.

Merge Excel Request Schema

The storage location along with name and file format of the merged excel is specified in the request URL for GET request, as:
GET: http://[:port]/api/excel/merge?FileName=<>&type=<>&FileNamesToMerge=<>&FileNamesToMerge

GET Request Schema to Merge Multiple Excel Files

The following illustration depicts a request URL with parameters to **merge** multiple excel files available in configured storage.



The following table shows request URL parameters (to merge multiple excel files, available in storage) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified

		by the user.
Type	xlsx, xls, xml	File format of the merged excel file.
FileNamesToMerge	Excel file names that storage manager recognizes	Excel files to merge, available in storage.

POST Request Schema to Merge Multiple Excel Files

POST request is appropriate if you are providing the excel files, to merge, through client. The request URL for POST appears as: **POST: http://[:port]/api/excel/merge**

The following table shows query parameters for POST request (to merge multiple excel files, posted from client) and their respective supported values.

Parameter	Values Supported	Description
FileName	String	Name of the generated excel file, to be specified by the user.
Type	xlsx, xls, xml	File format of the merged excel file.
FilesToMerge	xlsx, xls, and csv	Excel files to merge, to be uploaded from client.

Here, users need not specify the query parameters in the request URL. The topic Merge Multiple Excel Files Posted from Client discusses how the parameters of query string are sent in the HTTP message body in POST request.

Merge Multiple Excel Files Present in Storage

This section demonstrates how to call the Web API service through a client application and merge multiple excel files available in file storage (remote storage or storage at the same server) to workbook.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a GET request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

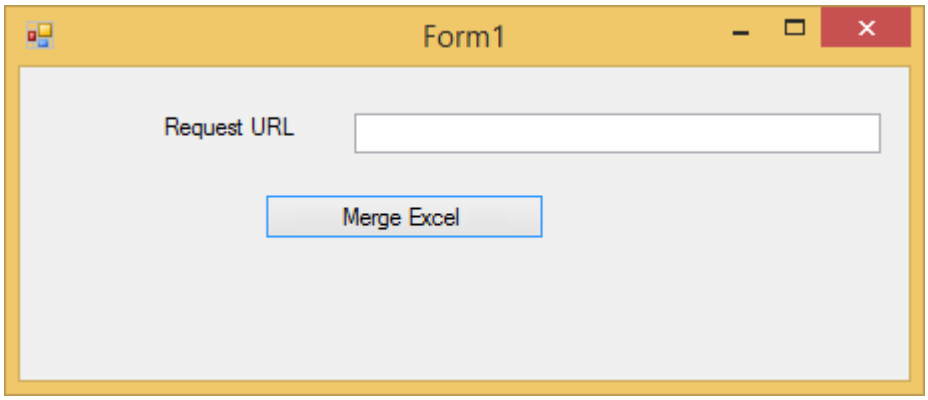
In the following example, the service URL takes name and location of excel data files (present in storage) to merge in **FileNamesToMerge** parameter and the desired file format, xls, in **Type** parameter. The specified excel files to merge, reside in root folder of the hosted service.

Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add one C1Label, one C1TextBox and one C1Button control. Your form will appear as shown below.



2. Define a method (for example: MergeExcel**()**) in form class of your WinForms application, to call the service application, as shown below.

C#

```
public void MergeExcel() {
  var apiURL = "http://demos.componentone.com/ASPNET/WebAPI/api/excel/merge?
  FileName=excel&type=xls&FileNamesToMerge=root%%2FHouston.xlsx";
  WebRequest request = WebRequest.Create(apiURL);
  WebResponse response = request.GetResponse();
  var fileStream = File.Create("D:\\MergedFile.xls");
  response.GetResponseStream().CopyTo(fileStream);
}
```

3. Call the MergeExcel() method on button-click event of **Merge Excel** button.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel/merge"
method="GET">
        <label for="fileName">File Name:</label>
        <input type="text" id="fileName" name="fileName" value="MergedFile" />
        <br /><br />
        <label for="fileFormat">File Format:</label>
        <input type="text" id="fileFormat" name="type" value="xls" />
        <br /><br />
        <label for="FileNamesToMerge">File Names to Merge:</label>
        <input type="text" id="FileNamesToMerge" name="FileNamesToMerge"
value="root/GAS.xls" />
        <input type="text" id="FileNamesToMerge" name="FileNamesToMerge"
value="root/Houston.xlsx" />
        <input type="submit" value="Merge Excel"/>
</form>
```

Note that, for GET request we set **method** attribute of <form> tag to GET, and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to merge multiple excel files, available in the storage, to the desired excel format.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Provide the service URL, along with the query string containing appropriate parameters, in the textbox corresponding to **Request URL** field.
- Click the **Merge Excel** button. The merged excel data stream will get downloaded at the location specified within the MergeExcel() method.

HTML Application

- Save your HTML file and open it in a browser.
- Set name and appropriate excel file format, and click **Merge Excel** button.

Explore detailed demo samples of REST API service to merge multiple excel files at:

- Merge Excel Live Demo

Back to Top

Merge Multiple Excel Files Posted from Client

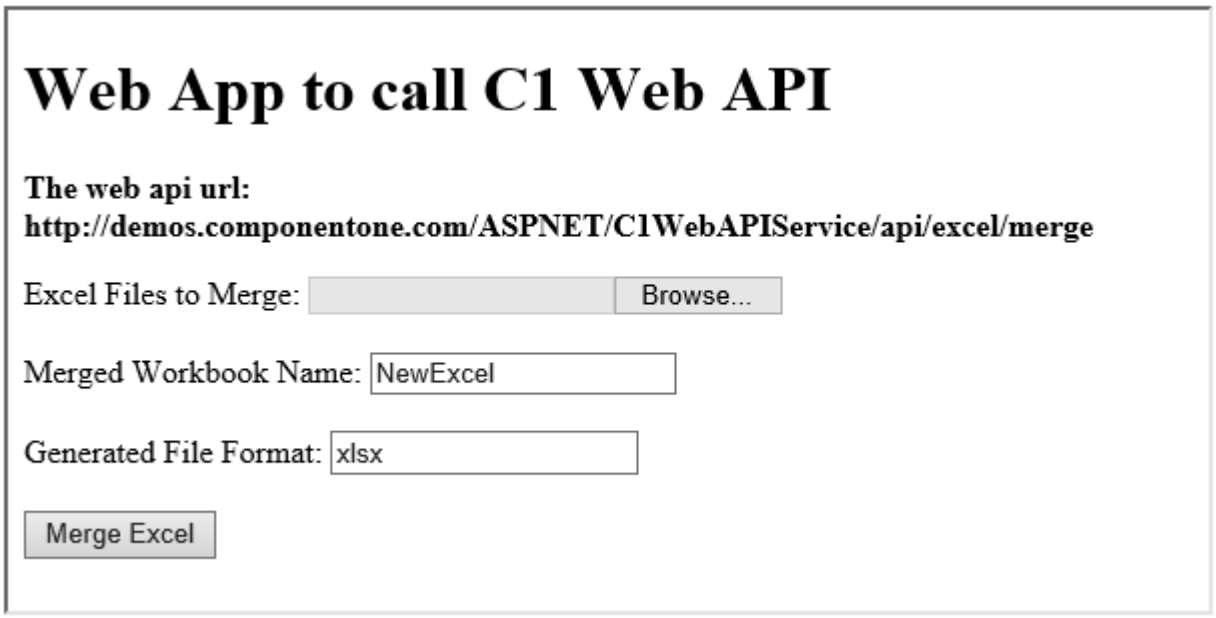
This section demonstrates how to call the Web API service through a client application and merge multiple excel files, that are posted from client, into workbook.

Step 1: Call the Service

Step 2: Run the Client Project

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a POST request to the service, which returns a response stream. This response stream is then saved in the desired excel file format.

In the following example, user provides the excel files (to merge into workbook) through the client applications. Moreover, the end user needs to specify the service URL, name and desired file format of workbook (generated upon merging multiple excel sheets) through client project. It is so, because the parameters or query string for merging excel are sent in the HTTP message body of the POST request and not in the request URL.



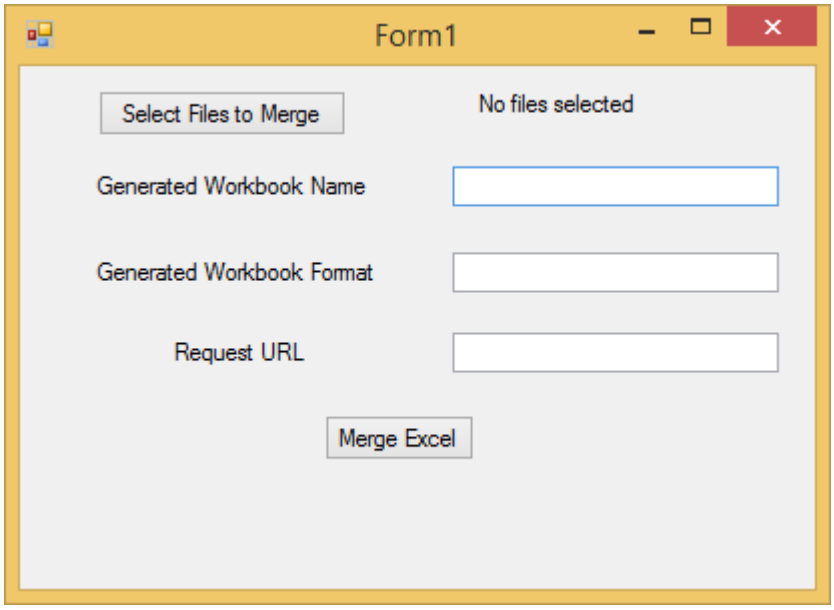
Note: Service can merge xls, xlsx, and csv files posted from the client.

Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add four C1Label, three C1TextBox, two C1Button controls and an OpneFileDialog component. Set their text properties so that your form appears as shown below.



2. Add the following code on button-click event of "Select Files to Merge" button.

C#

```
private void button1_Click(object sender, EventArgs e) {
  OpenFileDialog x = new OpenFileDialog();
  x.Multiselect = true;
  x.ShowDialog();
  result = x.FileNames;
  label1.Text = "Files Selected";
}
```

3. Add the following code on button-click event of "Merge Excel" button.

C#

