ComponentOne

ASP.NET MVC

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

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 $25 (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.

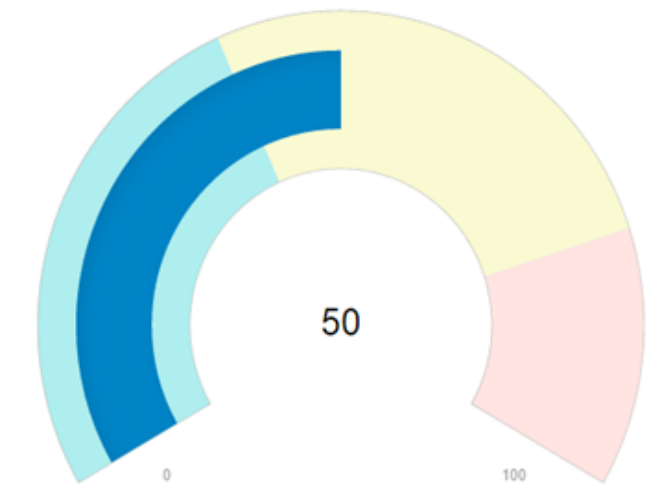
Copying and Distribution

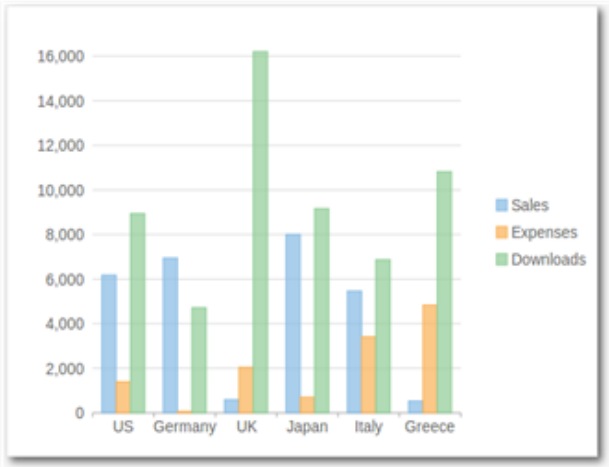
Table of Contents

ASP.NET MVC Controls

ComponentOne Studio ASP.NET MVC Edition is a collection of modern UI controls built upon latest cutting edge technologies like HTML5, CSS, ECMA5 without making compromises to support legacy browsers. Inside ASP.NET MVC Edition, you will find fast and lightweight controls ranging from data management to data visualization, project templates and professionally designed themes.

ID	Start	Product	Amount	Discount	Active
1	1/25/2015	Gadget	($4,257.83)	23 %	☐
2	2/25/2015	Gadget	$1,095.08	20 %	☐
3	3/25/2015	Gadget	$1,853.66	18 %	☐
4	4/25/2015	Widget	$3,708.86	3 %	☐
5	5/25/2015	Gadget	($3,447.73)	10 %	☐
6	6/25/2015	Gadget	($165.64)	4 %	☐
7	7/25/2015	Widget	$160.62	8 %	☐
8	8/25/2015	Widget	($2,575.30)	17 %	☐



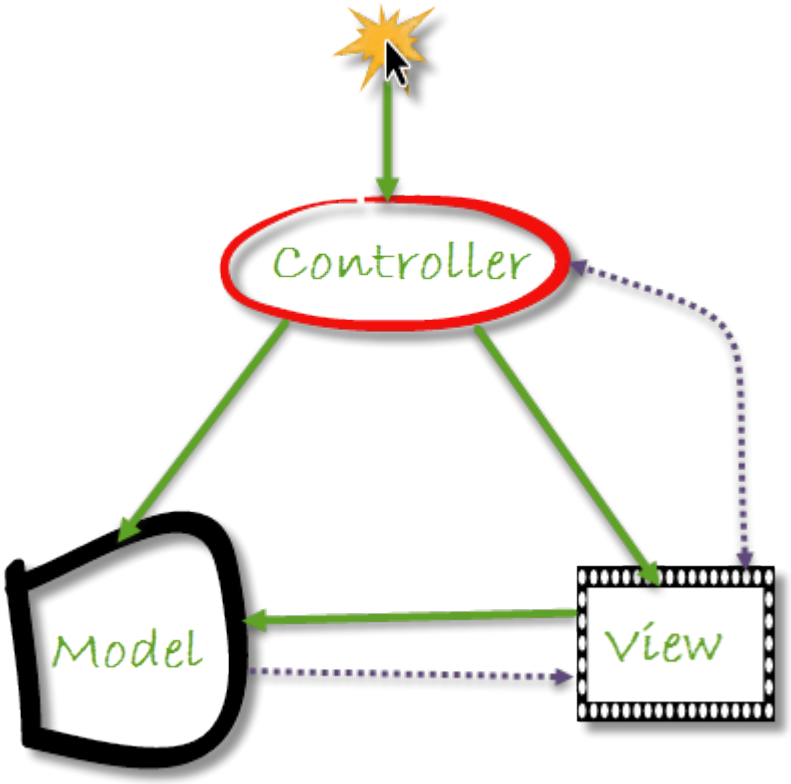


Getting Started

The following topics include information on installing ASP.NET MVC Edition, licensing, technical support, namespaces and more. This section is aimed at helping you get started with ASP.NET MVC Edition controls. For specific information on using controls, see the **Controls** section.

MVC Basics

Model-View-Controller (MVC) is a design pattern used by applications that require multiple views of the same data. The MVC pattern requires the separation of individual objects into three categories depicted in the following diagram:



Controllers: Classes that handle incoming requests to the application, retrieve model data, and then specify view templates that return a response to the client.

Models: Classes that represent the data of the application and that use validation logic to enforce business rules for that data.

Views: Template files that your application uses to dynamically generate HTML responses. Helper methods, such as HTML Helpers and Tag Helpers, simplify the generation of these HTML responses.

The control flow in an MVC application would be as follows:

1. Someone interacts with the UI in a way that triggers an event.
2. The controller notifies the model of the user's interaction and requests an action.
3. Model performs the requested action.
4. The controller requests for the view to display the result of the action.
5. The view (or views) query the model to generate the new view and grabs the data from the model.
6. The view displays the results.

Installation

Download the installer, **C1StudioInstaller.exe** from https://www.componentone.com/Downloads/. Follow the steps through the installation wizard to install the .NET DLLs and packages for ASP.NET MVC Edition. The installer also installs C1 ASP.NET MVC Web Application templates which can be used in place of the MVC templates provided by

Visual Studio, to make working with ComponentOne controls easier.

> All dependency references, style sheets and scripts are already added to the project created using C1 ASP.NET MVC Web Application template. There is no need to explicitly add the DLL to the project. But, you need to manually integrate the assembly into your MVC project which is created using Visual Studio templates, in order to use these controls.

Samples

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

C:\Users\<username>\Documents\ComponentOne Samples\ASP.NET MVC.

> To run these samples, ensure that the Copy Local property of the assembly reference, **C1.Web.Mvc.dll, C1.Web.Mvc.Finance.dll**, and **C1.Web.Mvc.FlexSheet.dll**, **C1.Web.Mvc.MultiRow.dll**, **C1.Web.Mvc.FlexViewer.dll** and **C1.Web.Mvc.Olap.dll** is set to True

Install C1 Scaffolder Visual Studio extension

- The **C1 Scaffolder** extension gets automatically installed with the C1Studio. You can check if the Visual Studio extension is installed or not by navigating to **Tools** | **Extensions and Updates**.

System Requirements

The following must be installed on your system, in order to install ASP.NET MVC Edition.

- **.NET 4** or above.
- **ASP.NET MVC 3** or above.
- **Visual Studio 2012** or above.

> For ASP.NET Core 2.0 MVC applications, **Visual Studio 2015** and **.NET Framework 4.5** (or above) are required.

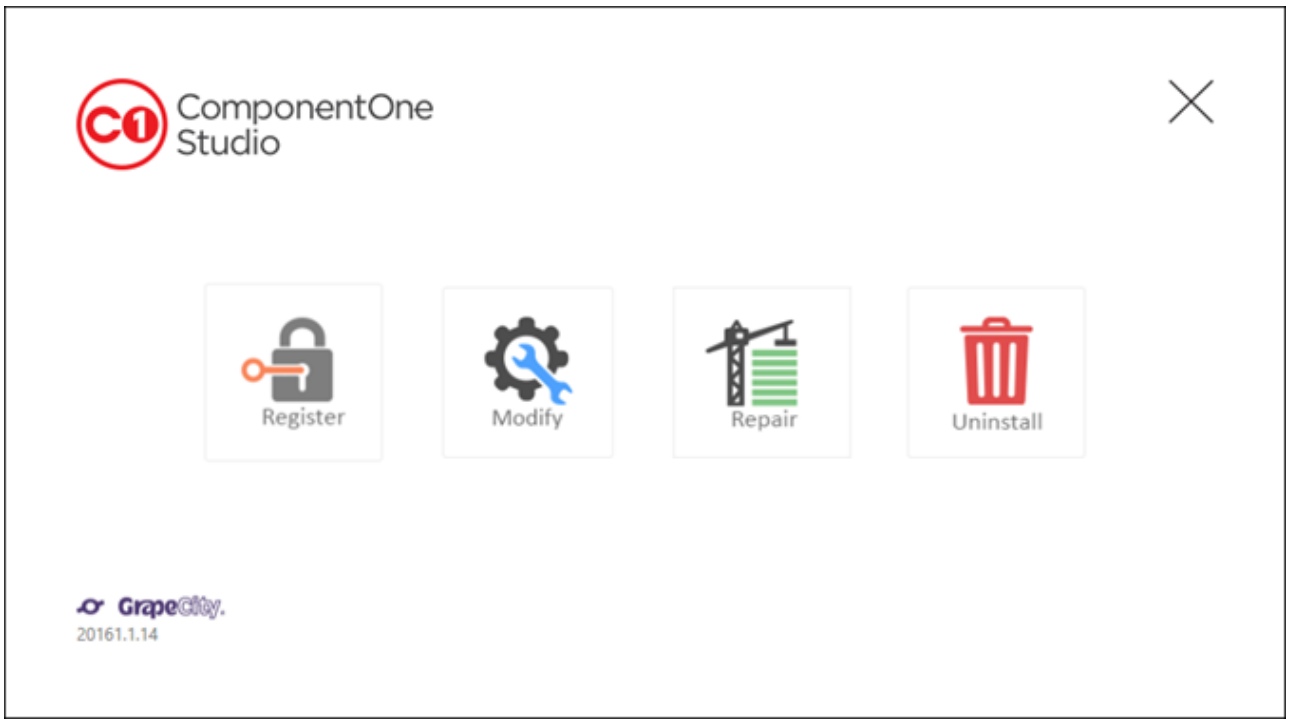
Browser Support

- Internet Explorer 9 and above.
- Microsoft Edge
- Mozilla Firefox
- Safari
- Chrome

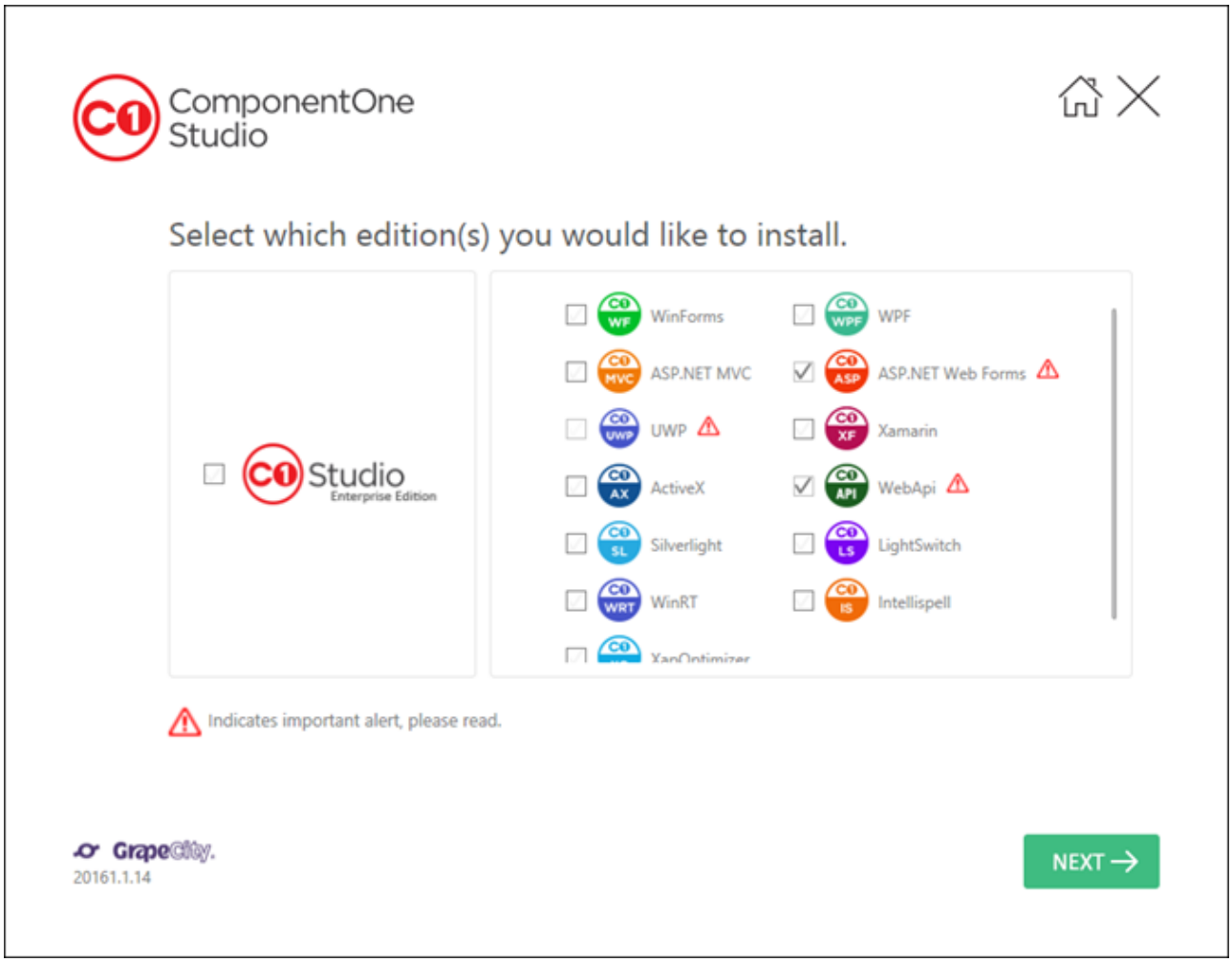
Uninstalling ASP.NET MVC Edition

To uninstall **ASP.NET MVC Edition**

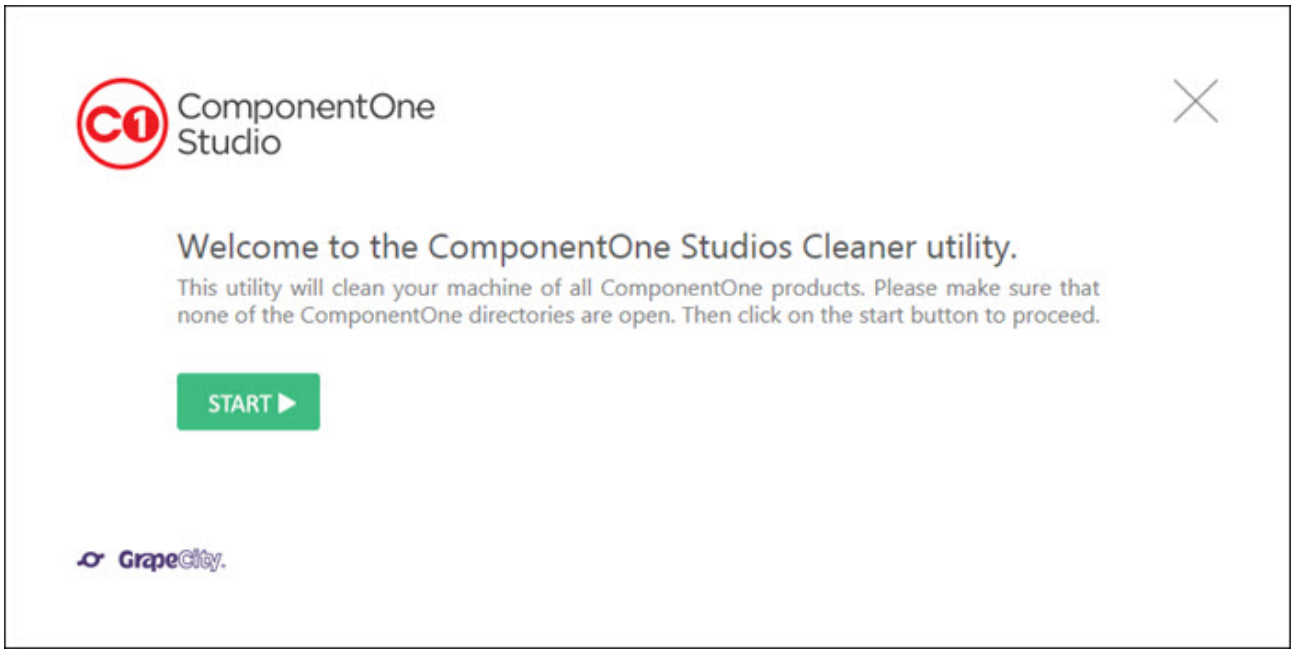
1. Open the **Control Panel** from the Start menu and then select **Uninstall a Program** under **Programs**.
2. Select **ComponentOne Studio** from the list and click **Uninstall** button.
3. In the Installer window that appears, click **Modify**.



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



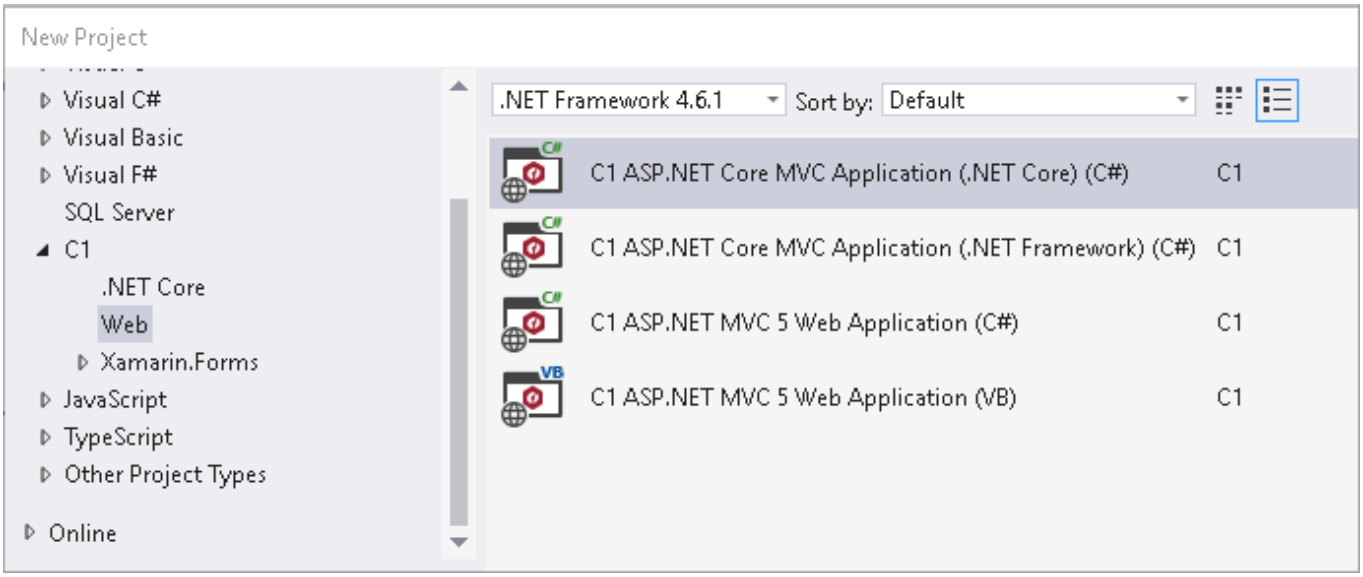
You can also use **ComponentOne Studios Cleaner** utility to clean your machine of all the **ComponentOne** products. All you need to do is download and install the utility from here, run the cleaner, and click **Start** to clean all the registries and directories of **ComponentOne Studio**.



C1 ASP.NET Project Templates

When you install **C1StudioInstaller.exe** on your system, we provide different project templates that are available for both **C#** and **VB.NET** for all the supported ASP.NET MVC versions. These templates help you to easily create an MVC application using C1 ASP.NET MVC controls. All the templates include a Project Wizard, allowing you to customize project settings before creating an application.

- **C1**
  - **.NETCore**
  - **Web**



These project templates are similar to the ASP.NET MVC Web Application template available in Visual Studio. However, our templates provide a simple web application that includes our control references. These generated

applications contain all the necessary project files, assembly references, and license information to reduce manual work in project creation.

The table below lists the C1 ASP.NET MVC project templates.

Project Template	Description
.NETCore	
C1 ASP.NET Core MVC Application(.NETCore)	Creates a C1 ASP.NET Core MVC web application.
Web - Consists of all the .NET Core templates, and the following ASP.NET templates	
C1 ASP.NET MVC 5 Web Application (C#)	Creates a C1 ASP.NET MVC 5 web application.
C1 ASP.NET MVC 5 Web Application (VB)	Creates a C1 ASP.NET MVC 5 web application.
C1 ASP.NET Core MVC Application(.NETFramework)	Creates a C1 ASP.NET Core MVC web application with .NET Framework.

The table below list the Project Types available with our MVC components.

Project Types	Description
Standard	Creates a standard C1 MVC Application.
Ajax Binding	Creates a responsive application with AJAX binding. The application includes a sample data and a view code to explain the implementation of AJAX binding in the FlexGrid control.
Model Binding	Creates an application with model binding. The application includes a sample data and a view code to explain the implementation of Model binding in the FlexGrid control.
Spreadsheet	Creates a business application with the FlexSheet control. The application includes Font.cs and Sale.cs data class and view code with all the necessary references to add data to FlexSheet.

Configuring NuGet Package Sources

ComponentOne MVC references have to be added to the projects that are created using Visual Studio MVC template. ComponentOne ASP.NET Core MVC references are available through NuGet, a Visual Studio extension that automatically adds libraries and references to your project. NuGet package for ASP.NET Core MVC is provided at http://nuget.grapecity.com/nuget/, which is available in Visual Studio on installing ASP.NET MVC Edition.

To Install NuGet

1. Go to https://www.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**.

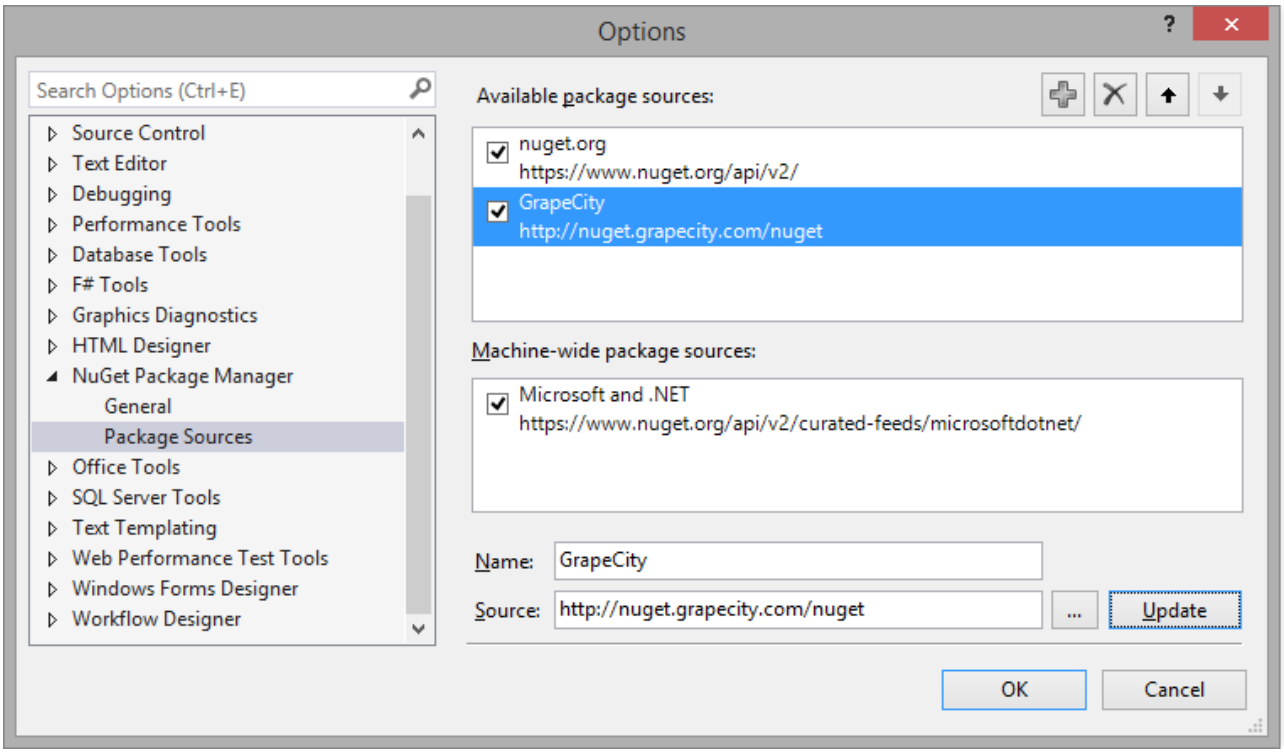
To Manually Create Nuget Feed Source

If you install the ASP.NET MVC Edition, GrapeCity NuGet source is available by default in Package Sources in your Visual

Studio. Otherwise, you need to manually add GrapeCity NuGet source URL to Package Sources in Visual Studio.

Complete the following steps to add NuGet feed URL to your NuGet settings in Visual Studio.

1. From the Tools menu, select **NuGet Package Manager | Package Manager Settings**. The **Options** dialog box appears.
2. In the left pane, select **Package Sources**.
3. Click the  button in the top right corner. A new source is added under **Available Package Sources**.
4. Set a **Name** of the new package source and the **Source** as http://nuget.grapecity.com/nuget/.



5. Click **Update | OK.**

NuGet package source has been created.

Licensing

ASP.NET MVC Edition is a part of ComponentOne Studio Enterprise and follows the standard subscription model like all other ComponentOne products. The licensing information for MVC 3, 4, and 5 applications is contained in **Licenses.licx** file, while ASP.NET Core MVC applications use application based license.

- ASP.NET
- ASP.NET Core

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 application 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?

You can work on multiple applications using the same license. You can even distribute your application across multiple users without worrying about paying any royalty or user fees for redistribution. 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 C:\Program Files (x86)\ComponentOne\C1StartMenu.

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 which 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 via 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/Activation**) .
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.

ASP.NET

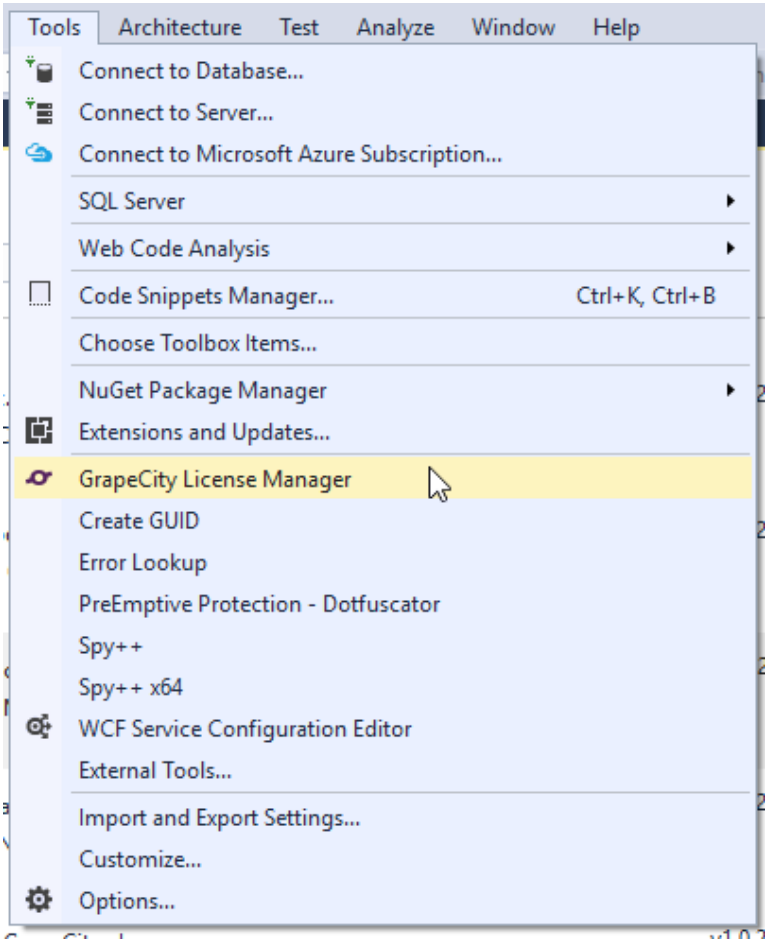
The licensing information for MVC 3, 4, and 5 applications is contained in **Licenses.licx** file. These licenses are bound to specified applications and are to be validated before execution.

- **To Generate License using GrapeCity License Manager Add-in**
- **To manually add the license file to the Application**

License key for the ASP.NET applications created on Visual Studio is generated through GrapeCity License Manager add-in provided in Visual Studio. Whereas, licenses for the applications created on IDE other than Visual Studio (VSCode, Sublime or Eclipse) can be generated on ComponentOne Website.

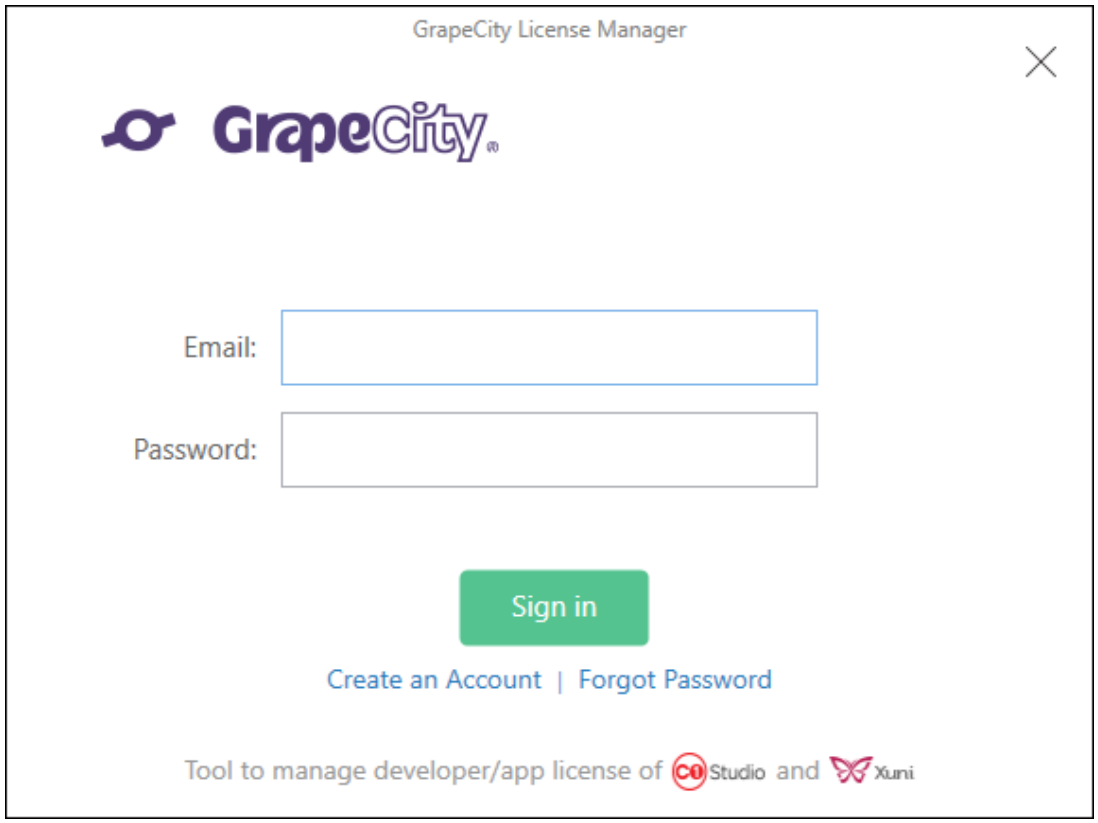
To Generate License using GrapeCity License Manager Add-in

The add-in for generating license is available for all the MVC applications which are created in Visual Studio.
The add-in is visible in options within **Tools** menu in Visual Studio.

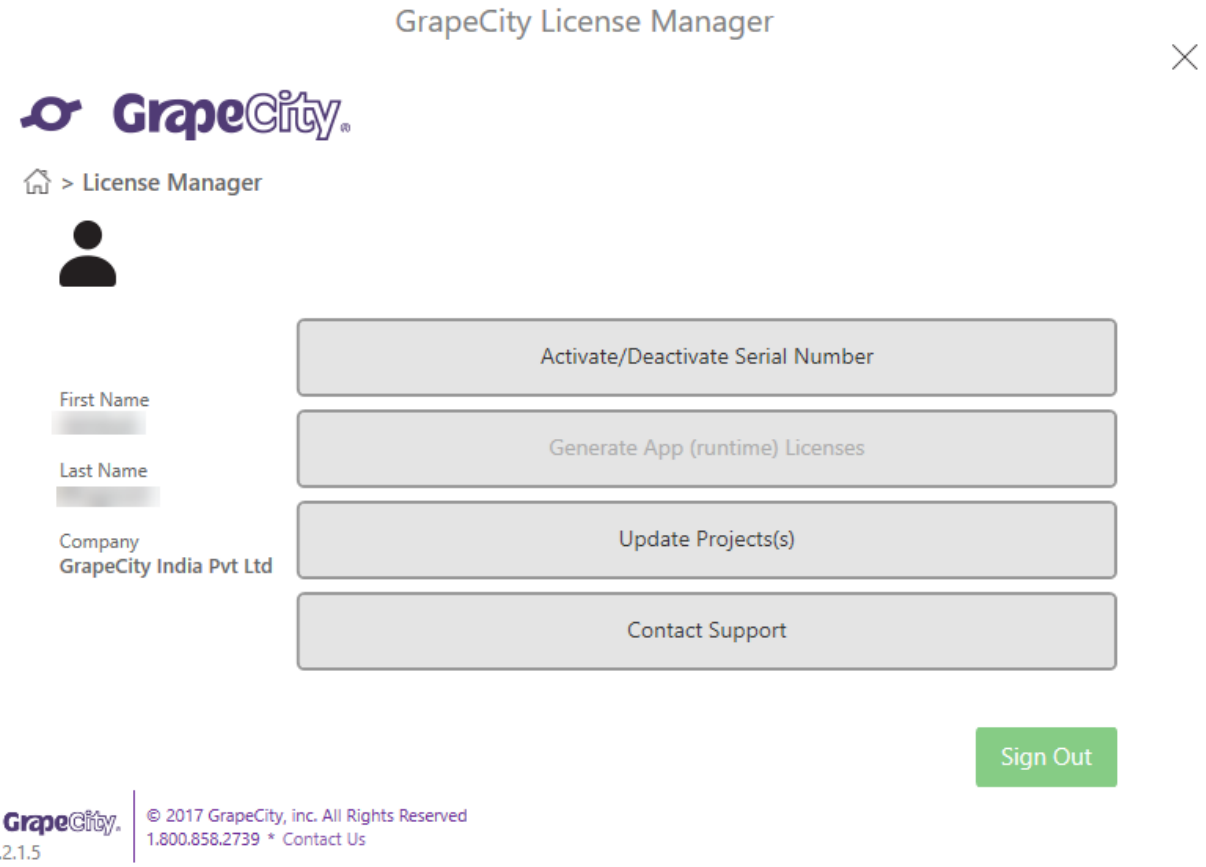


Complete the following steps to generate a trial or full license for your ASP.NET Core applications using Visual Studio add-in:

1. Create a new ASP.NET MVC application. For more information. see Creating a new MVC Application.
2. Add the required control references to your application.
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. You login information will be cached for 30 days in Visual Studio.