```
private void button2_Click(object sender, EventArgs e) {
  if (result.Length & lt; = 0) {
    MessageBox.Show("Please select excel files to merge.");
    return;
  }
  using(var client = new HttpClient())
  using(var formData = new MultipartFormDataContent())
  using(var fileStream = File.OpenRead(result[0])) {
    using(var fileStream1 = File.OpenRead(result[1])) {
      var fileName = string.IsNullOrEmpty(txtFileName.Text) ? "test" :
txtFileName.Text;
      var fileFormat = string.IsNullOrEmpty(txtFileFormat.Text) ? "xlsx" :
txtFileFormat.Text;
      formData.Add(new StringContent(fileName), "FileName");
      formData.Add(new StringContent(fileFormat), "Type");
      formData.Add(new StreamContent(fileStream), "filesToMerge", result[0]);
      formData.Add(new StreamContent(fileStream1), "filesToMerge", result[1]);
      var response = client.PostAsync(txtUrl.Text, formData).Result;
      if (!response.IsSuccessStatusCode) {
        MessageBox.Show("Get invalid response.");
        return;
```

```
        }
        var tempPath = Path.Combine(Path.GetTempPath(),
Guid.NewGuid().ToString());
        if (!Directory.Exists(tempPath)) {
          Directory.CreateDirectory(tempPath);
        }
        var tempFilePath = Path.Combine(tempPath, string.Format("{0}.{1}",
fileName, fileFormat));
        using(var newFile = File.Create(tempFilePath)) {
          response.Content.ReadAsStreamAsync().Result.CopyTo(newFile);
        }
        Process.Start(tempFilePath);
      }
    }
}
```

Note that for POST request, **formData** is used, which is an instance of System.Net.Http.MultipartFormDataContent class. Through code, we add HTTP content to formData and pass it while sending asynchronous POST request to the specified uri.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/excel/merge"
method="POST" enctype="multipart/form-data">
        <label for="FilesToMerge">Excel Files to Merge:</label>
    <input type="file" id="FilesToMerge" name="FilesToMerge" multiple/>
    <br /><br />
    <label for="FileName">Merged Workbook Name:</label>
    <input type="text" id="fileName" name="FileName" value="Excel"/>
    <br /><br />
    <label for="Type">Generated File Format:</label>
    <input type="text" id="Type" name="Type" value="xlsx" />
    <br /><br />
    <input type="submit" value="Merge Excel"/>
</form>
```

Note that, for POST request we set **method** attribute of <form> tag to POST, its **enctype** attribute to "multipart/form-data" and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various parameters to merge multiple excel files, provided by client, to the desired workbook format.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.

- Press **F5** to run the project.
- Click **Select Files to Merge** button, to select multiple excel files.

  > **Note:** Hold down CTRL or SHIFT keys while selecting multiple excel files.

- Specify the name and appropriate file format of the generated workbook, in respective text boxes.
- Provide the service URL for POST request (http://demos.componentone.com/ASPNET/WebAPI/api/excel/merge) in respective text box, and click the **Merge Excel** button. The merged excel data file will open.

HTML Application

- **Save** your HTML file and open it in a browser.
- Select and post excel files to merge.

  > **Note:** Hold down CTRL or SHIFT keys while selecting multiple excel files.

- Set name and appropriate file format for merged excel, and click **Merge Excel** button.

> Explore detailed demo samples of REST API service to merge multiple excel files at:
>
> - Merge Excel Live Demo

Back to Top

Split Excel File

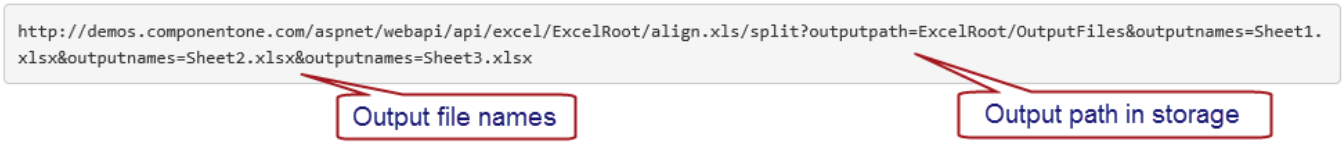
Excel services allows the user to perform various operations on an excel sheet by creating a C1 Web API service application. Split excel file feature allows the user to split an excel file from storage to multiple excel files and save it into storage.

Your client application sends an HTTP request to the Web API service application. This request uses GET and POST methods to seek a response from the service. While, GET method retrieves the intended information from the resource specified, POST method submits the data to the resource.

To split an excel file using output path and output names, you need to use a **GET** method. You can specify the request URL, as

GET: http://<host>[port]/api/excel/{excel path}/split

The following illustration depicts a request URL with parameters to split an excel file.



The following table elaborates request URL parameters required to split an excel file.

Parameter	Values Supported	Description
ExcelPath	xls, xlsx	The excel file name that storage manager can recognize.
Output Path	String	The output path in storage(if not provide, the default output path same as source).
Output Names	String	The output file names (if not provide, the output file names will be generated automatically).

Find/Replace Text

Excel Web API services allows the user to perform basic text operations such as find a text in an excel sheet and replace a text in an excel sheet.

To use the Find/Replace text feature in an excel file, you need to use a **GET** method.

Find a Text

To find a text in an excel sheet, you need to use the **GET** method.

GET: http://<host>[port]/api/excel/{excel path}/{sheet name}/find

The following illustration depicts a request URL with parameters to find a text in an Excel file.

The following table elaborates request URL parameters required to find a text in an excel file.

Parameter	Values Supported	Description
ExcelPath	xls, xlsx	The excel file name that storage manager can recognize.
Sheet Name	String	The sheet name (if not provide, find in all sheets).
Text	String	Text you want to search in the excel file.
Match Case	String	Text you want to use for Match Case.
Whole Cell	String	Text you want to find in whole cell.

Once you run the application, it will match the text provided in the URL and you will see the following elements in the output.

- Sheet name
- Row index
- Cell index
- Start index

Replace a Text

To replace a text in an excel sheet, you need to use the **GET** method.

GET: http://<host>[port]/api/excel/{excel path}/{sheet name}/replace

The following illustration depicts a request URL with parameters to replace a text in an Excel file.

The following table elaborates request URL parameters required to replace a text in an excel file.

Parameter	Values Supported	Description
ExcelPath	xls, xlsx	The excel file name that storage manager can recognize.
Sheet Name	String	The sheet name (if not provide, find in all sheets).
Text	String	Text you want to search in the excel file.
New Text	String	Text you want to replace.
Match Case	Sting	Text you want to use for match case.
Whole Cell	String	Text you want to find in whole cell.

Once you run the application, it will replace the text with new text specified in the request URL.

Add/Delete Row

Excel Web API service allows the user to add, update or delete rows from an excel sheet. Your client application sends an HTTP request to the Web API service application. This request uses GET and POST methods to seek a response from the service. While, GET method retrieves the intended information from the resource specified, POST method submits the data to the resource.

Excel Path, **Sheet Name**, and **Row Indexes** are common parameters for all the three operations specified below.

Add Row

To add a row in an excel sheet, you need to use a **POST** method.

POST: http://<host>[port]/api/excel/{excel path}/{sheet name}/rows/{row index}

The following illustration depicts a request URL with parameters to Add/Delete Row in an Excel file.

The following table elaborates request URL parameters required to Add/Delete Row in an Excel file.

Parameter	Values Supported	Description
ExcelPath	xls, xlsx	The excel file name that storage manager can recognize.
Sheet Name	String	The sheet name.
Row Index	Integer	The row index.

Update Rows

Allows the user to update the rows in an excel sheet, you can also use this REST API to hide/unhide or group/ungroup rows. To update a row in an excel sheet, you need to use the **PUT** method.

PUT: http://<host>[port]/api/excel/{excel path}/{sheet name}/rows/{row indexes}

> **Note:** You can also update the row properties using the same Excel API service by specifying the row property name and value. Available properties are **Visible(Boolean)** and **OutlineLevel(int)**.

Delete Rows

Allows the user to delete a row in an excel sheet. To delete a row in an excel sheet, you need to use the **DELETE**

method.

DELETE: http://<host>[port]/api/excel/{excel path}/{sheet name}/rows/{row indexes}

Add/Delete Columns

Excel Web API service allows the user to add, update or delete columns from an excel sheet. Your client application sends an HTTP request to the Web API service application. This request uses GET and POST methods to seek a response from the service. While, GET method retrieves the intended information from the resource specified, POST method submits the data to the resource.

Excel Path, **Sheet Name**, and **Column Indexes** are common parameters for all the three operations specified below.

Add Column

To add a column in an excel sheet, you need to use a **POST** method.

POST: http://<host>[port]/api/excel/{excel path}/{sheet name}/columns/{column index}

The following illustration depicts a request URL with parameters to Add/Delete Column in an Excel file.

The following table elaborates request URL parameters required to Add/Delete Column in an Excel file.

Parameter	Values Supported	Description
ExcelPath	xls, xlsx	The excel file name that storage manager can recognize.
Sheet Name	String	The sheet name.
Column Index	Integer	The column index.

Update Columns

Allows the user to update the columns in an excel sheet, you can also use this REST API to hide/unhide or group/ungroup columns. To update a column in an excel sheet, you need to use the **PUT** method.

PUT: http://<host>[port]/api/excel/{excel path}/{sheet name}/columns/{column indexes}

> **Note:** You can also update the column properties using the same Excel API service by specifying the column property name and value. Available properties are **Visible(Boolean)** and **OutlineLevel(int)**.

Delete Columns

Allows the user to delete a column in an excel sheet. To delete a column in an excel sheet, you need to use the **DELETE** method.

DELETE: http://<host>[port]/api/excel/{excel path}/{sheet name}/columns/{column indexes}

Image Services

Web API Studio Edition supports image export services for MVC and Wijmo 5 controls. Available as Visual Studio template, C1 Web API enables you to create Web API service on Visual Studio. Client applications then send a request to the Web API service application to export MVC and Wijmo 5 controls as an image. The service supports image

export to PNG, JPG, BMP, TIFF, and GIFF formats.