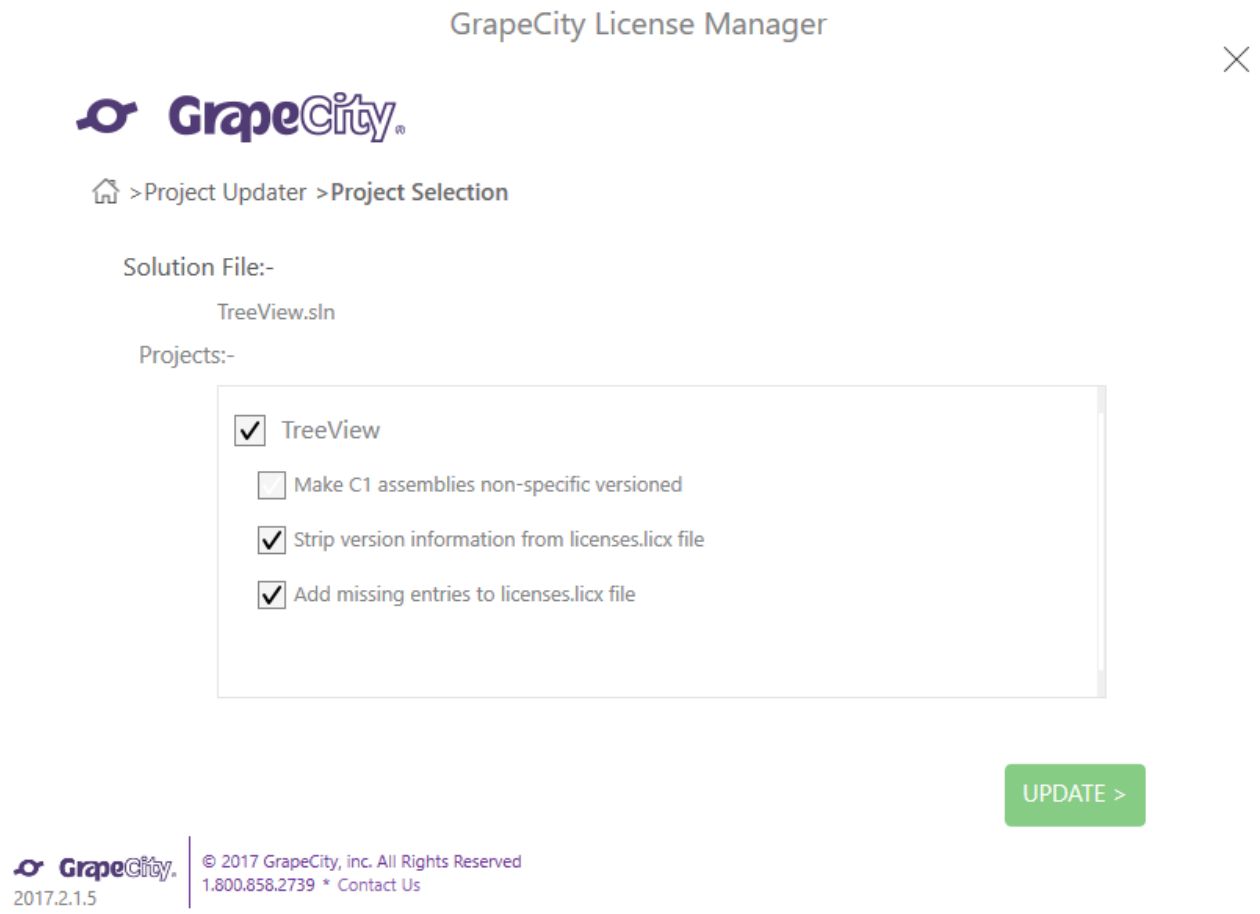


- **Activate/Deactivate Serial Number** - Allows the users to activate or deactivate the serial number using the

internet, C1 Website, By Email, or Over the phone.

- **Generate App (runtime) Licenses** - Allows the users to generate and activate license for each MVC application you are working on your system. This option is disabled for ASP.NET applications.
- **Update Project(s) -** Select one or more projects from the loaded solution to:
  - Modify all references to ComponentOne assemblies so that they are not version specific.
  - Update all entries in the licenses.licx file so that they are not version-specific.
  - Add entries in the licenses.licx file for referred dlls which have been already added.
- **Contact Support** - 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 **Update Project(s)** option.
7. In the GrapeCity License Manager window, select the license updates according to your own requirements. In our case, we have selected **Strip version information from license.licx file** and **Add missing entries to licenses.licx file** options.



8. In the GrapeCity License Manager window, click **UPDATE** to update the license information in your application. Once you click on **UPDATE**, a success message appears in the GrapeCity License Manager window.

   The **licenses.licx** file is updated and placed in the same location as the project file and it is an embedded resource to the project.

To manually add the license file to the Application

1. In the **Solution Explorer**, right click the project name 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. Paste the following code in the text file:

```
licenses.licx
```

```
C1.Web.Mvc.LicenseDetector, C1.Web.Mvc
```

5. Few additional steps may be required for using **FinancialChart, FlexSheet, MultiRow, FlexViewer and OLAP** controls in your application. In the **licenses.licx** file, press enter after **C1.Web.Mvc.LicenseDetector, C1.Web.Mvc**, and paste the following code:

licenses.licx

```
C1.Web.Mvc.Finance.LicenseDetector, C1.Web.Mvc.Finance
C1.Web.Mvc.Sheet.LicenseDetector, C1.Web.Mvc.FlexSheet
C1.Web.Mvc.Viewer.LicenseDetector, C1.Web.Mvc.FlexViewer
C1.Web.Mvc.Olap.LicenseDetector, C1.Web.Mvc.Olap
C1.Web.Mvc.MultiRow.LicenseDetector, C1.Web.Mvc.MultiRow
```

ASP.NET Core

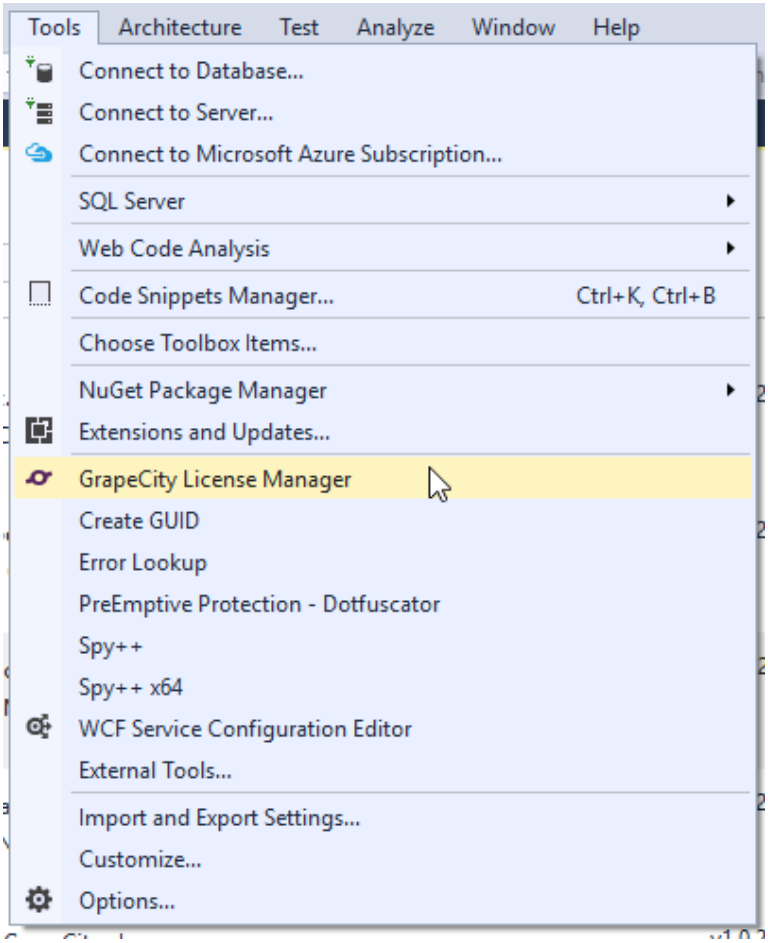
Applications that are created on ASP.NET Core require a unique license key. These licenses are bound to specified applications and are to be validated at runtime.

- **To Generate License using GrapeCity License Manager Add-in**
- **To Generate License on C1 Website**

License key for the ASP.NET Core applications created on Visual Studio is generated through **GrapeCity License Manager** add-in provided in Visual Studio. Whereas, licenses for the applications created on IDE other than Visual Studio (VSCode, Sublime or Eclipse) can be generated on C1 Website.

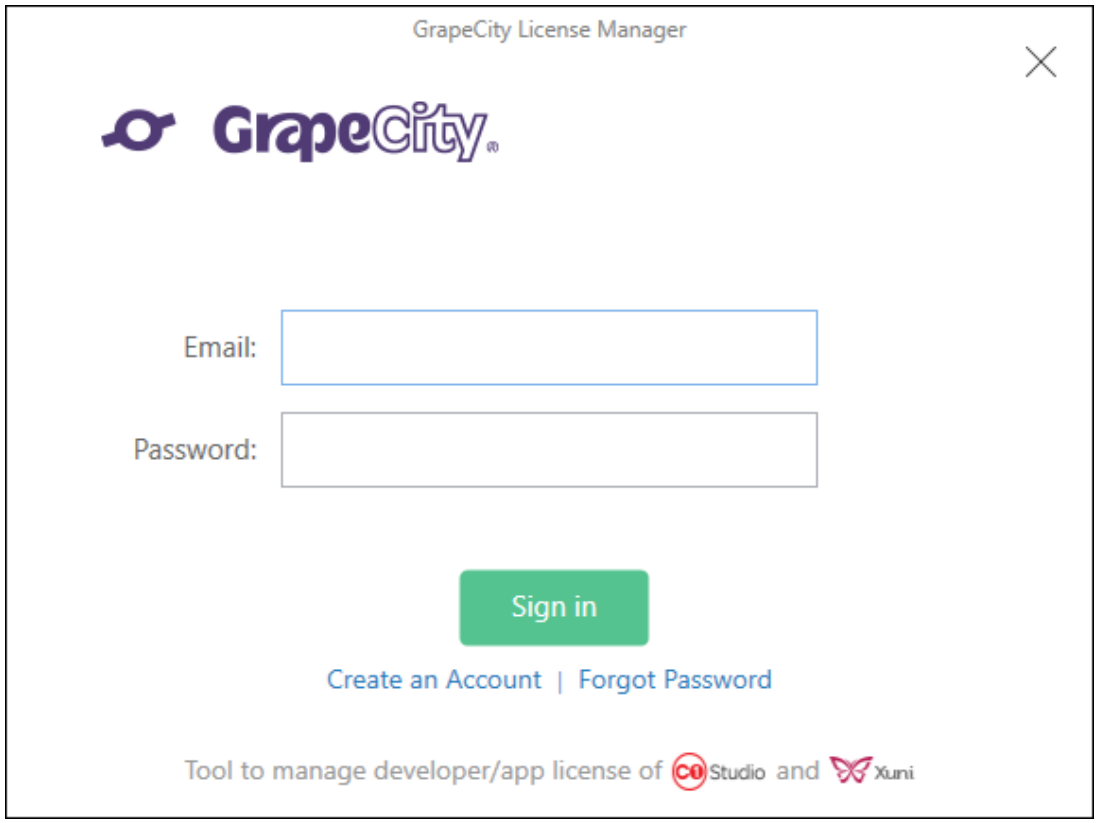
To Generate License using GrapeCity License Manager Add-in

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

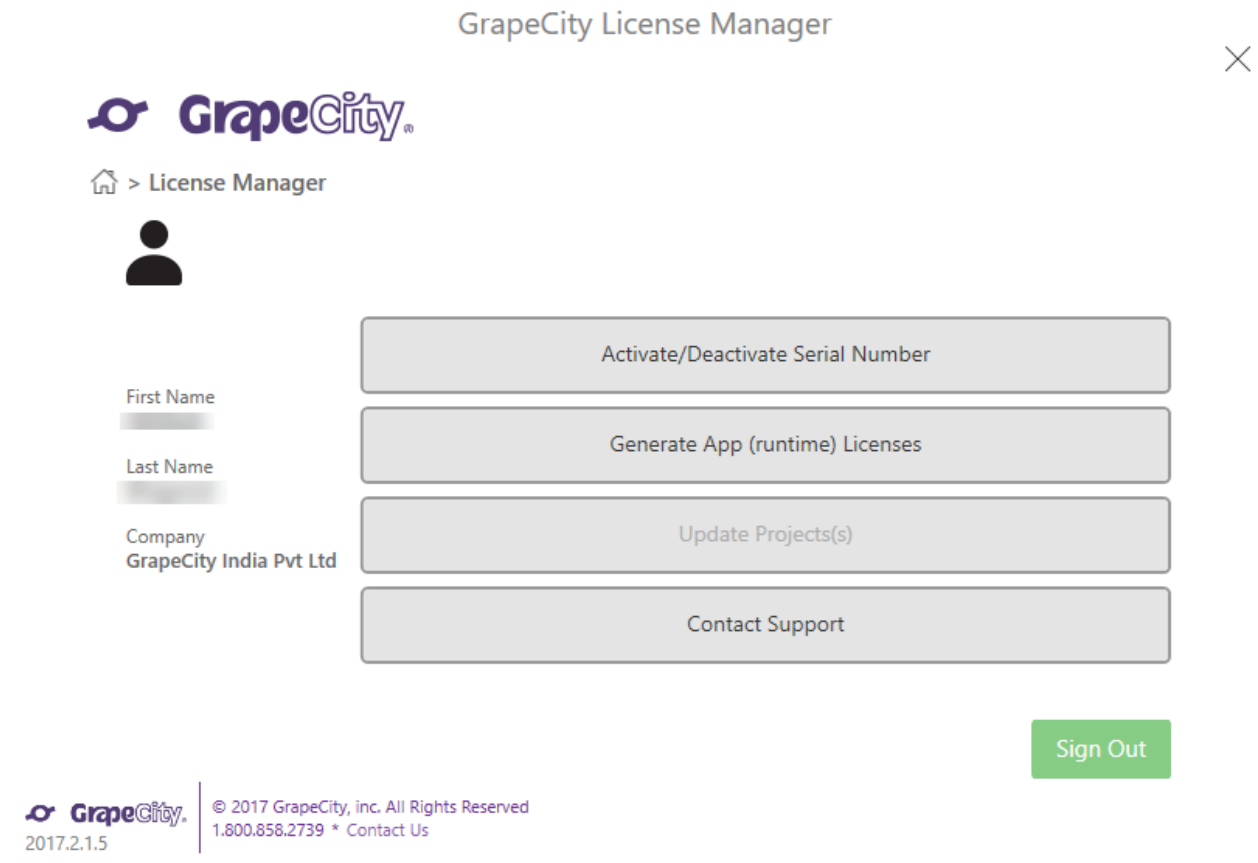


Complete the following steps to generate a trial or full license for your ASP.NET Core applications using Visual Studio add-in:

1. Create a new ASP.NET Core MVC application. For more information. see Creating a new MVC Application.
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. You login information will be cached for 30 days in Visual Studio.

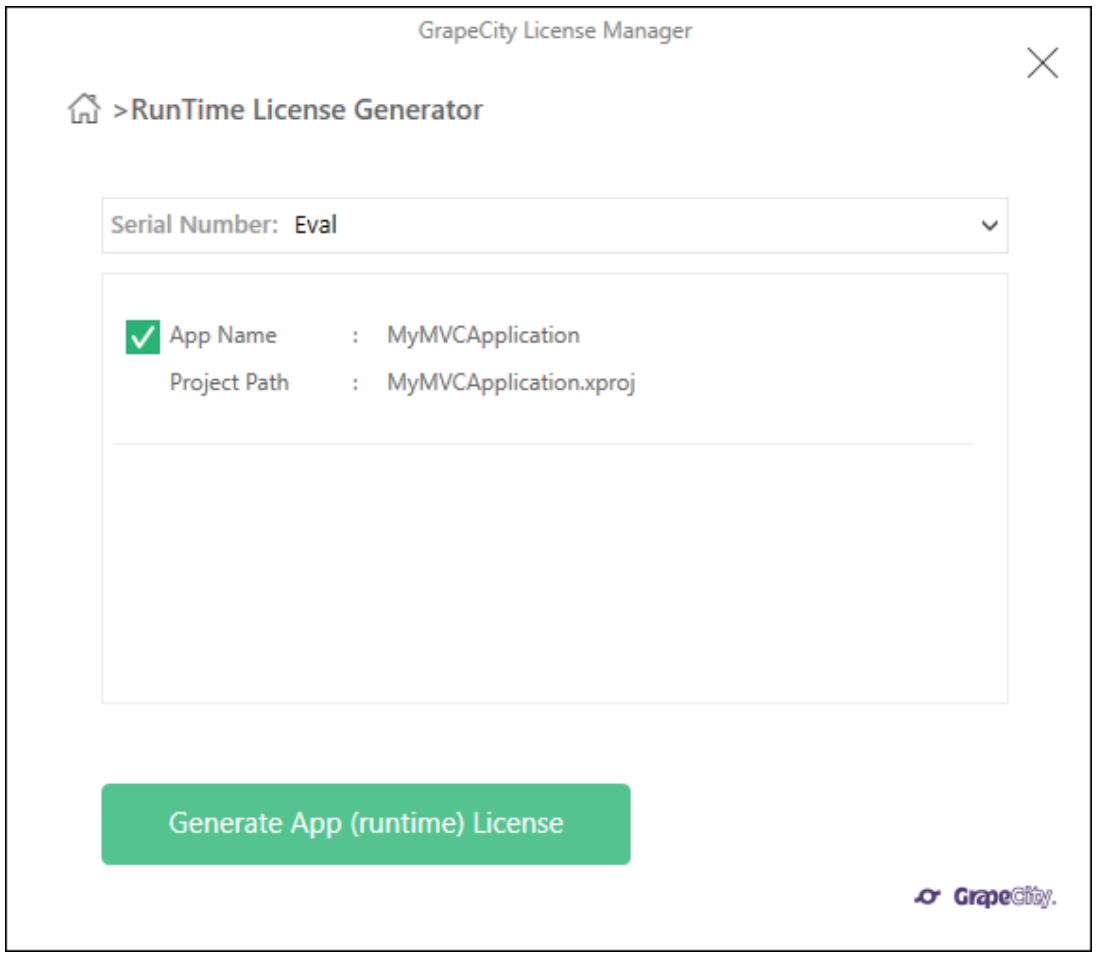


- **Activate/Deactivate Serial Number** - Allows the users to activate or deactivate the serial number using the

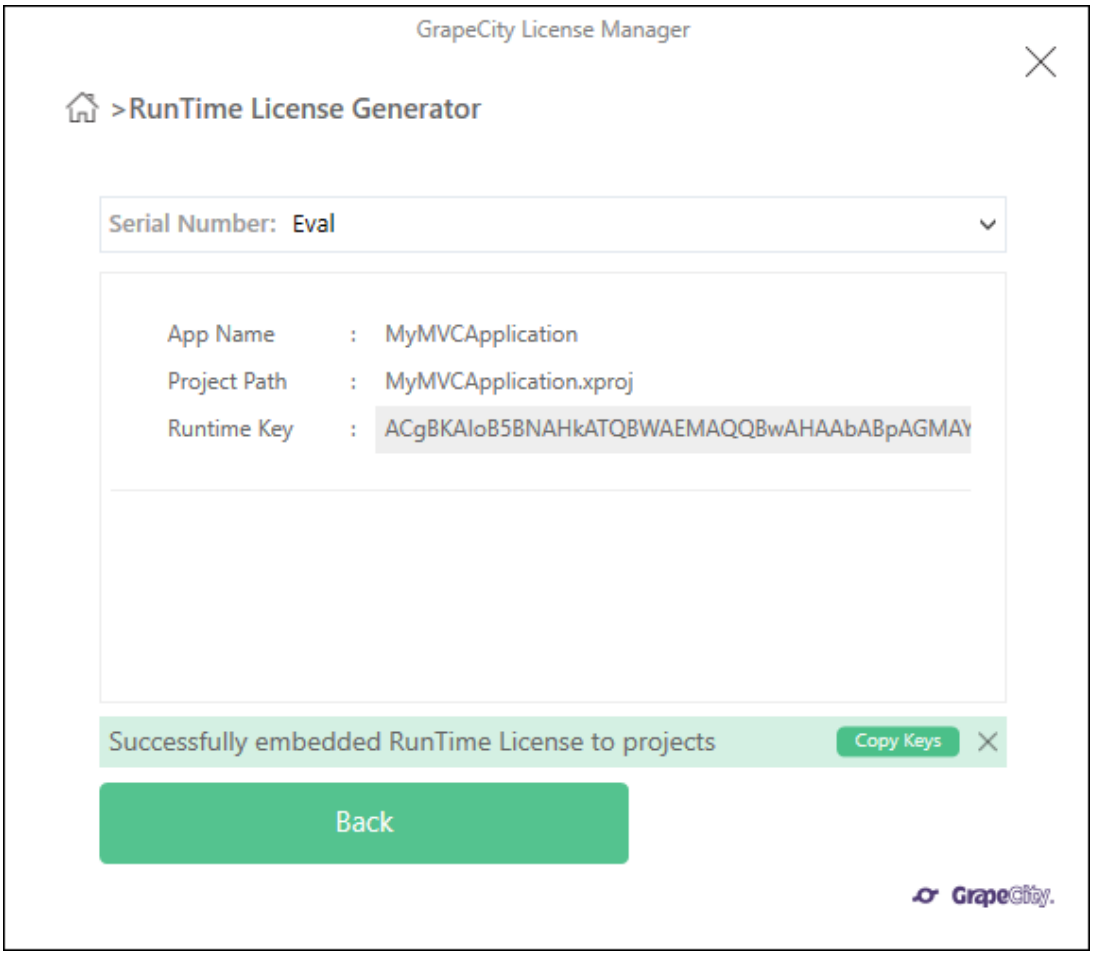
internet, C1 Website, By Email, or Over the phone.

- **Generate App (runtime) Licenses** - Allows the users to generate and activate license for each MVC application you are working on your system. This option is disabled for ASP.NET applications.
- **Update Project(s) -** Select one or more projects from the loaded solution to:
  - Modify all references to ComponentOne assemblies so that they are not version specific.
  - Update all entries in the licenses.licx file so that they are not version-specific.
  - Add entries in the licenses.licx file for referred dlls which have been already added.
    *This option is disabled for ASP.NET Core applications.
- **Contact Support** - Allows the user to open http://supportone.componentone.com/ website where the users can communicate with the support team for any support related issues.

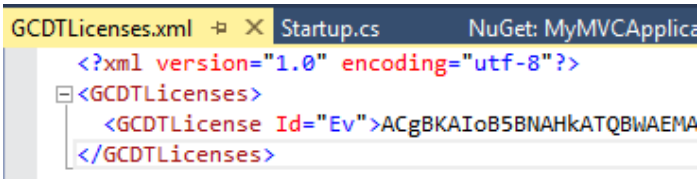
6. In the GrapeCity License Manager window, select **Generate App (runtime) Licenses** option to generate a license for your MVC application.
   Once you select this option the tool will detect the relevant C1 assemblies that are being used in the project or active solution.
7. In the GrapeCity License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

MVC for ASP.NET Core 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 **ASP.NET MVC 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 MVC for ASP.NET Core 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 ASP.NET Core MVC applications.

2. Click **Register a Product**.
3. Enter the **Serial Number** and **Purchase Date** and click **Register Product**.

To Generate License on C1 Website

Complete the following steps to generate a trial or full license for your applications (created on IDE other than Visual Studio) on C1 website:

1. Create a new ASP.NET Core MVC application (Refer to Creating a new MVC Application for detailed steps).
2. Add the required NuGet packages to your application through the **NuGet Package Manager** (Refer to Installation for steps to add Nuget packages).
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.

   > **Note:** To find your application's name, in the **Solution Explorer** right click your project's name 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. Replace the content of the **License.cs** file with the code given below.

      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.Mvc.LicenseManager.Key = License.Key;
                      C1.Web.Mvc.Finance.LicenseManager.Key = License.Key
                      C1.Web.Mvc.Sheet.LicenseManager.Key = License.Key
                      C1.Web.Mvc.Viewer.LicenseManager.Key = License.Key
                      C1.Web.Mvc.MultiRow.LicenseManager.Key = License.Key
                      C1.Web.Mvc.Olap.LicenseManager.Key = License.Key
              }
      ```

> **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.

Configuring your MVC Application

This section demonstrates how to create and configure your MVC application using ComponentOne template and Visual Studio template in Visual Studio. Make sure you check the System Requirements before proceeding.

- Using ComponentOne Template
- Using Visual Studio Template

Merits of ComponentOne Template over standard Visual Studio Template.

ComponentOne template has following merits:

ASP.NET

1. Registers the resources required to use MVC Edition controls.
2. Adds entries to the web.config file automatically.
3. Adds ASP.NET MVC Edition references to the project automatically.
4. Adds license.licx file to the project automatically.

ASP.NET Core

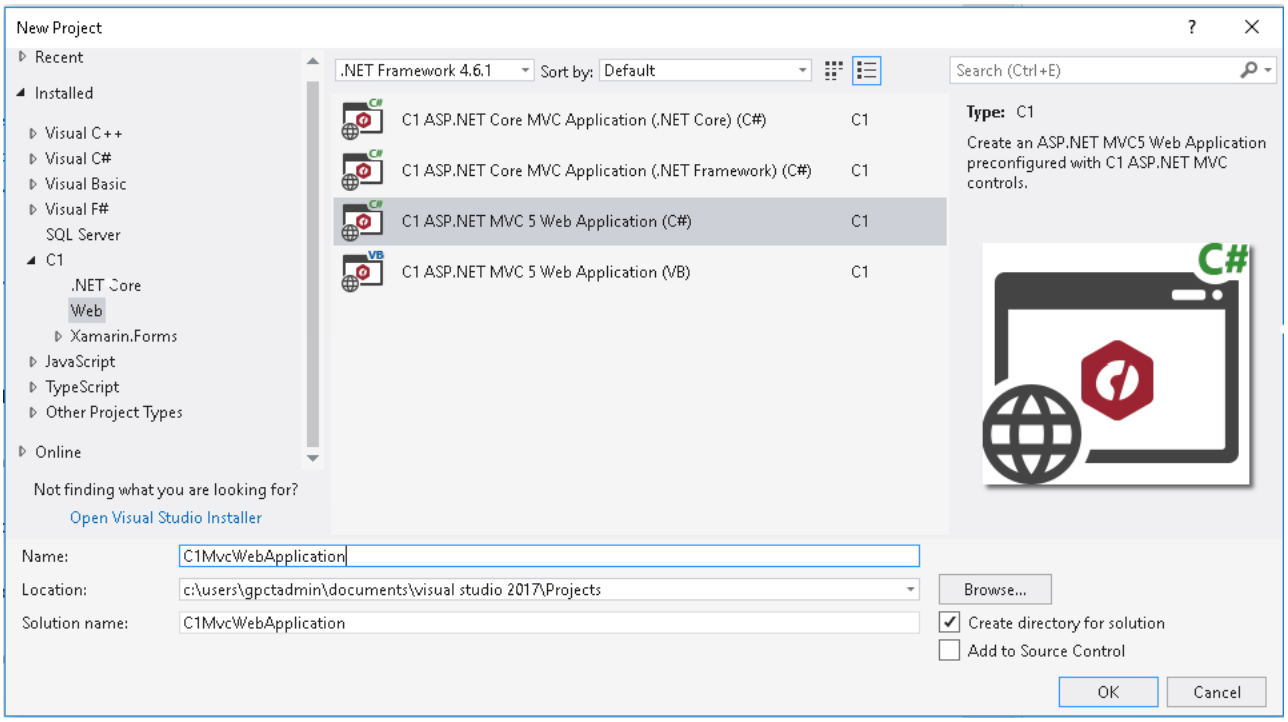
1. Registers the resources required to use MVC Edition controls.
2. Adds ASP.NET Core MVC Edition packages to the project automatically.
3. Adds the required MVC Edition namespace to the _ViewImports.cshtml file automatically.

Using ComponentOne Template

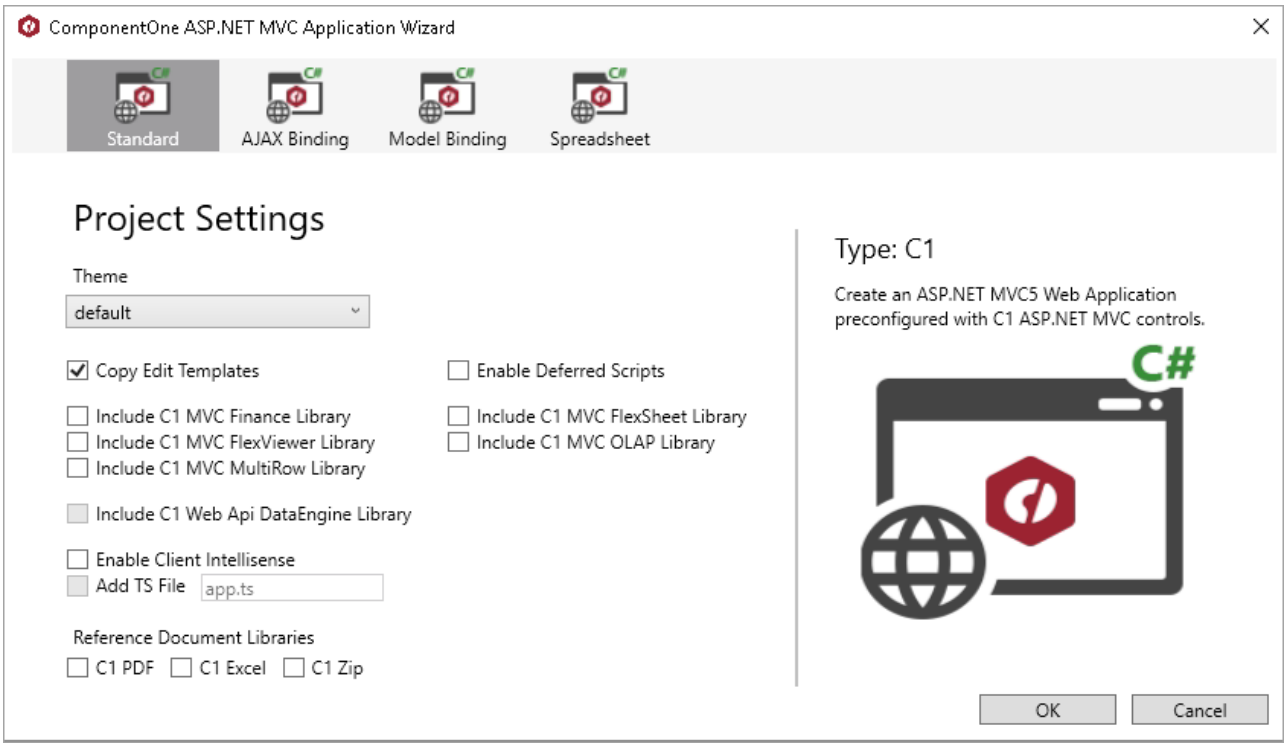
Complete the following steps to create a new MVC Application using C1 template:

ASP.NET

1. Select **File | New | Project.**
2. Under **Installed | Templates**, select **C1 | Web | C1 ASP.NET MVC 5 Web Application (C# or VB)** to create a C1 ASP.NET MVC Web application.



3. Set a **Name** and **Location** for your application.
4. Click **OK**. ComponentOne ASP.NET MVC Application Wizard appears.

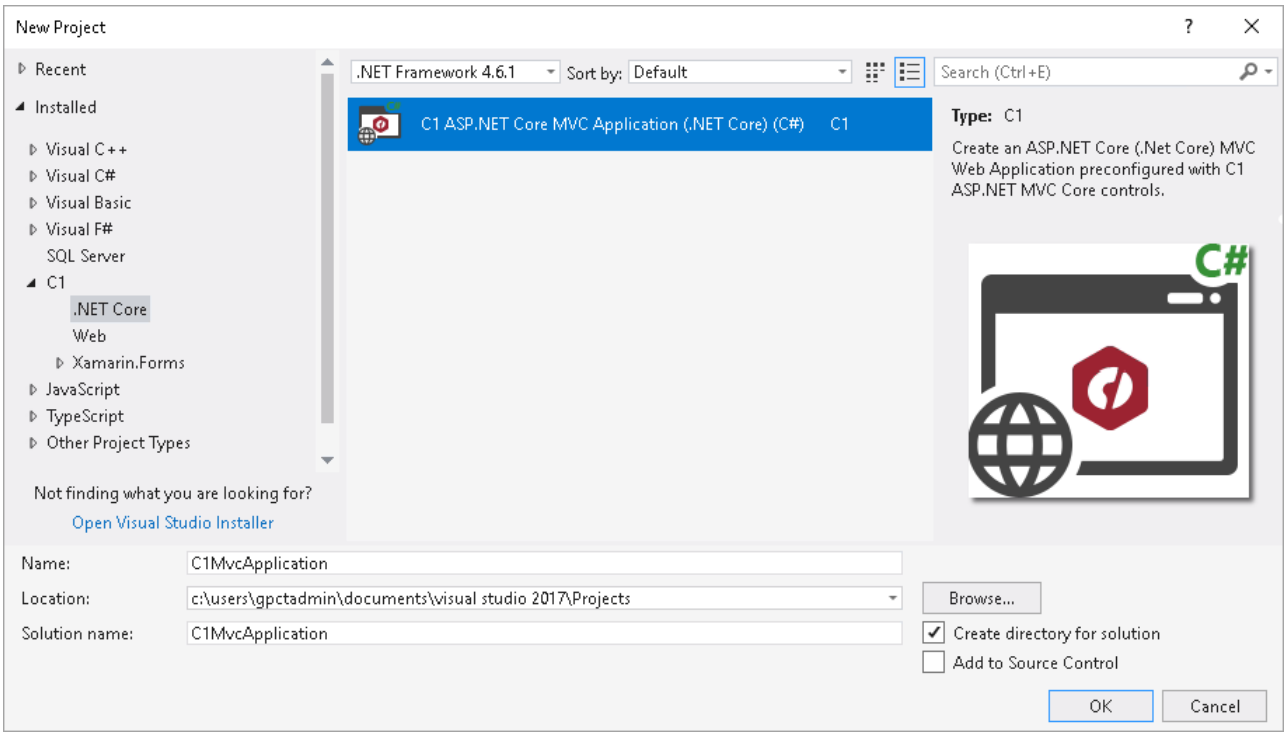


5. Select a project type. In our case, we have selected **Standard**. For more information, see C1 ASP.NET Project Templates.
6. Select a **Theme** and the document libraries to reference.
7. Click **OK**.
8. A new project configured to work with ASP.NET MVC Edition is created.

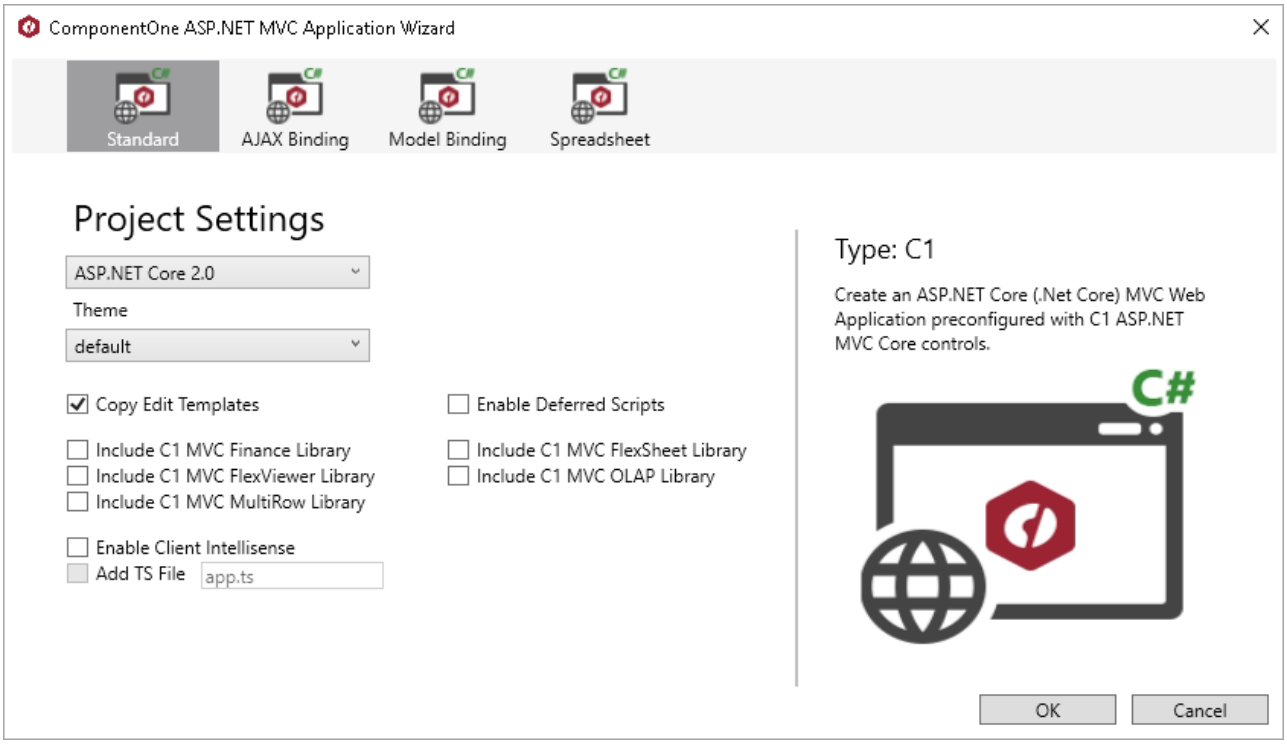
licenses.licx file is automatically added to your application. For more information on how to add the license file to the project, refer to Licensing.

ASP.NET Core

1. Select **File | New | Project** in Visual Studio 2015.
2. Under **Installed | Templates**, select **C1 | Visual C# | .NET Core | C1 ASP.NET Core MVC Application** to create a C1 ASP.NET MVC Web application. For more information, see C1 ASP.NET Project Templates.



3. Set a **Name** and **Location** for your application.
4. Click **OK**. ComponentOne ASP.NET MVC Application Wizard appears.



5. Select a project type. In our case, we have selected **Standard**. For more information, see C1 ASP.NET Project Templates.
6. Select the **ASP.NET Core framework** version for your application.
7. Select a **Theme**.
8. Click **OK**. A new project configured to work with ASP.NET Core MVC Edition is created.
9. To generate the runtime license for applications created in Visual Studio, use **GrapeCity License Manager** Add-in available in **Tools** menu in Visual Studio.