While you are working with the image services for exporting MVC or Wijmo5 controls, you need to add phantomjs.exe file to your service application. For more information, see How to add phantomjs in your Visual Studio application.

Export Services

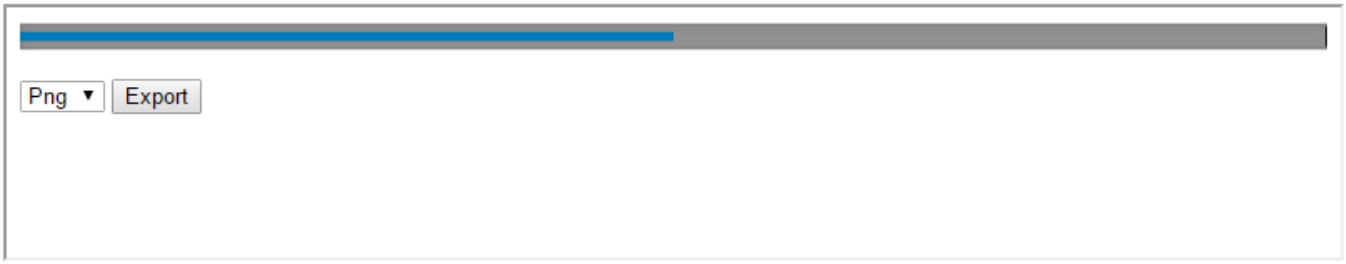
Export BulletGraph to Image

This section shows how to call the Web API service project through a client application and add an export function for exporting BulletGraph control as an image.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the BulletGraph appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For detailed steps refer to Adding Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views|BulletGraph|Index.cshtml** to export the BulletGraph control to image.

Index.cshtml copyCode

```
<script type = "text/javascript" >
  function exportImage() {
    var exporter = new wijmo.gauge.ImageExporter();
    imageType = document.getElementById("mySelect").value;
    control = wijmo.Control.getControl('#bulletGraph');
    exporter.requestExport(control,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/image", {
      fileName: "exportBulletGraph",
      type: imageType,
    });
  }
</script>
```

HTML

Add the following markup in <script> tags to export BulletGraph control to image.

JavaScript

```
<script type="text/javascript">
    function exportImage() {
        var exporter = new wijmo.gauge.ImageExporter();
        var bulletGraph = wijmo.Control.getControl("#BulletGraph");
        exporter.requestExport(bulletGraph,
"http://demos.componentone.com/ASPNET/WebAPI/api/export/image", {
            fileName: "exportBulletGraph",
            type: Png,
        });
    }
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.gauge.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | BulletGraph | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<select id="mySelect">
    <option selected>Png</option>
    <option>Jpg</option>
    <option>Gif</option>
    <option>Bmp</option>
```

```
    <option>Tiff</option>
</select>
<button onclick="exportImage()" title="Export">Export</button>
```

HTML

Add the following markup within the <body> tags to create button for export function.

HTML

```
<button onclick="exportImage()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the control as image.

> Explore detailed demo samples of image export service for MVC and Wijmo5 BulletGraph control in:
>
> - MVCBulletGraph Live Demo
> - Wijmo5BulletGraph Live Demo

Back to Top

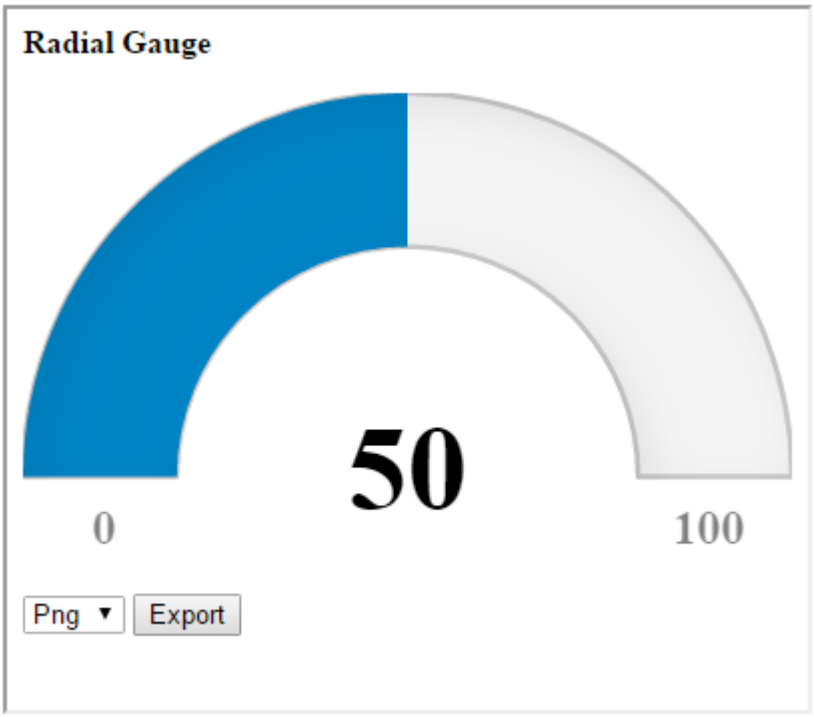
Export RadialGauge to Image

This section shows how to call the Web API service project through a client application and add an export function for exporting RadialGauge control as an image.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the RadialGauge appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For detailed steps refer to Adding Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views|RadialGauge|Index.cshtml** to export the RadialGauge control to image.

Index.cshtml copyCode

```
<script type="text/javascript">
    function exportImage() {
    var exporter = new wijmo.gauge.ImageExporter();
    imageType = document.getElementById("mySelect").value;
    control = wijmo.Control.getControl('#radialGauge');
    exporter.requestExport(control,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/image", {
        fileName: "exportRadialGauge",
        type: imageType,
    });
}
</script>
```

HTML

Add the following markup in <script> tags to export RadialGauge control to image.

JavaScript

```
<script type="text/javascript">
    function exportImage() {
        var exporter = new wijmo.gauge.ImageExporter();
        var radialGauge = wijmo.Control.getControl("#RadialGauge");
        exporter.requestExport(radialGauge, "
http://demos.componentone.com/ASPNET/WebAPI/api/export/image", {
            fileName: "exportRadialGauge",
            type: Png,
        });
    }
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.gauge.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | RadialGauge | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<select id="mySelect">
    <option selected>Png</option>
    <option>Jpg</option>
    <option>Gif</option>
    <option>Bmp</option>
    <option>Tiff</option>
</select>
<button onclick="exportImage()" title="Export">Export</button>
```

HTML

Add the following markup within the <body> tags to create button for export function.

HTML

```
<button onclick="exportImage()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the control as image.

> Explore detailed demo samples of image export service for MVC and Wijmo5 RadialGauge control in:
>
> - MVCRadialGauge Live Demo
> - Wijmo5RadialGauge Live Demo

Back to Top

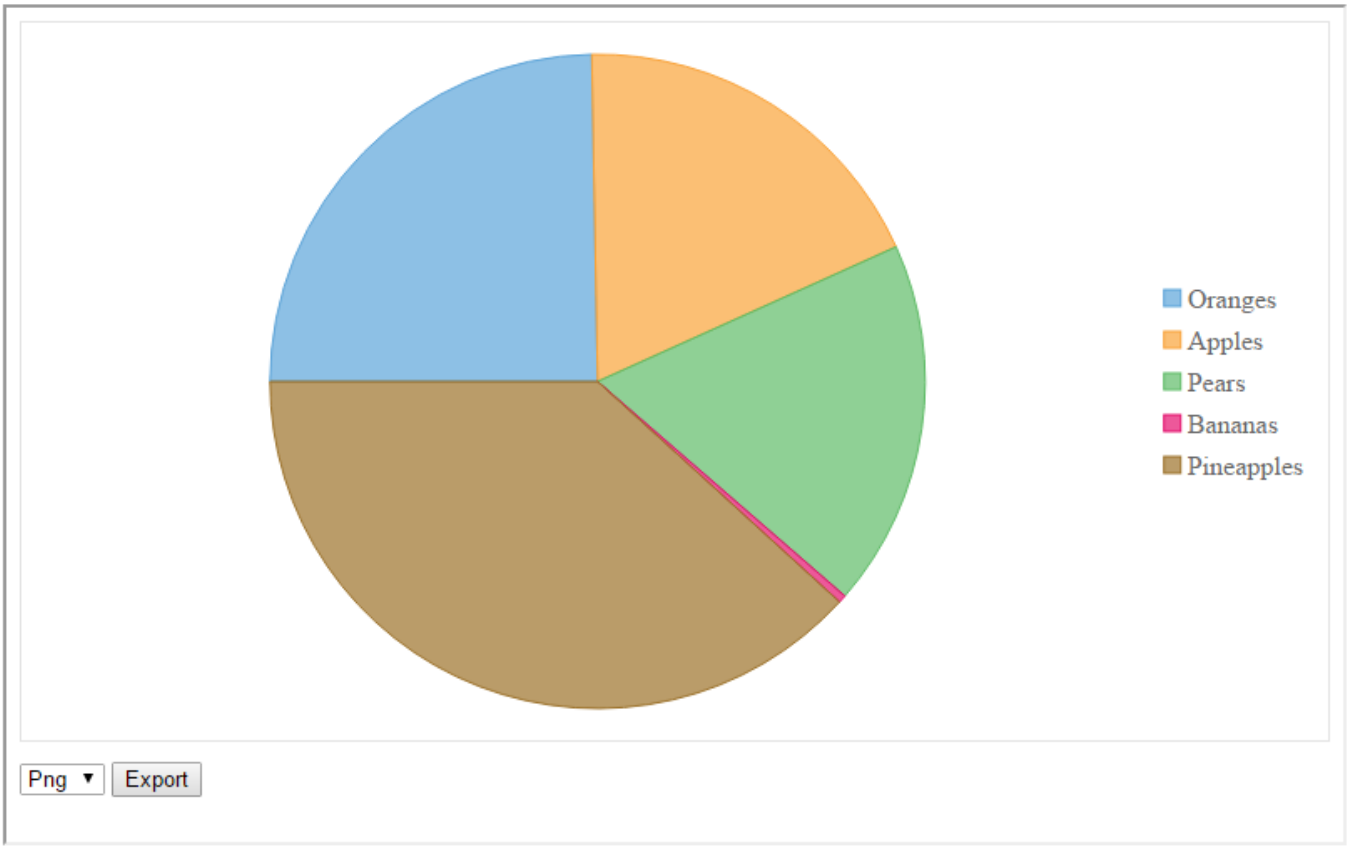
Export FlexPie to Image

This section shows how to call the Web API service project through a client application and add an export function for exporting FlexPie control as an image.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the FlexPie appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For detailed steps refer to Adding Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views|FlexPie|Index.cshtml** to export the FlexPie control to image.