   However, you can also generate runtime license through C1 website:

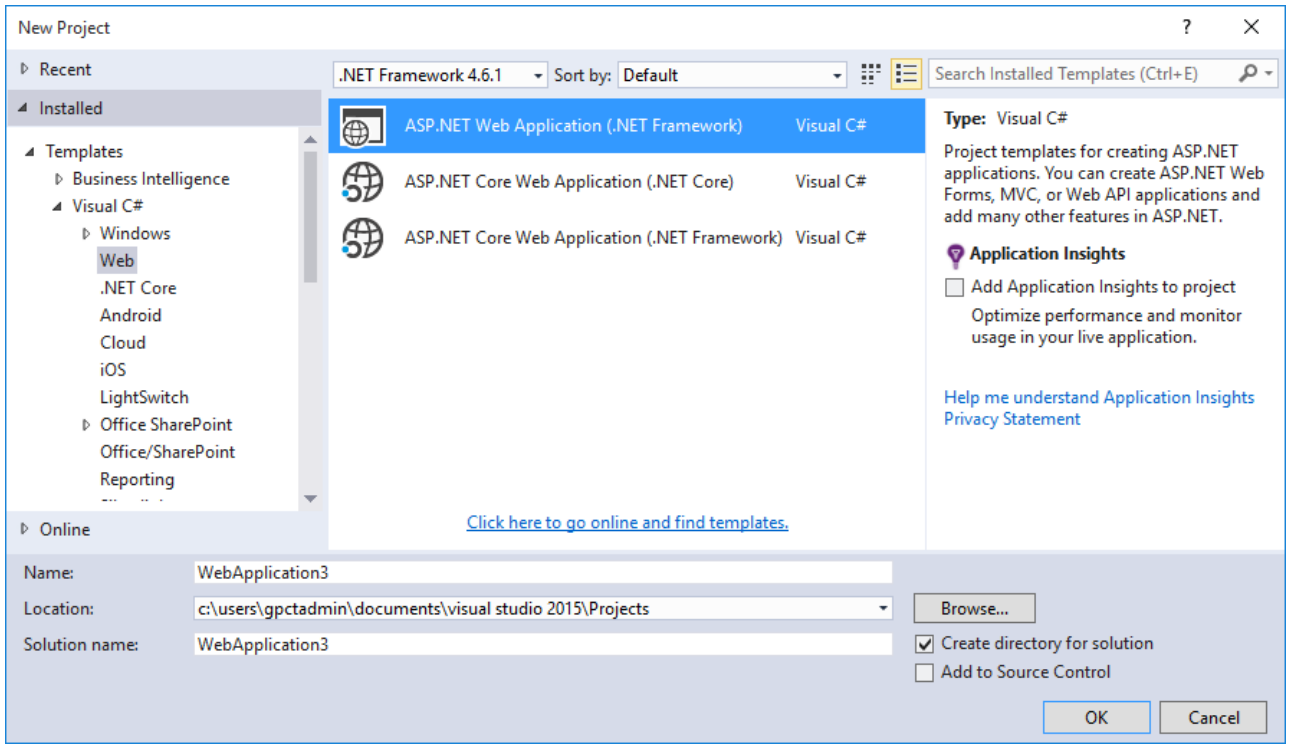
- Create a **License.cs** in your project (by right-clicking the **Project Name** in Solution Explorer and selecting **Add New | Class**).
- Add the license key generated on C1 website in this class.
- From the Solution Explorer, open **Startup.cs** file and assign the key to it (For more information on how to add license key to the project, see Licensing).

Using Visual Studio Template

Complete the following steps to create a new MVC Application using Visual Studio template:

ASP.NET

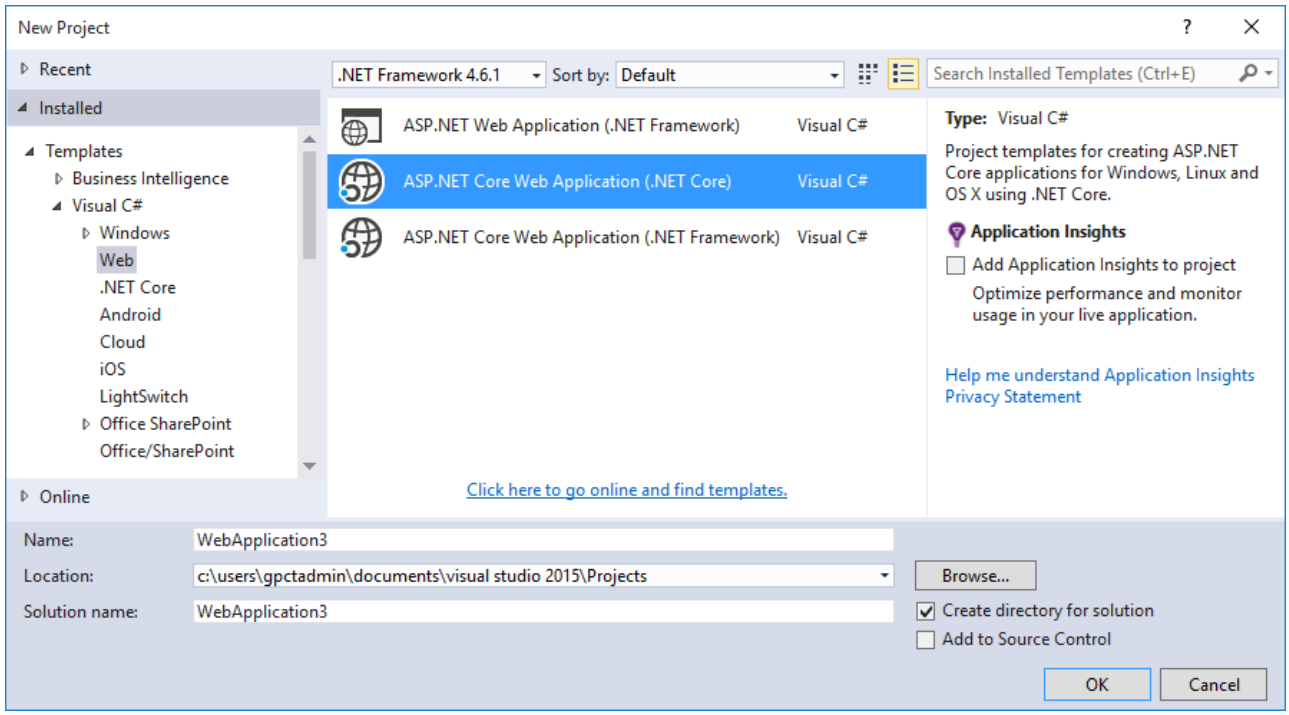
1. Select **File | New | Project.**
2. Under installed templates, select **Visual C# | Web | ASP.NET Web Application (.NET Framework)**.



3. Set a **Name** and **Location** for your application. Click **OK**.
4. In the **New ASP.NET Web Application (.NET Framework)** dialog, select **MVC,** and then click **OK**.

ASP.NET Core

1. Select **File | New | Project.**
2. Under installed templates, select **Visual C# | Web | ASP.NET Core Web Application (.NET Core or .NET Framework).**



3. Set a **Name** and **Location** for your application. Click **OK**.
4. In the **New ASP.NET Core Web Application** dialog, select **Web Application** under **ASP.NET Core Templates,** and click **OK.**

Configuring the project

Few additional steps are required to configure the project created using the basic Visual Studio template.

ASP.NET

1. From the **Solution Explorer**, expand the folder **Views** and double click the `web.config` file to open it.
2. Add the following markups in <namespaces></namespaces> tags, within the `<system.web.webPages.razor>` `</system.web.webPages.razor>` tags.

HTML

```
<add namespace="C1.Web.Mvc" />
<add namespace="C1.Web.Mvc.Fluent" />
```

3. (Optional) To add specific controls in your application, you need to add the following markups in <namespaces> </namespaces> tags below the C1.Web.Mvc markup. For example, in the image below we have added markups for Financial Chart control.

Control	Markup
Financial Chart	`<add namespace="C1.Web.Mvc.Finance" />` `<add namespace="C1.Web.Mvc.Finance.Fluent" />`
FlexSheet	`<add namespace="C1.Web.Mvc.Sheet" />` `<add namespace="C1.Web.Mvc.Sheet.Fluent" />`
OLAP	`<add namespace="C1.Web.Mvc.Olap" />` `<add namespace="C1.Web.Mvc.Olap.Fluent" />`
MultiRow	`<add namespace="C1.Web.Mvc.MultiRow" />` `<add namespace="C1.Web.Mvc.MultiRow.Fluent" />`
FlexViewer	`<add namespace="C1.Web.Mvc.Viewer" />`

	`<add namespace="C1.Web.Mvc.Viewer.Fluent" />`

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc, Version=5.1.0.0, Culture=neutral
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Optimization"/>
      <add namespace="System.Web.Routing" />
      <add namespace="C1MvcWebApplication3" />
      <add namespace="C1.Web.Mvc" />
      <add namespace="C1.Web.Mvc.Fluent" />
      <add namespace="C1.Web.Mvc.Finance" />
      <add namespace="C1.Web.Mvc.Finance.Fluent" />
    </namespaces>
  </pages>
</system.web.webPages.razor>

<appSettings>
  <add key="webpages:Enabled" value="false" />
</appSettings>
```

4. Save the changes made to Web.config file.
5. In the **Solution Explorer**, right click the project and select **Add | New Item.** The **Add New Item** dialog appears.
6. In the **Add New Item** dialog, select **C# | General** and select **Text File** in the right pane.
7. Name the text file as **licenses.licx**.
8. Paste the following code to the text file:

License.licx
`C1.Web.Mvc.LicenseDetector, C1.Web.Mvc`

9. To use any specific control in your application, add the following code in Licenses.licx based on the control. For more information, see Licensing.

Control	Markup
Financial Chart	`C1.Web.Mvc.Finance.LicenseDetector, C1.Web.Mvc.Finance`
FlexSheet	`C1.Web.Mvc.Sheet.LicenseDetector, C1.Web.Mvc.FlexSheet`
OLAP	`C1.Web.Mvc.Olap.LicenseDetector, C1.Web.Mvc.Olap`
MultiRow	`C1.Web.Mvc.MultiRow.LicenseDetector, C1.Web.Mvc.MultiRow`
FlexViewer	`C1.Web.Mvc.Viewer.LicenseDetector, C1.Web.Mvc.FlexViewer`

10. Add the ASP.NET MVC Edition references to the project. In the **Solution Explorer**, right click **References** and select **Add Reference**. Browse to the location- **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin**, select **C1.Web.Mvc.dll** and click **Add**. Set the **Copy Local** property of the C1.Web.Mvc.dll to **True**.

11. (Optional) To use specific controls in your MVC Application, you need to add additional references based on the control.

Control	Assembly (Location - C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin)
Financial Chart	`C1.Web.Mvc.Finance.dll`
FlexSheet	`C1.Web.Mvc.FlexSheet.dll`
OLAP	`C1.Web.Mvc.Olap.dll`
MultiRow	`C1.Web.Mvc.MultiRow.dll`
FlexViewer	`C1.Web.Mvc.FlexViewer.dll`

12. After completing the steps above, register the resources for the controls to be used in the application. For more information, see Registering Resources.

ASP.NET Core

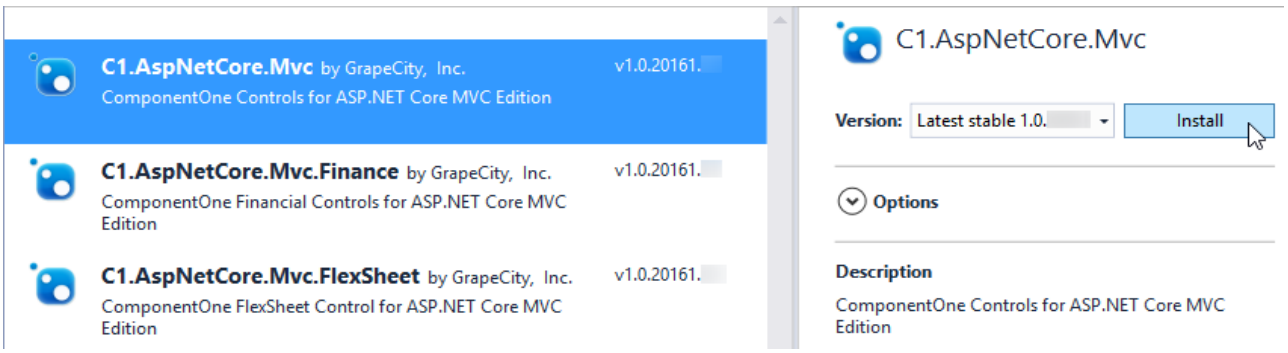
1. From the **Solution Explorer**, expand the folder **Views** and double click the _ViewImports.cshtml file to open it.
2. Add the following:

_ViewImports
`@addTagHelper *, C1.AspNetCore.Mvc`

3. (Optional) To add specific controls in your application, you need to add the following markups in **_ViewImports.cshtml**.

Control	Markup
Financial Chart	`@addTagHelper *, C1.AspNetCore.Mvc.Finance`
FlexSheet	`@addTagHelper *, C1.AspNetCore.Mvc.FlexSheet`
OLAP	`@addTagHelper *, C1.AspNetCore.Mvc.Olap`
MultiRow	`@addTagHelper *, C1.AspNetCore.Mvc.MultiRow`
FlexViewer	`@addTagHelper *, C1.AspNetCore.Mvc.FlexViewer`

4. Add the ASP.NET MVC Edition references to the project. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**. In **NuGet Package Manager**, select **GrapeCity** as the Package source. Search for **C1.AspNetCore.Mvc** package, and click **Install**. Refer to Configuring NuGet Package Sources for information on manually configuring GrapeCity NuGet source.



> **Note:** "`C1.AspNetCore.Mvc`" gets added under the "dependencies" within project.json file of your project once you restore the packages.

5. To use specific controls in your application, add the following NuGet Packages based on the control.

Control	Assembly (Location - C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin)
Financial Chart	`C1.AspNetCore.Mvc.Finance`
FlexSheet	`C1.AspNetCore.Mvc.FlexSheet`
OLAP	`C1.AspNetCore.Mvc.Olap`
MultiRow	`C1.AspNetCore.Mvc.MultiRow`
FlexViewer	`C1.AspNetCore.Mvc.FlexViewer`

6. To add a license, right-click the solution name from the Solution Explorer or go to the Tools menu, and then select

GrapeCity License Manager. Optionally, you can also generate runtime license key and add it to your application. For more information, see Licensing topic.

7. After completing the steps above, register the resources for the controls to be used in the application. For more information, see Registering Resources.

Registering Resources

By default, all the resources required to use the available controls get registered, when you create a new application using the ComponentOne MVC template. But, you need to register resources manually in case you create a project using standard Visual Studio templates.

Complete the following steps to register the required resources for using ASP.NET MVC Edition controls:

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click **_Layout.cshtml** to open it.
3. Add the following code between the `<head></head>` tags. This step will register all the MVC controls used in your application, except **FinancialChart, FlexSheet, MultiRow, FlexViewer** and **OLAP** controls.

Using HTML Helpers

Razor

```
@Html.C1().Styles()
@Html.C1().Scripts().Basic()
```

Using Tag Helpers

HTML

```
<c1-styles />
<c1-scripts>
        <c1-basic-scripts />
</c1-scripts>
```

4. (Optional) If you want to use any specific control such as **FinancialChart, FlexSheet, MultiRow, FlexViewer** and **OLAP** control in your application, replace the above code with the following code in `_Layout.cshtml`. You can add or remove the controls depending upon your project requirements.

Using HTML Helpers

Razor

```
@Html.C1().Styles()
@Html.C1().Scripts().Basic().Finance().FlexSheet().FlexViewer().Olap().MultiRow()
```

Using Tag Helpers

HTML

```
<c1-styles />
<c1-scripts>
        <c1-basic-scripts/>
```

```
        <c1-finance-scripts>
        <c1-flex-sheet-scripts />
        <c1-flex-viewer-scripts />
        <c1-olap-scripts />
        <c1-multi-row-scripts />
</c1-scripts>
```

ASP.NET MVC Edition supports **conditional resource registration**. Wherein, you may register only the resources that you wish to use in your application to keep your application lighter. For example, the following code will register the resources required to use FlexChart and FlexPie.

Using HTML Helpers

Razor

```
@Html.C1().Scripts().Basic(b=>b.Grid().Chart())
```

Using Tag Helpers

Razor

```
<c1-scripts> <c1-basic-scripts bundles="Grid, Chart" /> </c1-scripts>
```

Note: If your view is not using _Layout.cshtml, then add the above code to register resources towards the top of your view page.

Styles

To use styles in your MVC application, you need to register the styles (CSS) files which includes the resources for a theme and culture. You can also apply different themes using Theme method, the supported themes are defined in C1.Web.Mvc.Themes static class.

HTML Helpers

_Layout.cshtml

```
@Html.C1().Styles().Theme("cocoa")
```

Tag Helpers

_Layout.cshtml

```
<c1-styles theme="cocoa" />
```

Deferred Scripts

The HTML Script Element (<script>) is used to embed or reference an executable script within an HTML or XHTML document. Scripts without async or defer attributes, as well as inline scripts, are fetched and executed immediately, before the browser continues to parse the page. According to the industry standards, it is recommended that you put all the scripts at the bottom of your HTML page and style sheets at the top. When C1 scripts are registered at the

bottom, the startup scripts of controls also need be registered at the bottom.

You should follow the following process to add deferred scripts in the **_Layout.cshtml** page. You can refer to the image below, to understand the process of adding deferred script.

1. Register styles in the head tag.
2. Register script files before the end of <body> tag.
3. Register startup scripts before the end of <body> tag.