Index.cshtml copyCode

```
<script type="text/javascript">
function exportImage() {
    var exporter = new wijmo.chart.ImageExporter();
    imageType = document.getElementById("mySelect").value;
    control = wijmo.Control.getControl('#flexPie');
    exporter.requestExport(control,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/image", {
        fileName: "exportFlexPie",
        type: imageType,
    });
}
</script>
```

HTML

Add the following markup in <script> tags to export FlexPie control to image.

JavaScript

```
<script type="text/javascript">
    function exportImage() {
        var exporter = new wijmo.chart.ImageExporter();
        var control = wijmo.Control.getControl("#FlexPie");
        exporter.requestExport(control, "
http://demos.componentone.com/ASPNET/WebAPI/api/export/image", {
            fileName: "exportFlexPie",
            type: Png,
        });
    }
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.chart.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | FlexPie | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<select id="mySelect">
    <option selected>Png</option>
    <option>Jpg</option>
    <option>Gif</option>
    <option>Bmp</option>
    <option>Tiff</option>
```

```
</select>
<button onclick="exportImage()" title="Export">Export</button>
```

HTML

Add the following markup within the <body> tags to create button for export function.

HTML

```
<button onclick="exportImage()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the control as image.

> Explore detailed demo samples of image export service for MVC and Wijmo5 FlexPie control in:
>
> - MVCFlexPie Live Demo
> - Wijmo5FlexPie Live Demo

Back to Top

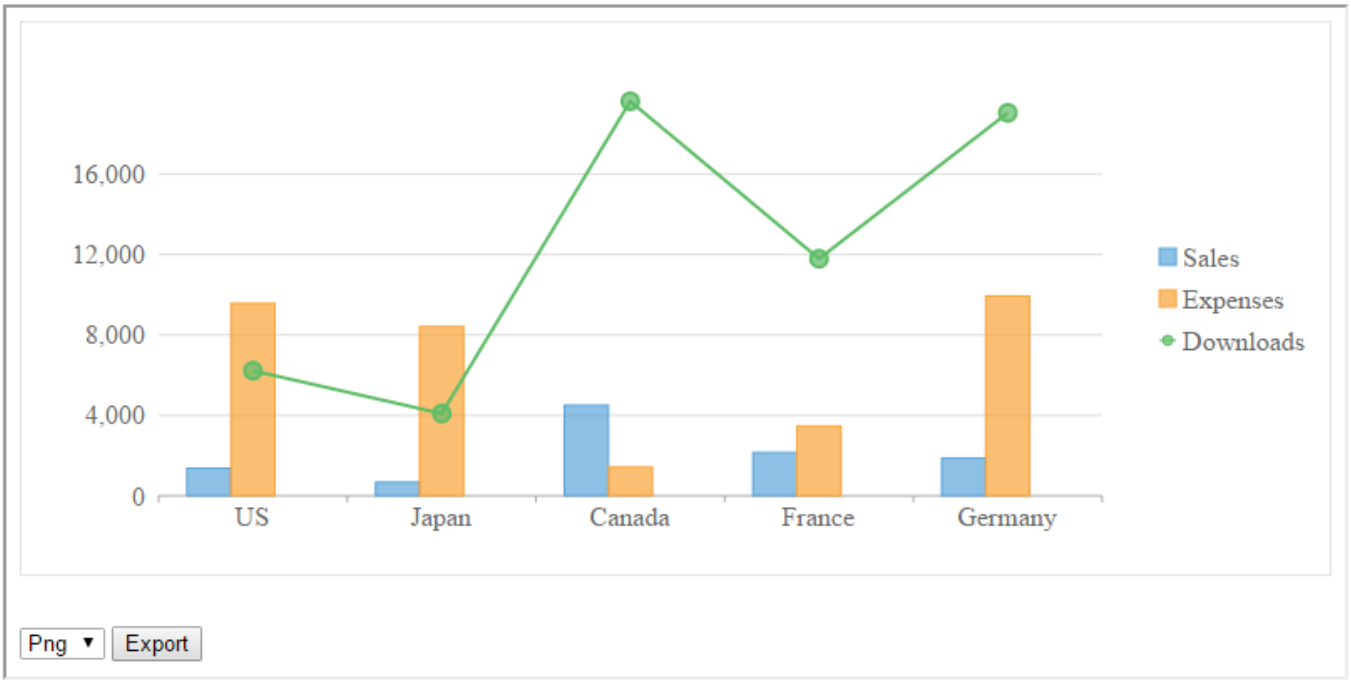
Export FlexChart to Image

This section shows how to call the Web API service project through a client application and add an export function for exporting FlexChart control as an image.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the FlexChart appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For detailed steps refer to Adding Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views|FlexChart|Index.cshtml** to export the FlexChart control to image.

Index.cshtml copyCode

```
<script>
function exportImage() {
    var exporter = new wijmo.chart.ImageExporter();
    imageType = document.getElementById("mySelect").value;
    control = wijmo.Control.getControl('#flexChart');
    exporter.requestExport(control,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/image", {
        fileName: "exportFlexChart",
        type: imageType,
    });
}
</script>
```

HTML

Add the following markup in <script> tags to export FlexChart control to image.

JavaScript

```
<script type="text/javascript">
    function exportImage() {
        var exporter = new wijmo.chart.ImageExporter();
        var control = wijmo.Control.getControl("#FlexChart");
        exporter.requestExport(control, "
http://demos.componentone.com/ASPNET/WebAPI/api/export/image", {
            fileName: "exportFlexChart",
            type: Png,
        });
    }
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.chart.min.js" type="text/javascript"></script>
<script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export and import functionalities.

Add the following code in **Views | FlexChart | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<select id="mySelect">
    <option selected>Png</option>
    <option>Jpg</option>
    <option>Gif</option>
    <option>Bmp</option>
    <option>Tiff</option>
</select>
<button onclick="exportImage()" title="Export">Export</button>
```

HTML

Add the following markup within the <body> tags to create button for export function.

```
HTML
<button onclick="exportImage()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the control as image.

> Explore detailed demo samples of image export service for MVC and Wijmo5 FlexChart control in:
>
> - MVCFlexChart Live Demo
> - Wijmo5FlexChart Live Demo

Back to Top

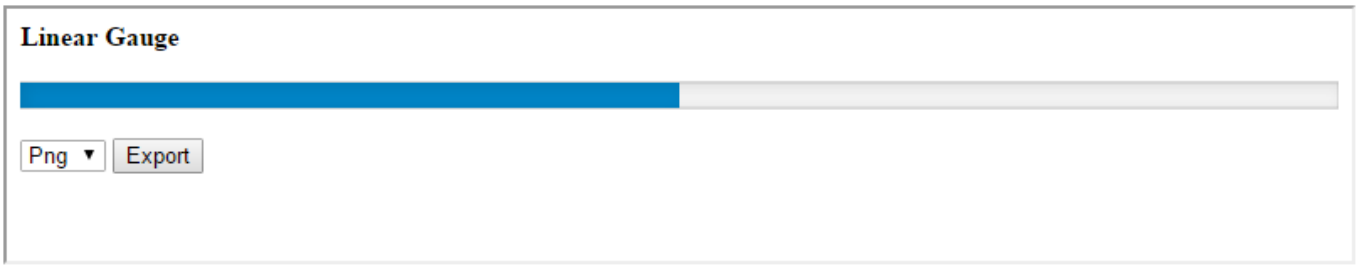
Export LinearGauge to Image

This section shows how to call the Web API service project through a client application and add an export function for exporting LinearGauge control as an image.

Step 1: Call the Service

Step 2: Run the Client Project

The following image shows how the LinearGauge appears after completing the steps above:



Step 1: Call the Service

Complete the following steps to call the Web API service.

1. Add **C1 Web API Client JavaScript** file and its reference to your MVC or HTML project (For detailed steps refer to Adding Web API Client JavaScript).
2. Create a function, using Client JavaScript Helper, to implement the export functionality.

MVC

Add the following code to **Views | LinearGauge | Index.cshtml** to export the LinearGauge control to image.

Index.cshtml copyCode

```
<script type="text/javascript">
function exportImage() {
    var exporter = new wijmo.gauge.ImageExporter();
    imageType = document.getElementById("mySelect").value;
    control = wijmo.Control.getControl('#linearGauge');
    exporter.requestExport(control,
    "http://demos.componentone.com/ASPNET/C1WebAPIService/api/export/image", {
        fileName: "exportLinearGauge",
        type: imageType,
    });
}
</script>
```

HTML

Add the following markup in <script> tags to export LinearGauge control to image.

JavaScript

```
<script type="text/javascript">
    function exportImage() {
        var exporter = new wijmo.gauge.ImageExporter();
        var gauge = wijmo.Control.getControl("#LinearGauge");
        exporter.requestExport(gauge,
"http://demos.componentone.com/ASPNET/WebAPI/api/export/image", {
            fileName: "exportLinearGauge",
            type: Png,
        });
    }
</script>
```

Note: To use Wijmo 5 controls in your HTML application, you need to include references to few Wijmo files in your HTML pages. You may either download these wijmo files and copy them to appropriate folders in your application, or reference Wijmo files hosted in cloud on our Content Delivery Network (CDN). If you download and place the Wijmo script files in a folder named "Scripts", css files in a folder named "Styles", and script files specific to Wijmo controls to "Controls" folder, then add the following references within <head> tags of your HTML pages:

HTML

```
<script src="Controls/wijmo.min.js" type="text/javascript"></script>
<link href="Styles/wijmo.min.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.10.2.min.js" type="text/javascript"></script>
<script src="Controls/wijmo.gauge.min.js" type="text/javascript"></script>
```

```
    <script src="Scripts/webapiclient.min.js" type="text/javascript"></script>
```

3. Add a button and call the export functionality on its button click for initiating export request.

MVC

- Add the following code in **Views | MVCFlexGrid | Index.cshtml** to add buttons for export functionality.

Add the following code in **Views | LinearGauge | Index.cshtml** to add buttons for export functionality.

Index.cshtml copyCode

```
<select id="mySelect">
    <option selected>Png</option>
    <option>Jpg</option>
    <option>Gif</option>
    <option>Bmp</option>
    <option>Tiff</option>
</select>
<button onclick="exportImage()" title="Export">Export</button>
```

HTML

Add the following markup within the <body> tags to create button for export function.

HTML

```
<button onclick="exportImage()" title="Export">Export</button>
```

Back to Top

Step 2: Run the Client Project

MVC Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.

HTML Application

- Save your HTML file and open it in a browser.

Export button appears along with the control, in the output. Use this button to export the control as image.

Explore detailed demo samples of image export service for MVC and Wijmo5 LinearGauge control in:

- MVCLinearGauge Live Demo
- Wijmo5LinearGauge Live Demo

Back to Top

Barcode Services

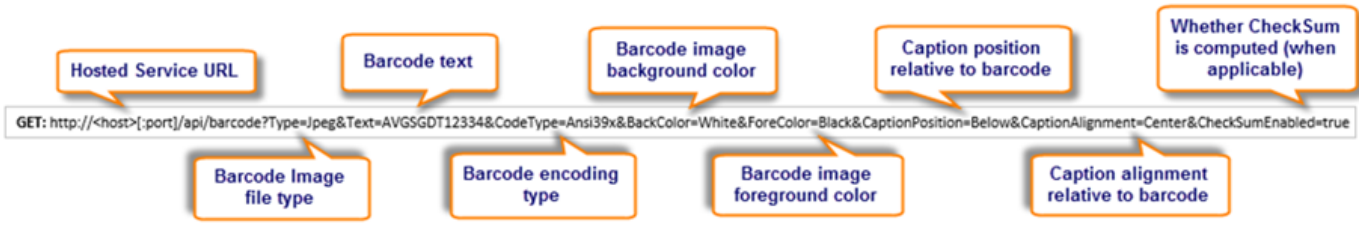
Generate scannable C1 supported barcode on the fly from the (given) text, in/using a variety of standard encoding types. The barcode so obtained can then be saved as image or stream.

Let's say you want to use barcode in PDF document or a form. You simply need to provide the desired text to be barcoded and the barcode type through your simple client application. The client application sends a GET request to the REST API service, which returns the generated barcode in response as an image stream. Barcode service lets you save the barcode image to PNG, JPEG, BMP, GIF, or TIFF image formats. Embed/ use the obtained barcode image in PDF or other documents, forms, databases, ID cards and more.

Barcode Service Request Schema

Your client application sends an HTTP request message to the service, as- **GET: http://<host>[:port]/api/barcode**.

You need to specify various barcode parameters in this service URL, to generate the desired barcode image. The following illustration depicts a request URL with parameters to generate barcode.



The following table shows barcode parameters and their respective supported values.

Barcode Parameter	Values Supported	Description
Type	PNG, JPEG, BMP, GIF, TIFF	Specifies the file type of the generated barcode image.
Text	Depends on encoding type. For details refer to Supported Barcode Symbologies.	Specifies the text string that is encoded as barcode image.
Code Type	Refer to Supported Barcode Symbologies.	Specifies all the supported encoding types to generate image from the text string.
Back Color	Transparent, White, Black, Red, Green, Blue, Yellow, Orange.	Specifies the background color in the generated barcode image.
Fore Color	Transparent, White, Black, Red, Green, Blue, Yellow, Orange.	Specifies the foreground color of the supported barcode image.
Caption Position	Above, Below, None	Specifies the position of the barcode caption on the generated barcode image.
Caption Alignment	Left, Center, Right	Specifies the alignment of the barcode caption on the generated barcode image.
CheckSum Enabled	True, False	Specifies whether a checksum will be computed for the barcode and included in the generated image.

Generate Barcode from Texts

This section demonstrates how to call the Web API service through a client application and generate barcode image from the desired text.

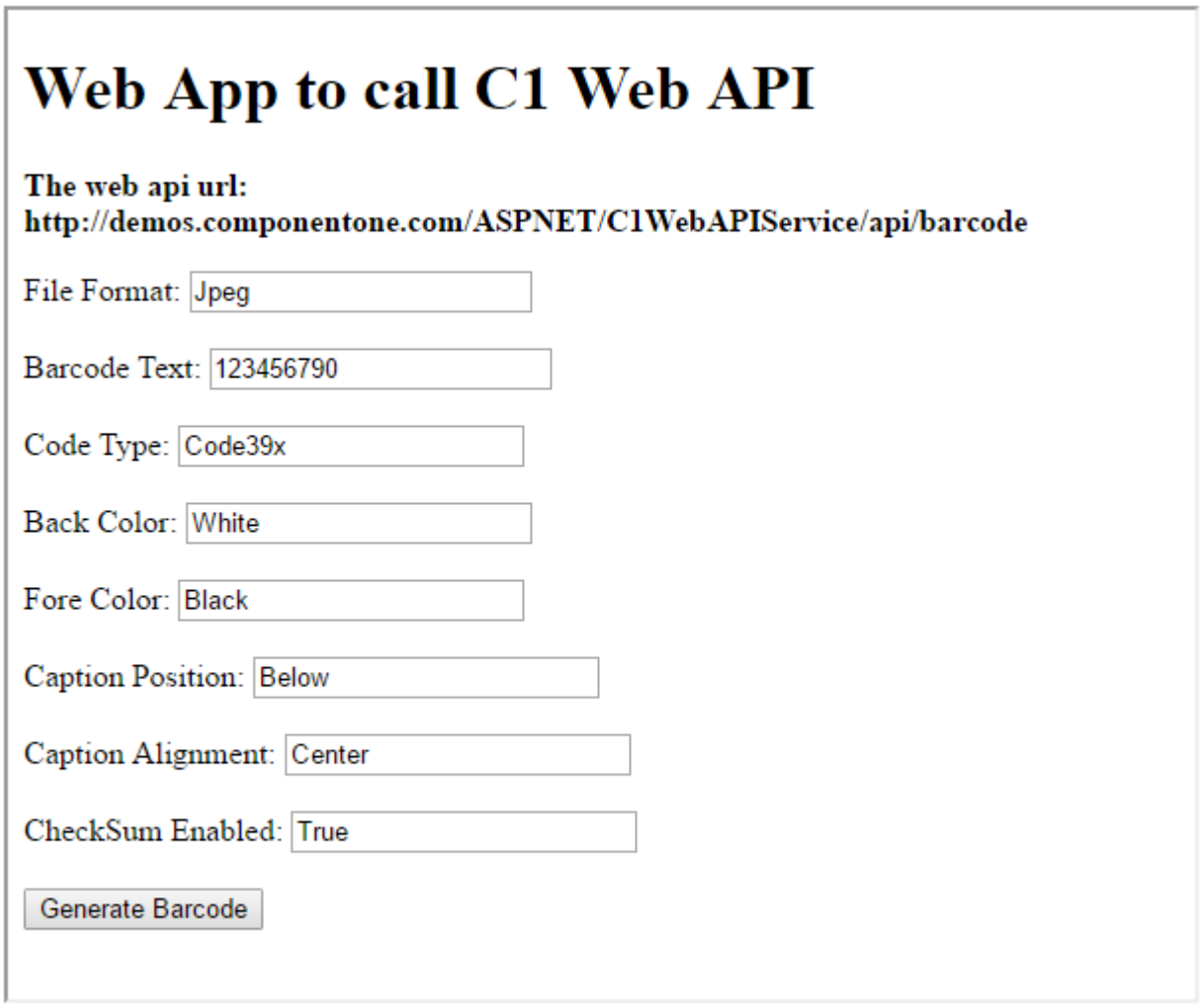
Step 1: Call the Service

Step 2: Run the Client Project

The following image shows the barcode generated after completing the steps above:

The following example makes a call to the Web API service through HTML as well as WinForms client applications. These clients send a GET request to the service, which returns a barcode stream in response. This response stream can then be saved as image, as the barcode image above.

In the following example, the service url takes 1234567890 in **Text** parameter and **encoding type** parameter as Code39x, to generate the above image.

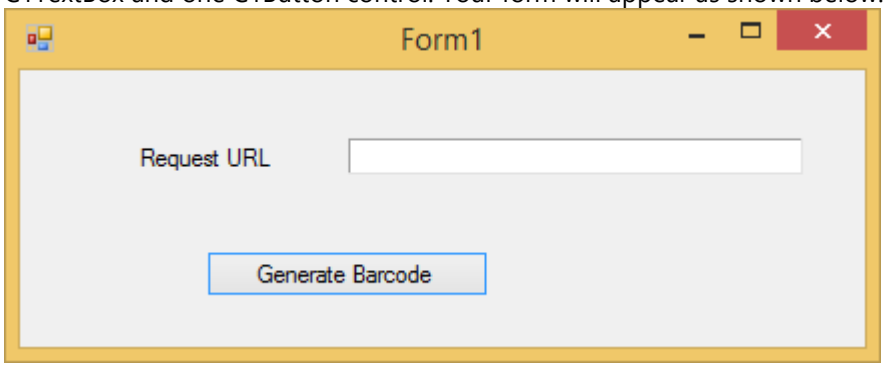


Step 1: Call the Service

Complete the following steps to call the Web API service.

C#

1. Create a WinForms application, as discussed in Configure Client for REST API service. Add one C1Label, C1TextBox and one C1Button control. Your form will appear as shown below.



2. Define a method (for example: GetBarcode**()**) in form class of your WinForms application, to call the service application, as shown below.