```
<body>
    @Html.C1().FlexGrid()
    @Html.C1().InputDate()
    @Html.C1().FinancialChart()

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")

    @Html.C1().Scripts()
    @RenderSection("scripts", required: false)
    @Html.C1().DeferredScripts()
</body>
</html>
```

The scripts for a MVC control include two parts: necessary scripts files and startup scripts. We provides different methods to enable deferred script for MVC 3/4/5 and ASP.NET Core application. There are 2 scopes through which deferred scripts can be applied: application level and page level. As the name implies, application level means all the ComponentOne controls in this application will use deferred scripts, while page level means only ComponentOne controls defined in the page use deferred script.

HTML Helpers

```
@Html.C1().DeferredScripts()
```

Tag Helpers

```
<c1-deferred-scripts />
```

Deferred Scripts - Switcher

When a user turn on the switcher, the startup script of the control is suspended.

At app level, the current razor page renders at first, and later the _layout page is rendered. In this case, the user needs to move deferred scripts app level switcher to web.config file or startup.cs file.

At switcher level, once you set EnableDeferredScripts to true in the Action method of the controller, all the startup scripts for the C1 MVC controls in the page are suspended.

HTML Helpers

Web.config

```
<configuration>
<appSettings>
<add key="C1:EnableDeferredScripts" value="true" />
```

Tag Helpers

Startup.cs

```
app.C1().EnableDeferredScripts();
```

Adding Controls

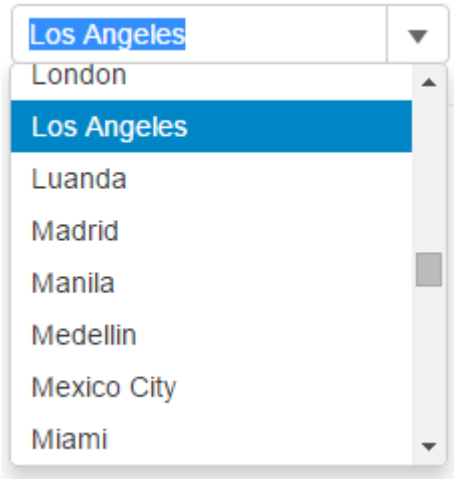
This topic will help you understand how to add an ASP.NET MVC control in your application. You will also learn how to add a model, view and controller in an MVC application. When you use MVC Controls in your web application, it reduces the code complexity, increases flexibility, and helps in reusing the code. For more information about working of MVC controls, see MVC Basics.

Complete the following steps to add an MVC Control.

- **Step 1: Add a Model**
- **Step 2: Add a Controller**
- **Step 3: Add the Control**
- **Step 4: Build and Run the Project**

> **Note:** Make sure that the required references are included in the project. For more information about references, see Installation.

The following image shows how **ComboBox** control appears after completing the steps above:



Step 1: Add a Model

Models are required to fetch data for the controls. A model is to be added only where it is required. This example uses `Cities.cs` model, and steps have been given to explain how to add model in your MVC application. For more information about Models, see MVC Basics.

ASP.NET

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: `Cities.cs`), and then click **Add**.
3. Add the following code to Cities.cs model. We are using Cities class to represent a list of countries.

C#

```
public class Cities
    {
        public static List<string> GetCities()
        {
            return new List<string>
            {
                "Abidjan", "Accra", "Ahmedabad", "Alexandria", "Ankara",
"Atlanta", "Baghdad", "Bandung", "Bangkok", "Barcelona", "Beijing", "Belo
Horizonte",
        "Bengaluru", "Bogota", "Boston", "Buenos Aires", "Cairo", "Calcutta",
"Chengdu", "Chennai", "Chicago", "Chongqung", "Dalian", "Dallas", "Delhi",
        "Detroit", "Dhaka", "Dongguan", "Essen", "Fuzhou", "Guadalajara",
"Guangzhou", "Hangzhou", "Harbin", "Ho Chi Minh City", "Hong Kong", "Houston",
        "Hyderabad", "Istanbul", "Jakarta", "Johannesburg", "Karachi",
"Khartoum", "Kinshasa", "Kuala Lumpur", "Lagos", "Lahore", "Lima", "London",
        "Los Angeles", "Luanda", "Madrid", "Manila", "Medellin", "Mexico City",
"Miami", "Milan", "Monterrey", "Moscow", "Mumbai", "Nagoya", "Nanjing",
        "Naples", "New York", "Osaka", "Paris", "Pheonix", "Philadelphia",
"Porto Alegre", "Pune", "Qingdao", "Quanzhou", "Recife", "Rio de Janeiro",
        "Riyadh", "Rome", "Saint Petersburg", "Salvador", "San Francisco",
"Santiago", "Sao Paulo", "Seoul", "Shanghair", "Shenyang", "Shenzhen",
        "Singapore", "Surabaya", "Surat", "Suzhou", "Sydney", "Taipei",
"Tehran", "Tianjin", "Toronto", "Washington", "Wuhan", "Xi'an-Xianyang",
"Yangoon",
        "Zhengzhou", "Tokyo"
            };
        }
    }
```

A new class is added to the application.

ASP.NET Core

1. In the **Solution Explorer**, right click the folder **Models** and select **Add | New Item**. The **Add New Item** dialog appears.
2. In the **Add New Item** dialog, expand the **Installed** tab towards left and select **Code | Class**.
3. Set the name of the class (for example: `Cities.cs`), and then click **Add**.
4. Add the following code to Cities.cs model. We are using Cities class to represent a list of countries.

C#

```
public class Cities
    {
        public static List<string> GetCities()
```

```
        {
            return new List<string>
            {
                "Abidjan", "Accra", "Ahmedabad", "Alexandria", "Ankara",
"Atlanta", "Baghdad", "Bandung", "Bangkok", "Barcelona", "Beijing", "Belo
Horizonte",
        "Bengaluru", "Bogota", "Boston", "Buenos Aires", "Cairo", "Calcutta",
"Chengdu", "Chennai", "Chicago", "Chongqung", "Dalian", "Dallas", "Delhi",
        "Detroit", "Dhaka", "Dongguan", "Essen", "Fuzhou", "Guadalajara",
"Guangzhou", "Hangzhou", "Harbin", "Ho Chi Minh City", "Hong Kong", "Houston",
        "Hyderabad", "Istanbul", "Jakarta", "Johannesburg", "Karachi",
"Khartoum", "Kinshasa", "Kuala Lumpur", "Lagos", "Lahore", "Lima", "London",
        "Los Angeles", "Luanda", "Madrid", "Manila", "Medellin", "Mexico City",
"Miami", "Milan", "Monterrey", "Moscow", "Mumbai", "Nagoya", "Nanjing",
        "Naples", "New York", "Osaka", "Paris", "Pheonix", "Philadelphia",
"Porto Alegre", "Pune", "Qingdao", "Quanzhou", "Recife", "Rio de Janeiro",
        "Riyadh", "Rome", "Saint Petersburg", "Salvador", "San Francisco",
"Santiago", "Sao Paulo", "Seoul", "Shanghair", "Shenyang", "Shenzhen",
        "Singapore", "Surabaya", "Surat", "Suzhou", "Sydney", "Taipei",
"Tehran", "Tianjin", "Toronto", "Washington", "Wuhan", "Xi'an-Xianyang",
"Yangoon",
        "Zhengzhou", "Tokyo"
            };
        }
    }
```

A new class is added to the application.

Step 2: Add a Controller

Controllers are simple class files. They are responsible for handling incoming requests to the application, retrieve data, and then specify view templates that return a response to the client.

ASP.NET

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, and then click **Add**.
    2. Set name of the controller (For example: `ComboBoxController`).
    3. Click **Add**.
4. Add the following code to replace the Index() method.

C#

```
@using <ApplicationName>.Model

public ActionResult Index()
{
    ViewBag.Cities = Cities.GetCities();
    return View();
}
```

A new controller is added to the application within the folder **Controllers**.

ASP.NET Core

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | New Item.** The **Add New Item** dialog appears.
3. Complete the following steps in the **Add New Item** dialog:
    1. Expand the **Installed** tab towards left, and select **ASP.NET|MVC Controller Class**.
    2. Set name of the controller (For example:`ComboBoxController`).
    3. Click **Add.**
4. Add the following code to replace the Index() method.

C#

```
@using <ApplicationName>.Model

public ActionResult Index()
{
    ViewBag.Cities = Cities.GetCities();
    return View();
}
```

A new controller is added to the application within the folder **Controllers**.

Step 3: Add the Control

View helps the user to view a visual representation of the model. View is most commonly associated with model and retrieves the data required with the help of controllers. We will add a code in the `Index.cshtml` to view **ComboBox** control in the browser.

ASP.NET

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `ComboBoxController`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**.

A view is added for the controller.

ASP.NET Core

1. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
2. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: ComboBox).
3. Right click the folder ComboBox, and select **Add | New Item**. The **Add New Item** dialog appears.
4. Complete the following steps in the **Add New Item** dialog:
    1. Expand the **Installed** tab towards left, and select **ASP.NET|MVC View Page**.
    2. Set name of the view (for example: Index.cshtml).
    3. Click **Add**.

A view is added for the controller.

ASP.NET

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize **ComboBox** control.

```
Razor
@{List<string> cities = ViewBag.Cities;}
<div>
@(Html.C1().ComboBox().Bind(cities).SelectedIndex(0))
</div>
```

ASP.NET Core

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click **Index.cshtml** to open it.
3. Replace the default code of the Index.cshtml file with the code given below to initialize **ComboBox** control.

```
Razor
@{List<string> cities = ViewBag.Cities;}
<div>
<c1-combo-box selected-index=0>
<c1-items-source source-collection=@cities></c1-items-source>
</c1-combo-box>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**ComboBox/index**) in the address bar of the browser to see the view.

Back to Top

Scaffolding in ASP.NET MVC Controls

Scaffolding is a technique in ASP.NET applications, using which you can quickly generate the code, and hence, save the time for making standard web applications. With Scaffolding, you can specify a data model and generate the code that interacts with the data model by setting the properties in the Scaffolder wizard.

- Data Source Configuration
- Using Scaffolding
- Insert MVC Control

The ComponentOne ASP.NET MVC Edition provides scaffolding for **FlexGrid, MultiRow, FlexSheet, FlexChart, FlexPie, Sunburst Chart, FlexRadar and Input** controls.

System Requirements

- Visual Studio 2013 or above
- ASP.NET MVC 4 or above
- Visual C# Template
- Entity Framework Data Model

Note: In case of ASP.NET Core MVC applications, **Visual Studio 2015** and **.NET Framework 4.5** (or above) are required.

Data Source Configuration

To configure the datasource for the application, let us use **C1NWind** database, the **C1NWind.mdf** file, available on your system in *C:\Users\<username>\Documents\ComponentOne Samples\ASP.NET MVC\MVC\CS\MvcExplorer\App_Data*. The steps to generate database for the application are as follows:

1. Add **C1NWind.mdf** file to the App_Data folder in the Solution Explorer.
2. In the Solution Explorer, right-click **Models|Add New Item|Data**, and select **ADO.NET Entity Data Model**.
3. Name the model as **C1NWind**, and click **Add**.
4. In the Entity Data Model Wizard, select **EF Designer from database**, click **Next**. **C1NWind.mdf** database is added to the data connection dropdown.
5. Click **Next** to choose Entity Framework version, and click **Next**.
6. In the **Choose Your Database Objects and Settings**, select Tables, and click **Finish**.

If you can see **C1NWind.edmx** added to your project under the **Models** folder, you have successfully configured the datasource for your application.

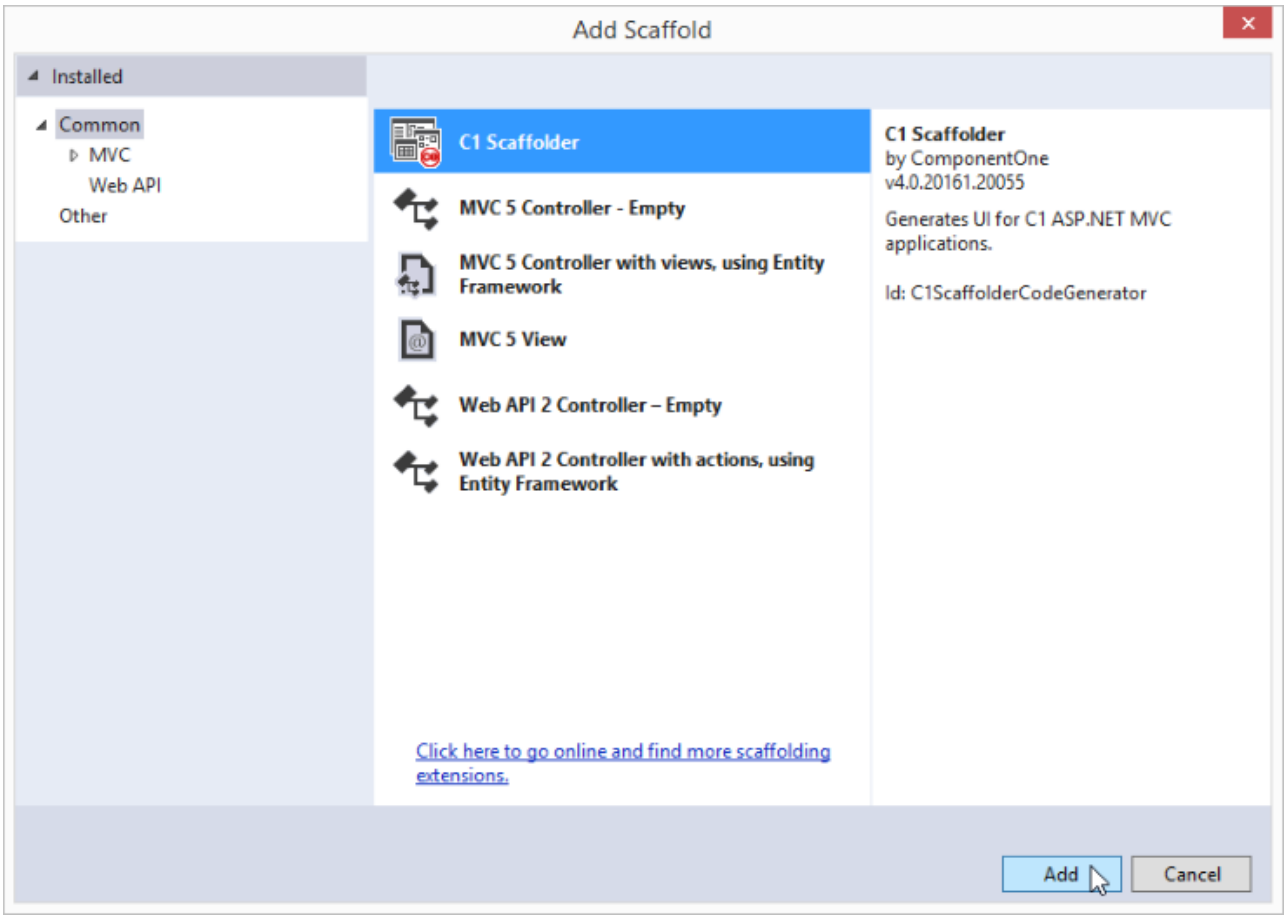
Using Scaffolding

ASP.NET Scaffolding is a code generation framework for ASP.NET MVC applications. You add scaffolding to your project when you want to quickly add code that interacts with data models. Using scaffolding can reduce the amount of time to develop standard data operations in your project.

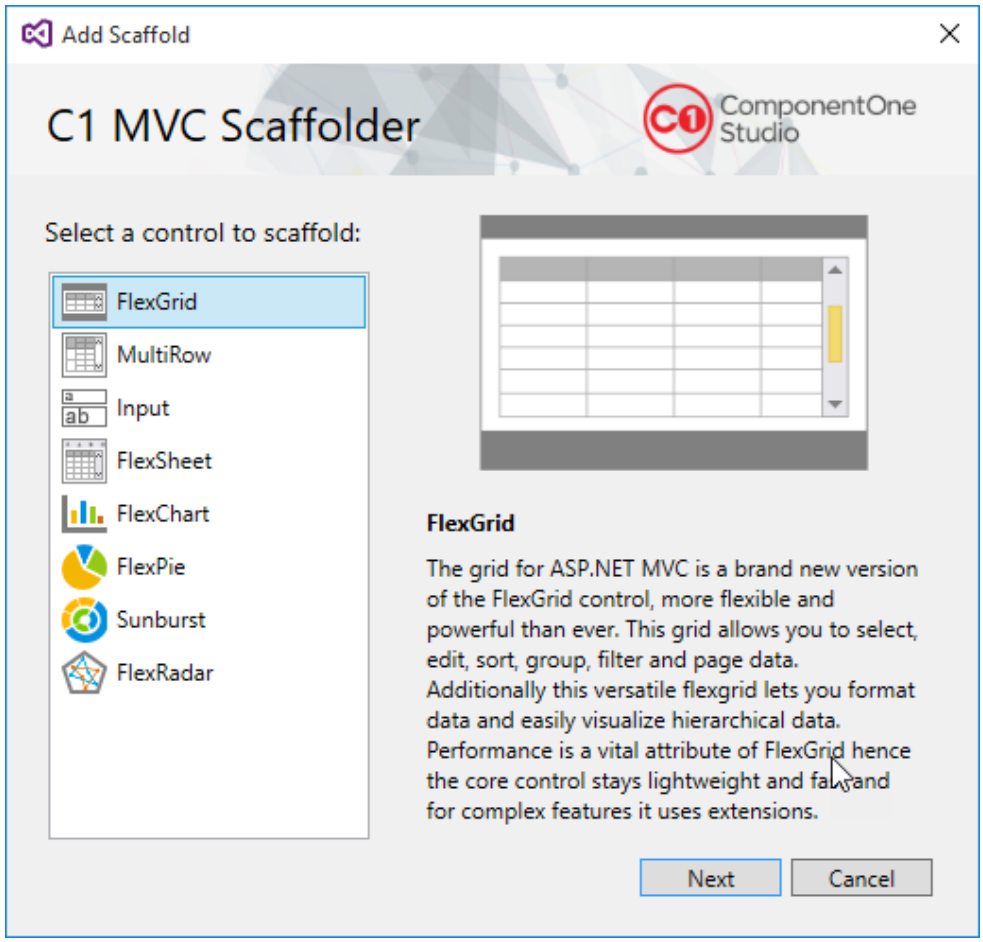
The ComponentOne ASP.NET MVC Edition provides scaffolding for **FlexGrid, MultiRow, FlexSheet, FlexChart, FlexPie, Sunburst Chart, FlexRadar and Input** controls.

The steps to scaffold control for ASP.NET MVC are as follows:

1. Configure the datasource. For more information on configuring a datasource, see Data Source Configuration topic.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane. You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.



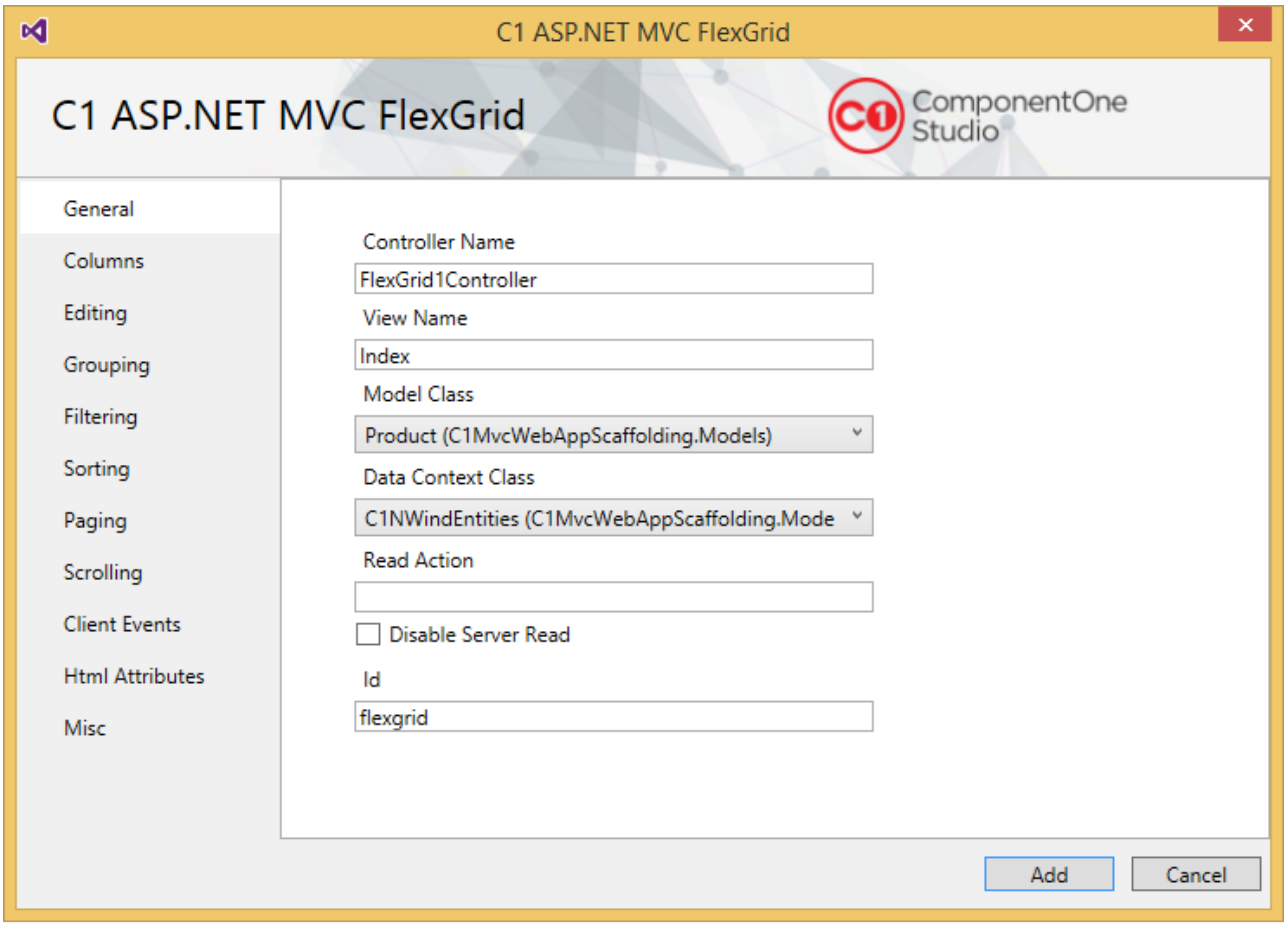
4. Click **Add**.
5. Select the control as per your requirements, and then click **Next**. In our case we have selected **FlexGrid** control.