C#

```
public void GetBarcode() {
  var apiURL = string.IsNullOrEmpty(c1TextBox1.Text) ?
  "http://demos.componentone.com/ASPNET/WebAPI/api/barcode?
  Type=Png&Text=1234567890&CodeType=Ansi39" : c1TextBox1.Text;
  WebRequest request = WebRequest.Create(apiURL);
  WebResponse response = request.GetResponse();
  var fileStream = File.Create("D:\\BarcodeImg.Png");
        //The file format specified here should be same as that
specified in the
  request url
  response.GetResponseStream().CopyTo(fileStream);
}
```

3. Call the GetBarcode() method on button-click event of **Generate Barcode** button.

HTML

1. Create an HTML application, as discussed in Configure Client for REST API service.
2. Add the following markups in the <form> tags, within <body> tags, of your HTML page.

HTML

```
<form action="http://demos.componentone.com/ASPNET/WebAPI/api/barcode"
method="GET">
  <label for="fileFormat">File Format:</label>
  <input type="text" id="fileFormat" name="type" value="Jpeg" />
  <br />
  <br />
  <label for="text"> Barcode Text:</label>
  <input type="text" id="text" name="text" value="123456790" />
  <br />
  <br />
  <label for="codeType">Code Type:</label>
  <input type="codeType" id="codeType" name="codeType" value="Code39x" />
  <br />
```

```
  <br />
  <label for="backColor">Back Color:</label>
  <input type="backColor" id="backColor" name="backColor" value="White" />
  <br />
  <br />
  <label for="foreColor">Fore Color:</label>
  <input type="foreColor" id="foreColor" name="foreColor" value="Black" />
  <br />
  <br />
  <label for="captionPosition">Caption Position:</label>
  <input type="captionPosition" id="captionPosition" name="captionPosition"
value="Below" />
  <br />
  <br />
  <label for="captionAlignment">Caption Alignment:</label>
  <input type="captionAlignment" id="captionAlignment" name="captionAlignment"
value="Center" />
  <br />
  <br />
  <label for="CheckSumEnabled">CheckSum Enabled:</label>
  <input type="CheckSumEnabled" id="CheckSumEnabled" name="CheckSumEnabled"
value="True" />
  <br />
  <br />
  <input type="submit" value="Generate Barcode" />
</form>
```

Note that, for GET request we set **method** attribute of <form> tag to GET, and set its **action** attribute to service request URL. Also, we create input controls on the HTML page, which take various barcode parameters to generate the barcode image, from the specified text, to the desired image format.

Back to Top

Step 2: Run the Client Project

WinForms Application

- Click **Build | Build Solution** to build the project.
- Press **F5** to run the project.
- Provide the service URL, along with the query string containing appropriate barcode parameters, in the textbox corresponding to Request URL field.
- Click the **Generate Barcode** button. The generated barcode image will get downloaded at the location specified within the GetBarcode() method.

HTML Application

- Save your HTML file and open it in a browser.
- Set the appropriate barcode parameters for the desired barcode image, and click **Generate Barcode** button.

> Explore detailed demo samples of REST API service to generate barcode at:
>
> - Generate Barcode Live Demo

Back to Top

Supported Barcode Symbologies

The various barcode encoding types (or barcode symbology) specify the scheme used to convert characters and digits of a text/message into a pattern of wide and narrow bars, spaces, and quiet zone in a barcode.

The following table illustrates the encoding types supported in Barcode Service.

Code Type	Example	Description
Ansi39		ANSI 3 of 9 (Code 39) uses upper case, numbers, - , * $ / + %. This is the default barcode style.
Ansi39x		ANSI Extended 3 of 9 (Extended Code 39) uses the complete ASCII character set.
Codabar		Codabar uses A B C D + - : . / $ and numbers.
Code_128_A		Code 128 A uses control characters, numbers, punctuation, and upper case.
Code_128_B		Code 128 B uses punctuation, numbers, upper case and lower case.
Code_128_C		Code 128 C uses only numbers.
Code_128auto		Code 128 Auto uses the complete ASCII character set. Automatically selects between Code 128 A, B, and C to give the smallest barcode.
Code_2_of_5		Code 2 of 5 uses only numbers.
Code93		Code 93 uses uppercase, % $ * / , + -, and numbers.
Code25intlv		Interleaved 2 of 5 uses only numbers.

Code39		Code 39 uses numbers, % * $ /. , - +, and upper case.
Code39x		Extended Code 39 uses the complete ASCII character set.
Code49		Code 49 is a 2D high-density stacked barcode containing two to eight rows of eight characters each. Each row has a start code and a stop code. Encodes the complete ASCII character set.
Code93x		Extended Code 93 uses the complete ASCII character set.
DataMatrix		Data Matrix is a high density, two-dimensional barcode with square modules arranged in a square or rectangular matrix pattern.
EAN_13		EAN-13 uses only numbers (12 numbers and a check digit). It takes only 12 numbers as a string to calculate a check digit (CheckSum) and add it to the thirteenth position. The check digit is an additional digit used to verify that a bar code has been scanned correctly. The check digit is added automatically when the CheckSum property is set to True.
EAN_8		EAN-8 uses only numbers (7 numbers and a check digit).
EAN128FNC1		EAN-128 is an alphanumeric one-dimensional representation of Application Identifier (AI) data for marking containers in the shipping industry. This type of bar code contains the following sections: • Leading quiet zone (blank area) • Code 128 start character • FNC (function) 1 character which allows scanners to identify this

		as an EAN-128 barcode • Data (AI plus data field) • Symbol check character (Start code value plus product of each character position plus value of each character divided by 103. The checksum is the remainder value.) • Stop character • Trailing quiet zone (blank area) The AI in the Data section sets the type of the data to follow (i.e. ID, dates, quantity, measurements, etc.). There is a specific data structure for each type of data. This AI is what distinguishes the EAN-128 code from Code 128. Multiple AIs (along with their data) can be combined into a single bar code. EAN128FNC1 is a UCC/EAN-128 (EAN128) type barcode that allows you to insert FNC1 character at any place and adjust the bar size, etc., which is not available in UCC/EAN-128. To insert FNC1 character, set "\n" for C#, or "vbLf" for VB to Text property at runtime.
IntelligentMail		Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.
JapanesePostal		This is the barcode used by the Japanese Postal system. Encodes alpha and numeric characters consisting of 20 digits including a 7-digit postal code number, optionally followed by block and house number information. The data to be encoded can include hyphens.
Matrix_2_of_5		Matrix 2 of 5 is a higher density barcode consisting of 3 black bars and 2 white bars.
MicroPDF417		MicroPDF417 is two-dimensional (2D), multi-row symbology, derived from PDF417. Micro-PDF417 is designed for applications that need to encode data in a two-dimensional (2D) symbol (up to 150 bytes, 250 alphanumeric characters, or 366 numeric digits) with

		the minimal symbol size. MicroPDF417 allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set "\n" for C#, or "vbLf" for VB to Text property at runtime.
MSI		MSI Code uses only numbers.
Pdf417		Pdf417 is a popular high-density 2-dimensional symbology that encodes up to 1108 bytes of information. This barcode consists of a stacked set of smaller barcodes. Encodes the full ASCII character set. It has ten error correction levels and three data compaction modes: Text, Byte, and Numeric. This symbology can encode up to 1,850 alphanumeric characters or 2,710 numeric characters.
PostNet		PostNet uses only numbers with a check digit.
QRCode		QRCode is a 2D symbology that is capable of handling numeric, alphanumeric and byte data as well as Japanese kanji and kana characters. This symbology can encode up to 7,366 characters.
RM4SCC		Royal Mail RM4SCC uses only letters and numbers (with a check digit). This is the barcode used by the Royal Mail in the United Kingdom.
RSS14		RSS14 is a 14-digit Reduced Space Symbology that uses EAN.UCC item identification for point-of-sale omnidirectional scanning.
RSS14Stacked		RSS14Stacked uses the EAN.UCC information with Indicator digits as in the RSS14Truncated, but stacked in two rows for a smaller width. RSS14Stacked allows you to set Composite Options, where you can select the type of the barcode in the

		Type drop-down list and the value of the composite barcode in the **Value** field.
RSS14StackedOmnidirectional		RSS14StackedOmnidirectional uses the EAN.UCC information with omnidirectional scanning as in the RSS14, but stacked in two rows for a smaller width.
RSS14Truncated		RSS14Truncated uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale.
RSSExpanded		RSSExpanded uses the EAN.UCC information as in the RSS14, but also adds AI elements such as weight and best-before dates. RSSExpanded allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at runtime.
RSSExpandedStacked		RSSExpandedStacked uses the EAN.UCC information with AI elements as in the RSSExpanded, but stacked in two rows for a smaller width. RSSExpandedStacked allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at runtime.
RSSLimited		RSS Limited uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale. RSSLimited allows you to set Composite Options, where you can

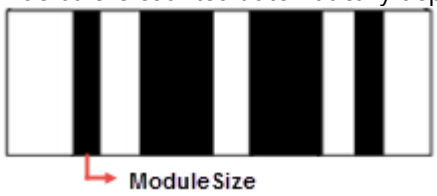
		select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
UCCEAN128		UCC/EAN –128 uses the complete ASCII character Set. This is a special version of Code 128 used in HIBC applications.
UPC_A		UPC-A uses only numbers (11 numbers and a check digit).
UPC_E0		UPC-E0 uses only numbers. Used for zero-compression UPC symbols. For the Caption property, you may enter either a six-digit UPC-E code or a complete 11-digit (includes code type, which must bezero) UPC-A code. If an 11-digit code is entered, the Barcode control will convert it to a six-digit UPC-E code, if possible. If it is not possible to convert from the 11-digit code to the six-digit code, nothing is displayed.
UPC_E1		UPC-E1 uses only numbers. Used typically for shelf labeling in the retail environment. The length of the input string for U.P.C. E1 is six numeric characters.

Barcode Features

Following are some features of C1 supported barcode, that can be generated using Web API service:

- **BarCodeDirection:** Lets you select the barcode's direction. The available options are:
  - LeftToRight: The barcode symbol is printed left to right (default).
  - RightToLeft: The barcode symbol is printed right to left.
  - TopToBottom: The barcode symbol is printed top to bottom.
  - BottomToTop: The barcode symbol is printed bottom to top.
- **BarHeight:** Lets you specify the height of a barcode in screen pixels. If the bar height exceeds the height of the control, this property is ignored.
- **CodeType:** Lets you select encoding that should be applied to the value stored in the Text property to generate the barcode image.
- **CaptionAlignment:** Lets you select the display position of the value of barcode. The available options are Left, Right, and Center.
- **CaptionGrouping:** Lets you specify a value indicating whether to add spaces between groups of characters in the caption to make long numbers easier to read.
- **CaptionPosition**: Lets you select the caption's vertical position relative to the barcode symbol. The available options are None, Above, and Below.
- **Image:** Gets an image of the barcode that represents the value in the Text property, obtained using the encoding specified by the CodeType.
- **ModuleSize:** Lets you specify the module (narrowest bar width) of a barcode in screen pixels. The width of

wide bars is counted automatically depending on the barcode type.



- **QuietZone**: Lets you specify the quiet zone(s) in a barcode. A quiet zone is an area of blank space on either side of a barcode that tells the scanner where the symbology starts and stops. The options available are as follows:
  - Left: Enter the size of blank space to leave to the left of the barcode.
  - Right: Enter the size of blank space to leave to the right of the barcode.
  - Top: Enter the size of blank space to leave at the top of the barcode.
  - Bottom: Enter the size of blank space to leave at the bottom of the barcode.

  The following image shows Left and Right quiet zones:

- **Text**: Lets you specify the value that is encoded as a barcode image.
- **WholeSize:** Lets you specify the size of the overall barcode. WholeWidth represents the width and WholeHeight represents the height of the overall barcode.

- **FixLength**: Lets you specify the fixed number of digits of values of the barcode. It takes the integer value.
- **AutoSize:** Lets you specify whether the barcode should stretch to fit the control. It takes the value True or False.

  When AutoSize is set to True,
  - the barcode automatically stretches to fit the control.
  - the readable size is calculated by the barcode itself.
  - the size of Matrix barcodes is calculated by **OnCalculateSize** method.
  - the size of the non-matrix barcodes, it is calculated by **BarHeight** and **ModuleSize**.

  When AutoSize is set to False,

- the size size of the barcode is determined by Width or Height properties.
- the control gets clipped if the BarHeight is larger than control's height
- some empty space between the barcode and the control is left if the BarHeight is smaller than height,

The options that are specific to the type of barcodes are as follows:

CheckSumEnabled: Lets you specify whether the check digits are automatically added or not. When data to be bound already includes check digits, programmers sometimes want to prevent controls from automatically including them. This property is supported for Code49, Code128, PostNet5/9/11, and JapanesePostal barcodes.

Ean128Fnc1Options:

- **Dpi:** Lets you specify the resolution of the printer. It takes the integer value.
- **BarAdjust:** Lets you specify the adjustment size by dot.
- **ModuleSize:** Lets you specify the horizontal size of the barcode module. It takes the integer value.

Code25intlvOptions:

- **BearBar:** Lets you select whether or not to display bearer bar to ITF (Interleaved Two of Five) barcode. It takes the value True or False.
- **LineStroke:** Lets you select the color of the bearer bar.
- **LineStrokeThickness:** Lets you select the line width of the bearer bar. It takes the integer values.

Code49Options:

- **Grouping:** Lets you use grouping in the barcode. Its value is either True or False.
- **Group:** Obtains or sets group numbers for barcode grouping. Its value is between 0 and 8. If the value of **Grouping** is True, the range of value of Group is from 0 to 8. If the value of Grouping is False, value of Group is 0. If the value of **Grouping** is True, and the **Group** value is smaller than 0 or larger than 8, the BarCodeException.EnumErrorCode.Code49GroupNo will be thrown.

DataMatrixOptions:

- **EccMode:** Lets you select the ECC mode. The possible values are ECC000, ECC050, ECC080, ECC100, ECC140, or ECC200.
- **Ecc200SymbolSize:** Lets you select the size of ECC200 symbol. The default value is SquareAuto.
- **Ecc200EncodingMode:** Lets you select the ECC200 encoding mode. The possible values are Auto, ASCII, C40, Text, X12, EDIFACT, or Base256.
- **Ecc000_140SymbolSize:** Lets you select the size of the ECC000_140 symbol.
- **StructuredAppend:** Lets you select whether the current barcode symbol is part of structured append symbols.
- **StructureNumber:** Lets you specify the structure number of current symbol within the structured append symbols. The range of this value is from 0 to 15.
- **FileIdentifier:** Lets you specify the file identifier of a related group of structured append symbols. The valid file indentifier value should be within [1,254]. Setting file identifier to 0 lets the file identifier to be calculated automatically.

GS1CompositeOptions:

- **Type:** Lets you select the composite symbol type. Its value can be None or CCA. CCA (Composite Component - Version A) is the smallest variant of the 2-dimensional composite component.
- **Value:** Lets you specify the CCA character data.

MicroPDF417Options:

- **CompactionMode:** Lets you select the type of CompactionMode. The possible values are Auto, TextCompactionMode, NumericCompactionMode, and ByteCompactionMode.
- **FileID:** Lets you specify the file id of structured append symbol. It takes the value from 0 to 899. If this value is smaller than 0 or larger than 899, the BarCodeException.EnumErrorCode.MicroPDF417FileID is thrown.

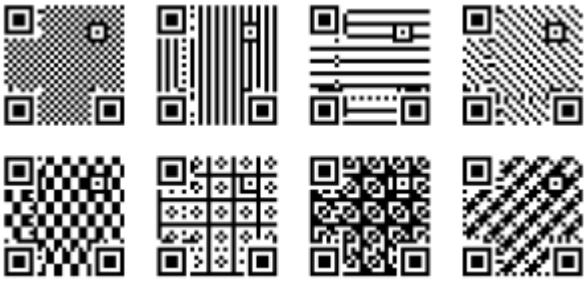
- **SegmentCount:** Lets you specify the segment count of structured append symbol. It takes the value from 0 to 99999. If this value is smaller than 0 or larger than 99999, the BarCodeException.EnumErrorCode.MicroPDF417SegmentCount is thrown.
- **SegmentIndex:** Lets you specify the segment index of structured append symbol. It takes the value from 0 to 99998 and less than the value of segment count. If this value is smaller than 0 or larger than 99998, the BarCodeException.EnumErrorCode.MicroPDF417SegmentIndex is thrown.
- **Version:** Lets you select the symbol size. The default value is ColumnPriorAuto.

PDF417Options:

- **Column:** Lets you specify the column numbers for the barcode. It takes the integer value; the default value is -1 and the range of this value is 1 to 30. If this value is not equal to -1 or smaller than 1 or larger than 30, the BarCodeException.EnumErrorCode.PDF417Col is thrown.
- **Row:** Lets you specify the row numbers for the barcode. It takes the integer value; the default value is -1 and the range of this value is from 3 to 90. If this value is not equal to -1 or smaller than 3 or larger than 90, the BarCodeException.EnumErrorCode.PDF417Row is thrown.
- **ErrorLevel:** Lets you specify the error correction level for the barcode. It takes the integer value; the value is -1 or the range of this value is from 0 to 8. If this value is not equal to -1 or is smaller than 0 or larger than 8, the BarCodeException.EnumErrorCode.PDF417ErrorLevel is thrown.
- And Level 0 is low strength and the default value is -1.
- **Type:** Lets you select the type of PDF417 barcode. The available types are Normal and Simple.

QRCodeOptions:

- **Model:** Lets you select the model of QRCode. The available models are Model1 and Model2.
- **ErrorLevel:** Lets you select the error correction level for the barcode. The available options are Low, Medium, Quality, and High.
- **Version:** Lets you specify the version of the barcode.
- **Mask:** Lets you select the pattern used for masking barcode. In order to make sure QRCode being successfully read, mask process is required to balance brightness. The options available are Auto, Mask000, Mask001, Mask010, Mask011, Mask100, Mask101, Mask110, and Mask111. The following image shows masking in QRCode:



- **Connection:** Lets you select whether connection is used for the barcode. It takes the value True of False.
- **ConnectionNumber:** Lets you specify the connection number for the barcode. It takes the integer value ranging from 0 to 15. If this value is smaller than 0 or larger than 15, EnumErrorCode.QRCodeConnectionNo exception will be thrown.
- **Encoding:** Lets you select the encoding for the barcode. It takes the integer value. The value is -1 or the range is from 1 to 14 when the Model property is set to Model1. The value is -1 or the range is from 1 to 40 when the Model property is set to Model2.

RssExpandedStackedOptions:

- **RowCount:** Lets you specify the number of stacked rows. It takes the integer value; the range is from 1 to 11. If this value is smaller than 1 or larger than 11, the BarCodeException.EnumErrorCode.RSSExpandedStackedCount is thrown.

Release History

The release history contains information on all the new controls, breaking changes, improvements, new features and bug fixes for Web API Edition since the previous release. Choose a release version to learn more:

2017 v3

Web API

Enhancements

- Added support for ASP.NET Core 2.0 Framework.

Excel Service

Bugs

- Fixed an issue where error occurs after post the file using "WebApi.Core20.2017.sln" as a server link.

Report Service

Bug Fixes

- Fixed an issue where user could not export in Excel format(xls, xlsx) when file size is over 75kb.

2016 v2

Core

Bugs

- Fixed an error where selecting one parameter does not check another parameter option in ASP.NET Core.

Excel Service

Enhancements

- Added support for Generate Excel From Given Template And Data.
- Added support for split operation.
- Added support for add/delete row/column operation in Excel.
- Added support for find/replace operation in Excel.
- Added support for Group/Ungroup column/row of excel file in storage.
- Added support for Hide/Unhide column/row of excel file in storage.
- Added support for Add posted Excel file to storage.

Report Service

Bug Fixes

- Optimize parameters and clear action for report cache controller.
- Optimize search and bookmark action of report cache controller.
- Fixed an issue where the user could not search a text that included symbols.
- Fixed an issue where changing from portrait to landscape mode does not display correctly in some reports.