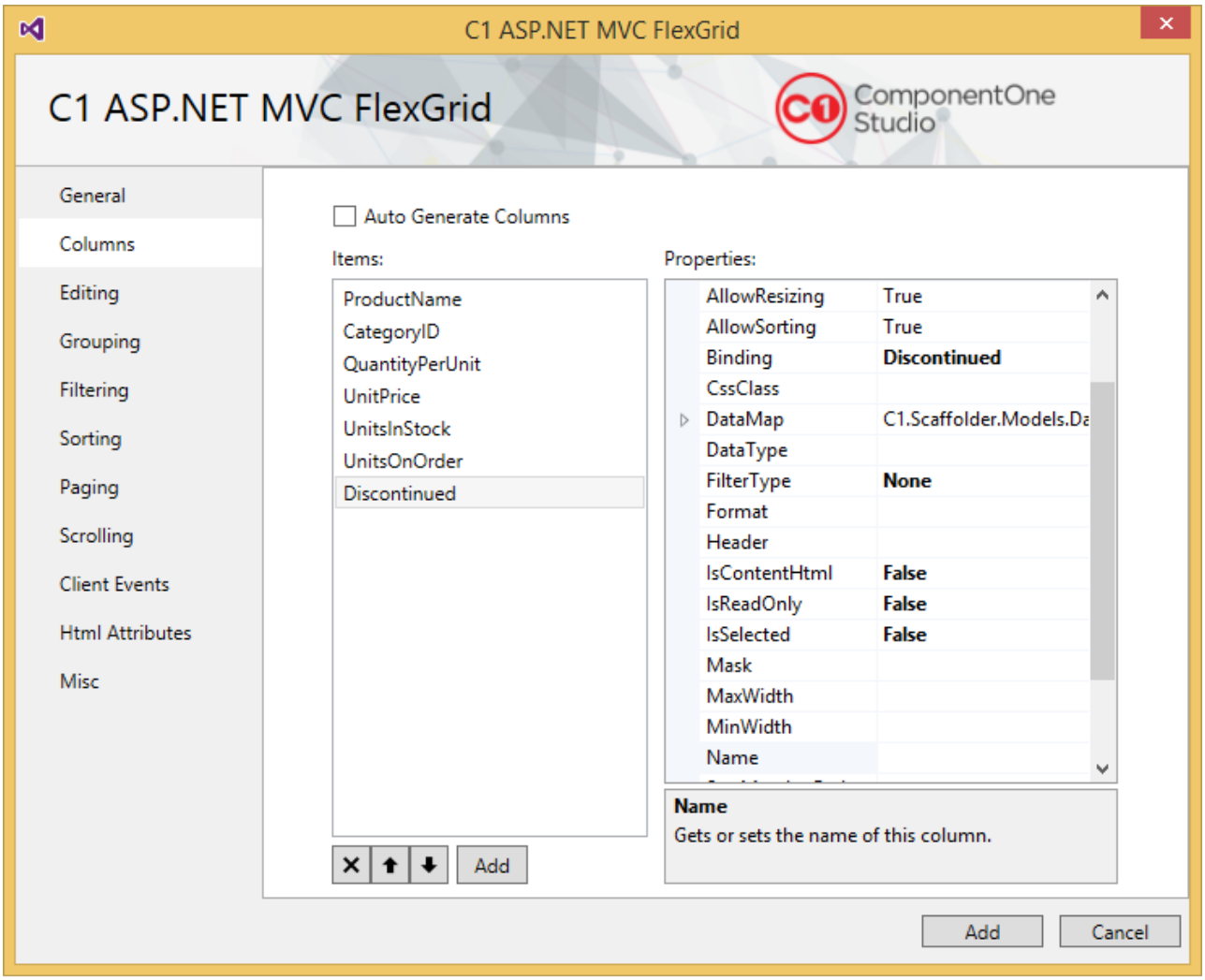
The **C1 ASP.NET MVC FlexGrid** wizard appears with the General tab selected by default.

6. In the **General** tab, specify the model as follows:

    1. Fill in the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in Step 1. In our case, we select Product to populate products in the FlexGrid.
    3. Select **Data Context Class** from the drop-down list. In our case, we select **C1NWindEntities**.



7. Go to **Columns** tab to specify the columns in the FlexGrid control. By default **Auto Generate Columns** is checked; if not, then you can add, delete, or move columns upward or downward in the sequence in which they should appear in the final view. In our case, we have selected columns as shown in the following image:



8. Go to **Editing** tab and check **Allow Edit** and **Allow Delete** check boxes.
9. Go to **Grouping** tab. In the **Group Settings**, check **Show Groups** and CategoryID check boxes from Group Descriptions, and give Group Header Format a name, say 'Group by Category ID'.
10. Go to **Filtering** tab and check **Allow Filtering** check box. Let the other settings be same as default.
11. Go to **Sorting** tab and let the setting be same as default - both **Allow Sorting** and **Show Sort** check boxes should be checked.
12. Go to **Client Events** and check **BeginningEdit** check box.
13. Click **Add**. You will notice that the Controller and View for the selected model is added to your project. The codes generated for the Controller and View are as follows:

FlexGrid1Controller.cs

FlexGrid1Controller.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using C1.Web.Mvc;
using C1.Web.Mvc.Serialization;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Validation;
```

```
using System.Web;
using System.Web.Mvc;

// This code was generated by C1 Scaffolder.

namespace C1MvcWebAppScaffolding.Controllers
{
    public partial class FlexGridController : Controller
    {
        private C1MvcWebAppScaffolding.Models.C1NWindEntities db = new
C1MvcWebAppScaffolding.Models.C1NWindEntities();
        public ActionResult Index()
        {
            var model = db.Products;
            return View(model);
        }

        public ActionResult FlexGrid_Update([C1JsonRequest]CollectionViewEditRequest
requestData)
        {
            return Update(requestData, db.Products);
        }

        private ActionResult Update(CollectionViewEditRequest requestData, DbSet
data) where T : class
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                string error = string.Empty;
                bool success = true;
                try
                {
                    db.Entry(item as object).State = EntityState.Modified;
                    db.SaveChanges();
                }
                catch (DbEntityValidationException e)
                {
                    error = string.Join(",", e.EntityValidationErrors.Select(result
=>
                    {
                        return string.Join(",", result.ValidationErrors.Select(err
=> err.ErrorMessage));
                    }));
                    success = false;
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }
                return new CollectionViewItemResult
                {
                    Error = error,
                    Success = success && ModelState.IsValid,
                    Data = item
```

```
                };
            }, () => data.ToList()));
        }

        public ActionResult FlexGrid_Delete([C1JsonRequest]CollectionViewEditRequest
requestData)
        {
            return Delete(requestData, db.Products, item => new object[] {
item.ProductID });
        }

        private ActionResult Delete(CollectionViewEditRequest requestData, DbSet
data, Func getKeys) where T : class
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                string error = string.Empty;
                bool success = true;
                try
                {
                    var resultItem = data.Find(getKeys(item));
                    data.Remove(resultItem);
                    db.SaveChanges();
                }
                catch (DbEntityValidationException e)
                {
                    error = string.Join(",", e.EntityValidationErrors.Select(result
=>
                    {
                        return string.Join(",", result.ValidationErrors.Select(err
=> err.ErrorMessage));
                    }));
                    success = false;
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }
                return new CollectionViewItemResult
                {
                    Error = error,
                    Success = success && ModelState.IsValid,
                    Data = item
                };
            }, () => data.ToList()));
        }

    }
}
```

Index

HTML Helpers

Razor (Index.cshtml)

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@using C1.Web.Mvc.Grid
@model IEnumerable<C1MvcWebAppScaffolding.Models.Product>
    <script type="text/javascript">
        function beginningEdit(sender, e) {
            // Implement the event handler for beginningEdit.
        }
    </script>
@(Html.C1().FlexGrid<C1MvcWebAppScaffolding.Models.Product>()
    .Id("flexgrid")
    .Bind(bl => { bl.Bind(Model)
        .Update(Url.Action("FlexGrid_Update"))
        .Delete(Url.Action("FlexGrid_Delete"))
        .GroupBy("CategoryID")
    ;})
    .AutoGenerateColumns(false)
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ProductName"));
        bl.Add(cb => cb.Binding("CategoryID"));
        bl.Add(cb => cb.Binding("QuantityPerUnit"));
        bl.Add(cb => cb.Binding("UnitPrice"));
        bl.Add(cb => cb.Binding("UnitsInStock"));
        bl.Add(cb => cb.Binding("UnitsOnOrder"));
        bl.Add(cb => cb.Binding("Discontinued"));
    })
    .AllowDelete(true)
    .GroupHeaderFormat("Group by Category ID")
    .Filterable(f => { f
        .DefaultFilterType(FilterType.Both)
    ;})
    .OnClientBeginningEdit("beginningEdit")
)
```

Tag Helpers

HTML

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Grid
@addTagHelper *, C1.AspNetCore.Mvc
@model IEnumerable<C1MvcApplication1.Models.Product>
    <script type="text/javascript">
        function beginningEdit(sender, e) {
            // Implement the event handler for beginningEdit.
        }
    </script>
<c1-flex-grid id="flexgrid" auto-generate-columns="false" allow-delete="true" group-
header-format="Group by Category ID"
```

```
            beginning-edit="beginningEdit">
    <c1-items-source source-collection="Model" update-action-
url="@(Url.Action("FlexGrid_Update"))"
                     delete-action-url="@(Url.Action("FlexGrid_Delete"))"
group-by="CategoryID"></c1-items-source>
    <c1-flex-grid-column binding="ProductName"></c1-flex-grid-column>
    <c1-flex-grid-column binding="CategoryID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="QuantityPerUnit"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitPrice"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitsInStock"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitsOnOrder"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discontinued"></c1-flex-grid-column>
    <c1-flex-grid-filter default-filter-type="FilterType.Both"></c1-flex-grid-filter>
</c1-flex-grid>
```

14. Run the project.

	ProductNa...	CategoryID	QuantityP...	UnitPrice	UnitsInSto...	UnitsOnOr...	Discontin...
	Group by Category ID						
	Chai	1	10 boxes x 20...	18	39	0	☑
	Guaraná Fa...	1	12 - 355 ml c...	4.50	20	0	☑
	Sasquatch Ale	1	24 - 12 oz bot...	14	111	0	☐
	Steeleye Stout	1	24 - 12 oz bot...	18	20	0	☐
	Côte de Blaye	1	12 - 75 cl bottles	263.50	17	0	☐
	Chartreuse ...	1	750 cc per bo...	18	69	0	☐
	Ipoh Coffee	1	16 - 500 g tins	46	17	10	☐
	Laughing Lu...	1	24 - 12 oz bot...	14	52	0	☐
	Outback Lager	1	24 - 355 ml b...	15	15	10	☐
	Rhönbräu Kl...	1	24 - 0.5 l bottles	7.75	125	0	☐
	Group by Category ID						
	Uncle Bob's ...	7	12 - 1 lb pkgs.	30	15	0	☑
	Tofu	7	40 - 100 g pkgs.	23.25	35	0	☐
	Rössle Sau...	7	25 - 825 g cans	45.60	26	0	☑
	Manjimup D...	7	50 - 300 g pkgs.	53	20	0	☐
	Longlife Tofu	7	5 kg pkg.	10	4	20	☐
	Group by Category ID						
	Mishi Kobe ...	6	18 - 500 g pkgs.	97	29	0	☑
	Alice Mutton	6	20 - 1 kg tins	39	0	0	☑
	Thüringer R...	6	50 bags x 30 ...	123.79	0	0	☑
	Perth Pasties	6	48 pieces	32.80	0	0	☑

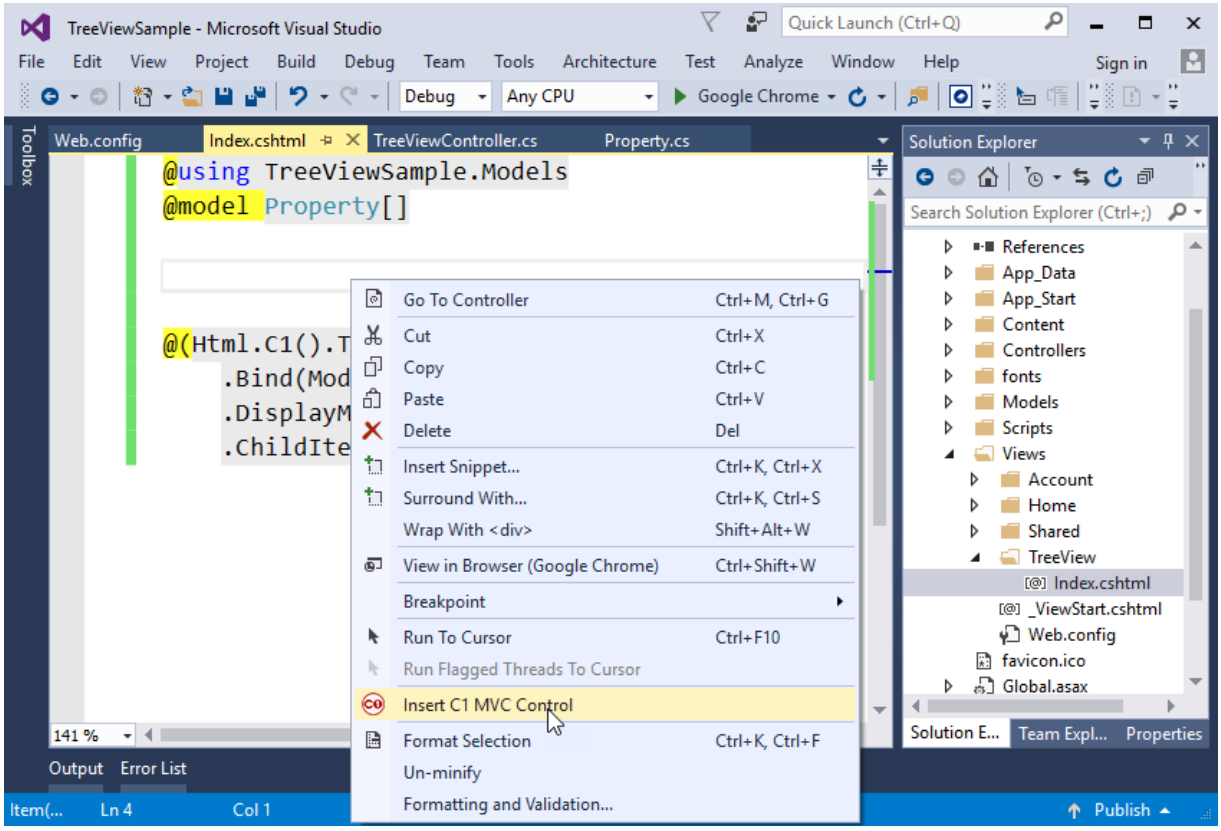
Insert MVC Control

The **Insert C1 MVC Control** option allows you to easily add a fully functional control in the view code of an existing application. The dialog can be invoked by right-clicking the design area of a view. Note that a view must be focused in order to invoke the **AddScaffold** dialog.

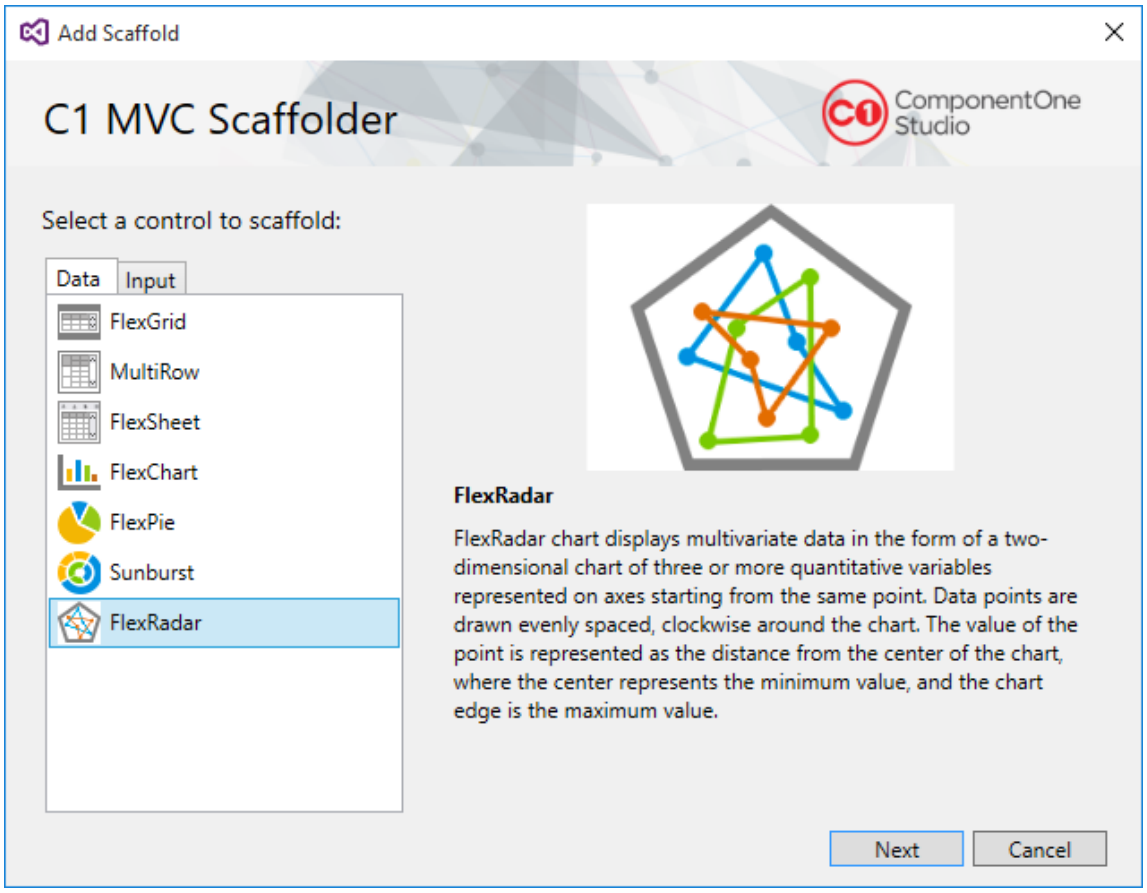
To insert C1 MVC Control in View code

Complete the following steps to add a new control in the view code using Scaffolding.

1. In an existing MVC application, open the **Index.cshtml** view file.
2. In the Index.cshtml file, right-click anywhere in the view code to open the context menu, and then select **Insert C1 MVC Control** option. The **Add Scaffold** dialog appears.

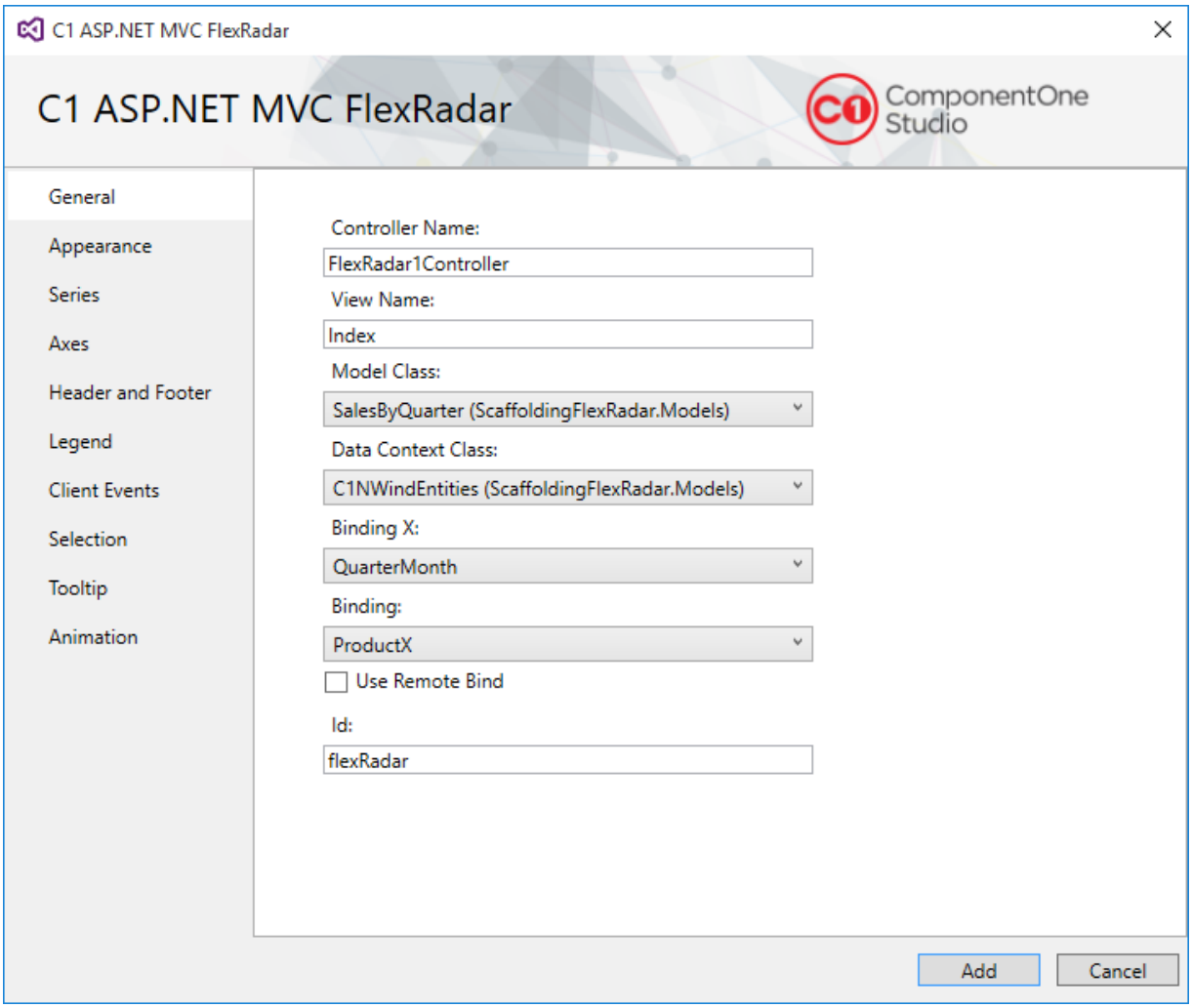


3. In the **Add Scaffold** dialog, select a control you want to add in your application, and then click **Next**. In our case, we have selected **FlexRadar** from the **Data** tab.



4. In the General tab, specify the model details as follows:

    1. Select **Model Class** from the drop-down list. The list shows all the available model types in the application. In our case, we select **SalesByQuarter** to populate data in FlexRadar.
    2. Select **Data Context Class** from the drop-down list. In our case, we select **C1NWindEntities**. For more information, see Data Source Configuration.
    3. In the **BindingX** drop-down, select a column to bind it with your FlexRadar. In our case, we have selected **QuarterMonth**.
    4. In the **Binding** drop-down, select a column to bind it with your FlexRadar. In our case, we have selected **ProductX**.



5. Click **Add**. It automatically generates the **View** code for the control and inserts it into the View at the cursor position. Once the code is generated, you can run the project using the **F5** button. Similarly, we can also add a new control in the view code of **ASP.NET Core** application using the same steps.

Controller

Razor