ASP.NET Core Web API

Bug Fixes

- Fixed an issue where an error is displayed after importing excel to FlexGrid using ASP.NET Core Web API

Web API Explorer

Bug Fixes

- Suggest to fix remain the page of storage after uploading file successfully.
- Fixed an issue where move down takes no effect and need to confirm the "Move do Beginning".
- Fixed data match the template for the sample.

2017 v1

Core

Enhancements

- Updated WebApi Visual Studio 2017 ASP.NET Core project templates.
- Document - Added 'Group' property for supported format response data ExportOptionDescription. Type change of ExportOptionDescription.DefaultValue from string to object.
- BarCode, Excel and Image services are made virtual that will allow the user to override.

Bug Fixes

- Fixed an issue where some inconsistent behavior was observed after exporting the data in some page.
- Fixed some inconsistent behavior found in PDF sample.
- Fixed an issue where a user try to export a report for a particular format, it gets exported in HTML file format.
- Fixed an issue where server side cannot respond "Find" request of "Excel Service" in WebApiExplorer.

Report Service

Bug Fixes

- Fixed an issue where JavaScript error was observed when user enter a blank space in search box.
- Fixed an issue where Report does not load and error is observed when the user click the print layout button of

SSRS report.
- Fixed an issue where error was displayed after exporting Report/PDF by using post method in WebApi2 application.

DataEngine Service

Bug Fixes

- Fixed an issue where the detail row show no date if the fields contains "Date" in the OlapExplorer sample.

2016 v3.5

Breaking Change

- Report, PDF, and DataEngine services are completely re-designed for better performance and improved standards. For more information, see ComponentOne Studio Web API documentation.
- In future, any FlexViewer build released on or after 2016v3.5 (4.0.20163.101), should use Web Api build (4.0.20163.79) and later.

Core

Enhancements

- Updated WebAPI sample to support Visual Studio 2017.

Report Service

Enhancements

- Added SSRS support in Report WebAPI.
- Added pageSettings in the parameters of export action of report controller.

Bug Fixes

- Fixed an issue where an internal error displays after the user set the wrong path of SSRS to the service of WebApi.
- Fixed an issue where Report does not load and error is observed when the user click the print layout button.

Data Engine

Enhancements

- Allow the user to set the customized max count for the aggregated data. For information about max count, see Data Source Manager topic.
- Added total items count for getting detail cell data and raw data.
- Supports progress when getting the aggregated data.
- Reafactoring in DataEngine Web API URL.

Bug Fixes

- Fixed an issue where the return "status" value of DataEngine URL is always "Executing" after setting the wrong "ViewDefinition".

2016 v3

Core

Enhancements

- Implemented 2016v3 license from PDF, Report and Data Engine services.
- Added PDF Web API Service.
- Added Data Engine Service.

Excel Service

Bug Fixes

- Merge cell can be exported correctly to any desirable format.

Report Service

Enhancements

- Added SSRS support in Report WebAPI.
- Added pageSettings in the parameters of export action of report controller.

Bug Fixes

- Fixed an issue where, the request of "OutLines" returns "false" while the report contains outline.
- Fixed. Clicking Hyperlink navigate correctly to "Sales_by_Region" report.
- Optimize search and bookmark action of report cache controller.

Data Engine

Bug Fixes

- Fixed an issue where some parameters in "analysis" request invite underlying error.
- Fixed an issue where internet error "A task may only be disposed if it is in a completion state" in the browser if move the fields to frequently.
- Fixed an issue where Dataengine token's status is always "Executing" when you send "analysis" request without any data.

Web API Explorer

Bug Fixes

- Fixed an issue where, PivotPanel and PivotGrid are overlapped in IE 11.
- Fixed Sample project bug, where user fails to send "detail" request.
- Fixed Sample project bug, where user fails to send "uniquevalues" request.
- Fixed data match the template for the sample.

2016 v2.5

Core

Enhancements

- C1 WebApi refactotring. For more information, see Installation topic.

Excel Service

Bugs

- Fixed an issue where any user cannot generate data source to JSON format.
- Fixed an issue wehere user cannot export Excel in virtual scrolling page.

Report Service

Bug Fixes

- Fixed an issue where error was displayed when setting PageSettings without parameters.
- Fixed an issue where setting a parameter creates problem in Cascading Parameters report.

2016 v2

Core

Bugs

- Fixed an error where selecting one parameter does not check another parameter option in ASP.NET Core.

Excel Service

Enhancements

- Added support for Generate Excel From Given Template And Data.
- Added support for split operation.
- Added support for add/delete row/column operation in Excel.
- Added support for find/replace operation in Excel.
- Added support for Group/Ungroup column/row of excel file in storage.
- Added support for Hide/Unhide column/row of excel file in storage.
- Added support for Add posted Excel file to storage.

Report Service

Bug Fixes

- Optimize parameters and clear action for report cache controller.
- Optimize search and bookmark action of report cache controller.
- Fixed an issue where the user could not search a text that included symbols.
- Fixed an issue where changing from portrait to landscape mode does not display correctly in some reports.

ASP.NET Core Web API

Bug Fixes

- Fixed an issue where an error is displayed after importing excel to FlexGrid using ASP.NET Core Web API

Web API Explorer

Bug Fixes

- Suggest to fix remain the page of storage after uploading file successfully.
- Fixed an issue where move down takes no effect and need to confirm the "Move do Beginning".
- Fixed data match the template for the sample.

ComponentOne Studio Web API Edition 169

2016 v1.5

Image Service

Other Notes

- While working on image services, user needs to add **phantomjs.exe** file to the service application. Refer to How to add phantomjs file to service for more information.

ASP.NET Core Web API Sample

Bug Fix

- Fixed the issue where "C1.Web.Api" reference is missing in project.json file, due to which license generation failed after delete key information in startup.cs.

2016 v1

Core

Enhancements

- Added support for Report service.

Excel Service

Bug Fixes

- Fixed the issue where excel file was not getting exported and error occurred after generating excel file from a XML file available in storage, which was posted from client.
- Fixed the issue where user observed a javascript error after importing an excel file in Excel Service sample page.
- Fixed the issue where File Name was not getting displayed correctly in the generated excel file.

Enhancements

- Column header is exported to excel file after exporting FlexGrid to excel format by setting IncludeColumnHeader to false.

Image Service

No change

Web API Explorer

Bug Fixes

- Fixed issue related to IIS deployment.
- Fixed error [The "FlexGridXlsxConverter.fromWorkbookOM" method is deprecated xxx] that occurs while generating excel file which is set "Type" as "JSON" configured on server and posted from client.
- Fixed issue where user could not export flexgrid to excel by using "WebApi3" as service after updating the reference to Japanese.
- Resolved request to change text "Select a report file" as bookmark.

2015 v3.5

ComponentOne Studio Web API Edition 170

Core

Enhancements

- Provided support for ASP.NET 5 Release Candidate.

Excel Service

Bug Fixes

- Fixed the issue where HTTP access error occurs on generating, converting, and merging excel file.

Image Service

No change

Web API Explorer

No change

2015 v3

Core

Enhancements

- Provided support for ASP.NET 5 Beta 7.
- Added support for barcode service.

Excel Service

Enhancements

- Added support for excel merging service.
- Added support for excel generating service.

Bug Fixes

- Fixed the issue where date value could not be imported to FlexGrid, in IE11 and FireFox.
- Fixed the multiple issues that are observed when excel containing DateTime format column is imported.

Image Service

Bug Fixes

- Fixed the issue where one series is lost in exported image when exporting FlexChart having multiple axes and set to 'Stacked100%'.
- Fixed the issue where exported image for FlexChart is displayed as blank on binding FlexChart with database.

Web API Explorer

Bug Fixes

- Fixed the issue where JavaScript error occurs when FlexGrid's grouping data is exported.

ComponentOne Studio Web API Edition 171

2015 v2.5

Core

Updates

- Package dependency of PhantomJS has been removed in ASP.NET 5.

Bug Fixes

- Fixed the issue where "Invalid character" JavaScript error occurs when importing a file to FlexGrid by Web API service with expired/invalid license.

Excel Service

Bug Fixes

- Fixed the issue where group's style settings could not be exported to excel file.
- Fixed the issue where the exported font size is not correct when FlexGrid's content is exported to excel file.
- Fixed the issue where format with precision settings could not be exported to excel file.
- Fixed the issue where JS error occurs on exporting footer grid.
- Fixed the issue where in the exported help file cells with number type's value have green color at the top-left corner.
- Fixed the issue where "OnlyCurrentPage" option does not work when "DisableServerRead" is set to true in FlexGrid.
- Fixed the issue where C1Excel license dialog shows while exporting excel file by VS2015 WebAPI service which is run in VS2015.
- Fixed the issue where redundant row with column names is exported when a FlexGrid with new row is exported to excel file.
- Fixed the issue where FlexGrid cannot be exported to excel file correctly after exporting the imported excel file.
- Fixed the issue where Imported/Exported data type is lost in FlexGrid.
- Fixed the issue where column width was not exported appropriately.
- Fixed the issue where the excel file having JP/CN character in name could not be imported.
- Fixed the issue where hidden row is also imported when excel file with filter settings is imported to FlexGrid.
- Fixed the issue where empty rows at the end of the excel sheet lead to wrong format parsing while importing.
- Fixed the issue where data disappears after performing scrolling operation on FlexGrid with imported data.
- Fixed the issue where height of rows is not correct after importing excel data to FlexGrid.

Image Service

Bug Fixes

- Fixed the issue where chart could not be exported to image file on binding new fields to axisX by using Bind(string bindingFields, string readActionUrl).

Web API Explorer

Bug Fixes

- Exporter class of MVC FlexGrid has been changed from "c1.mvc.ExcelExporter" to "c1.mvc.grid.ExcelExporter".
- Fixed the issue where the values in filter dialog and column are inconsistent after “Export” button is clicked in “Remote DataBind”.

VS Template

Bug Fixes

- Fixed the issue where the web port is same for multiple projects created by the same Web Api VS template.