```
public class TreeViewController : Controller
    {
        TreeViewSample.Models.C1NWindEntities db = new C1NWindEntities();
        // GET: TreeView
        public ActionResult Index()
        {
            ViewBag.SalesByQuarters = db.SalesByQuarters;
            return View(Property.GetData());
        }
    }
```

View

HTML

```
@using TreeViewSample.Models
@model Property[]
```

```
@(Html.C1().FlexRadar<SalesByQuarter>()
    .BindingX("QuarterMonth")
    .Series(csbsb =>
    {
        csbsb.Add();
    })
    .Binding("ProductX")
    .Bind(cvsb => cvsb.Bind((IEnumerable<SalesByQuarter>)ViewBag.SalesByQuarters))
    .Height("300px")
    .Id("flexRadar4"))

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
```

Handling Client-Side Methods and Events

MVC Controls provide an advanced client-side API, which include methods, events and properties that allow you to update or modify your control at client-side. This enables the user to develop coherent web applications by combining server-side and client-side processing. The events, methods and properties works similar to the way **.NET Framework** handles events, methods and properties. The client-side API provided with the product is implemented using JavaScript. JavaScript code that implements the client API is embedded in the view file or embedded as a resource file.

- **Client-Side Methods**
- **Client-Side Events**

Client-Side Methods

Client-side API provides a variety of client-side methods to control the behavior and appearance of a control. To use a client-side API method in your application, you need to add a JavaScript code after the control declaration. In the JavaScript code, you can define or set methods, events and properties of any specific control.

Scenario

We want to display the sales data (Country, Product, and Amount) using the FlexGrid control. However, we also want to display the value of corresponding columns (Amount 2, Active, and Discount) in labels, based on the current row selected in the FlexGrid. This scenario can be implemented using the client-side methods. For more information about client-side methods, see Client-Side API Reference.

Before we even begin implementing the scenario, we need to first identify a suitable method in the FlexGrid client-side API section. In the FlexGrid class, we find the OnSelectionChanged method that helps us to process the selection changes. Once we select the OnSelectionChanged method, it further provides us details on the argument type, which in this case is CellRangeEventArgs. The CellRangeEventArgs provide row, column, data, range and panel arguments for the method. Now that we have selected the correct method for our scenario, we can now proceed to the implementation in the FlexGrid control.

In the code example below, we are using the **OnSelectionChanged()** method in the razor code that raises the **selectionChanged** event. We create a JavaScript function **gridSelectionChange(s, e)** to fetch the details of each row using CollectionView class. In the gridSelectionChange function, **s** stands for the sender(FlexGrid), and **e** is type of **CellRangeEventArgs** class. Moving further, we declare reference variables for the three label items, Amount2, Active and Discount respectively. CollectionView class provides currentItem property that returns the current item that is currently selected in the grid. We finally use this property to fetch the value of the three columns and assign them to the labels.

The following code example fetches the details of each row and display it in the labels using the

OnClientSelectionChanged() method at client-side. This example uses the **Sale.cs** model created in the FlexGrid: Quick Start topic.

HTML Helpers

Razor

```
@using FlexGridClientSide.Models
@using C1.Web.Mvc.Grid

@model IEnumerable<Sale>
<script>
    function gridSelectionChange(s, e)
        {
            var cv =s.collectionView;

            var amt2 = document.getElementById('lblAmt2');
            var active = document.getElementById('lblActive');
            var discount = document.getElementById('lblDisc');
            if (cv.currentItem)
            {
                amt2.innerText = cv.currentItem.Amount2;
                active.innerText = cv.currentItem.Active;
                discount.innerText = cv.currentItem.Discount;
            }
    }
</script>
// Instantiate FlexGrid and set its properties
@(Html.C1().FlexGrid().Id("fg").Bind(Model).AutoGenerateColumns(false).Height("440px")
        .OnClientSelectionChanged("gridSelectionChange")
        .Columns(c => c.Add(cb => cb.Header("Country").Binding("Country"))
        .Add(cb => cb.Binding("Product").Header("Product"))
        .Add(cb => cb.Binding("Amount").Header("Amount")))
)
<div class="col-sm-4">
    <label>Amount2:</label>
    <label id="lblAmt2"></label>
    <br />
    <label>Active:</label>
    <label id="lblActive"></label>
    <br />
    <label>Discount:</label>
    <label id="lblDisc"></label>
</div>
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>
```

```
<script>
    function gridSelectionChange(s, e)
        {
            var cv =s.collectionView;

            var amt2 = document.getElementById('lblAmt2');
            var active = document.getElementById('lblActive');
            var discount = document.getElementById('lblDisc');
            if (cv.currentItem)
            {
                amt2.innerText = cv.currentItem.Amount2;
                active.innerText = cv.currentItem.Active;
                discount.innerText = cv.currentItem.Discount;
            }
    }
</script>
<c1-flex-grid auto-generate-columns="false" class="grid" id="fg" height="440px"
              selection-changed="gridSelectionChange">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-grid-column binding="Country" header="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" header="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c" header="Amount"></c1-flex-grid-
column>
</c1-flex-grid>

<div class="col-sm-4">
    <label>Amount2:</label>
    <label id="lblAmt2"></label>
    <br />
    <label>Active:</label>
    <label id="lblActive"></label>
    <br />
    <label>Discount:</label>
    <label id="lblDisc"></label>
</div>
```

Client-Side Events

Client-side event handlers and callbacks are implemented using a JavaScript code. To handle a client-side event, the control provides a property that accepts either event handling code directly or the name of a handling function. The function must be initialized in the view code or in a separate js file.

Scenario

We want to display the sales data (Country, Product, and Amount) and show the amount column data in red color, if the **Amount** value is less than 1,000 dollars. This scenario can be implemented using the client-side events. For more information about client-side events, see Client-Side API Reference.

Before we even begin implementing the scenario, we need to first identify a suitable event in the FlexGrid client-side API section. For the following scenario, we will use the formatItem event in the FlexGrid class, as the requirement in our case is to format the cells in a grid. Once we select the formatItem event, it will provide us details on the argument type, which in this case is FormatItemEventArgs. The FormatItemEventArgs provide row, column, data, range and panel

arguments for the event. Now that we have selected the correct event for our scenario, we can now proceed to the implementation in the FlexGrid control.

In the code example below, we subscribe the **OnClientFormatItem** event by assigning the JavaScript function name in the razor code. In the <Script> section, we declare the **formatItem** function to check, if the cell to be formatted is of type cell and column to be formatted is **Amount**. Moving further, we get the cell data using the getCellData method, and then check the value of Amount column to apply the color using cell.style.color property. Once you execute the code, notice that the cells containing amount less than 1,000 dollars are displayed in red color.

The following code example shows how to use OnClientFormatItem event in the FlexGrid control. This example uses the **Sale.cs** model created in the FlexGrid: Quick Start topic.

HTML Helpers

Razor

```
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>
<script>
    function formatItem(s, e) {
        if (e.panel.cellType == wijmo.grid.CellType.Cell &&
e.panel.columns[e.col].binding == "Amount") {
            var val = e.panel.grid.getCellData(e.row, e.col, false);
            if (val < 1000) {
                e.cell.style.color = 'red';
            }
        }
    }
</script>
// Instantiate FlexGrid and set its properties
@(Html.C1().FlexGrid().Id("fg").Bind(Model).AutoGenerateColumns(false).Height("440px")
        .OnClientFormatItem("formatItem")
        .Columns(c => c.Add(cb => cb.Header("Country").Binding("Country"))
        .Add(cb => cb.Binding("Product").Header("Product"))
        .Add(cb => cb.Binding("Amount").Header("Amount")))
)
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

<script>
    function formatItem(s, e) {
        if (e.panel.cellType == wijmo.grid.CellType.Cell &&
e.panel.columns[e.col].binding == "Amount") {
            var val = e.panel.grid.getCellData(e.row, e.col, false);
            if (val < 1000) {
                e.cell.style.color = 'red';
            }
        }
```

```
    }
</script>
<c1-flex-grid auto-generate-columns="false" class="grid" id="fg" height="440px"
              format-item="formatItem">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-grid-column binding="Country" header="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" header="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c" header="Amount"></c1-flex-grid-
column>
</c1-flex-grid>
```

Event handling using addHandler()

Another way of handling events is by using the **addHandler** method, just like the .NET events. The events can be subscribed using addHandler method and unsubscribed using the removeHandler method. The addHandler statement allows you to specify an event handler. However, addHandler when used with removeHandler, provides greater flexibility, allowing you to dynamically add, remove, and change the event handler associated with an event.

In the code example below, we are using the **addHandler** method to define a function for the **formatItem** event.

HTML Helpers

Razor

```
@model IEnumerable<Sale>
<script>
    var fg;
    c1.documentReady(function () {
        fg = wijmo.Control.getControl("#fg");
        fg.formatItem.addHandler(function (s,e) {
            if (e.panel.cellType == wijmo.grid.CellType.Cell &&
e.panel.columns[e.col].binding == "Amount") {
                var val = e.panel.grid.getCellData(e.row, e.col, false);
                if (val < 1000)
                    e.cell.style.color = 'red';
            }
        })
    });
</script>
// Instantiate FlexGrid and set its properties
@(Html.C1().FlexGrid().Id("fg").Bind(Model).AutoGenerateColumns(false).Height("440px")
        .Columns(c => c.Add(cb => cb.Header("Country").Binding("Country"))
        .Add(cb => cb.Binding("Product").Header("Product"))
        .Add(cb => cb.Binding("Amount").Header("Amount"))))
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>
```

```
<script>
    var fg;
    c1.documentReady(function () {
        fg = wijmo.Control.getControl("#fg");
        fg.formatItem.addHandler(function (s,e) {
        if (e.panel.cellType == wijmo.grid.CellType.Cell &&
e.panel.columns[e.col].binding == "Amount")            {
                var val = e.panel.grid.getCellData(e.row, e.col, false);
                if (val < 1000)
                    e.cell.style.color = 'red';
            }
        })
    });
</script>
<c1-flex-grid auto-generate-columns="false" class="grid" id="fg">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-grid-column binding="Country" header="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" header="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c" header="Amount"></c1-flex-grid-
column>
</c1-flex-grid>
```

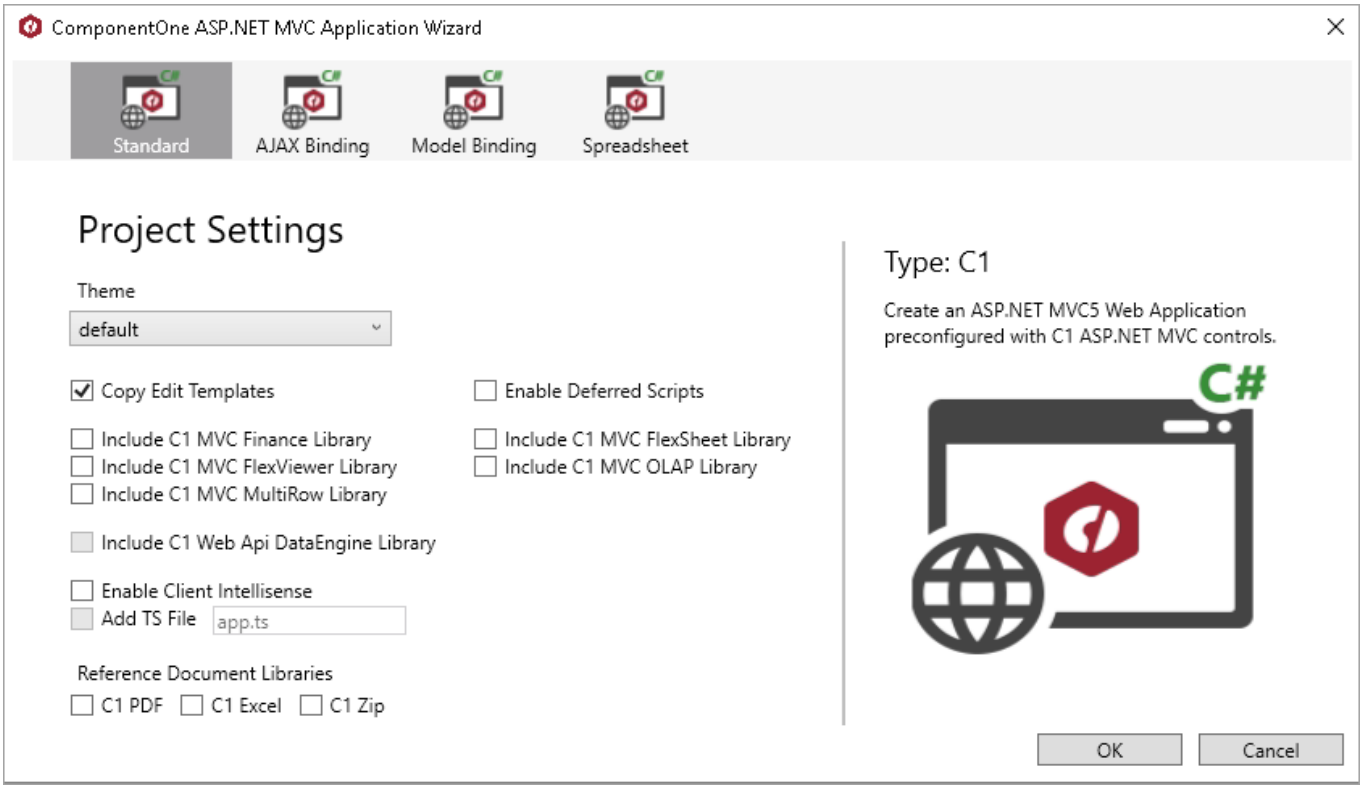
Client Side IntelliSense Support

IntelliSense helps the developer to speed up the process of coding application by reducing typos and other common mistakes. While you are working with the client script in the typescript editor, IntelliSense helps you by displaying the objects, functions, and properties based on your current context. You can select an option from the pop-up list provided by the MVC IntelliSense to complete the code.

- **TypeScript IntelliSense**
- **JavaScript IntelliSense**
- **Using Cast Method**
- **JavaScript IntelliSense in VisualStudio 2017**

TypeScript IntelliSense

Check the **Enable Client IntelliSense** check box from the **ComponentOne ASP.NET MVC Application Wizard** to use IntelliSense for MVC controls in your Visual Studio application.

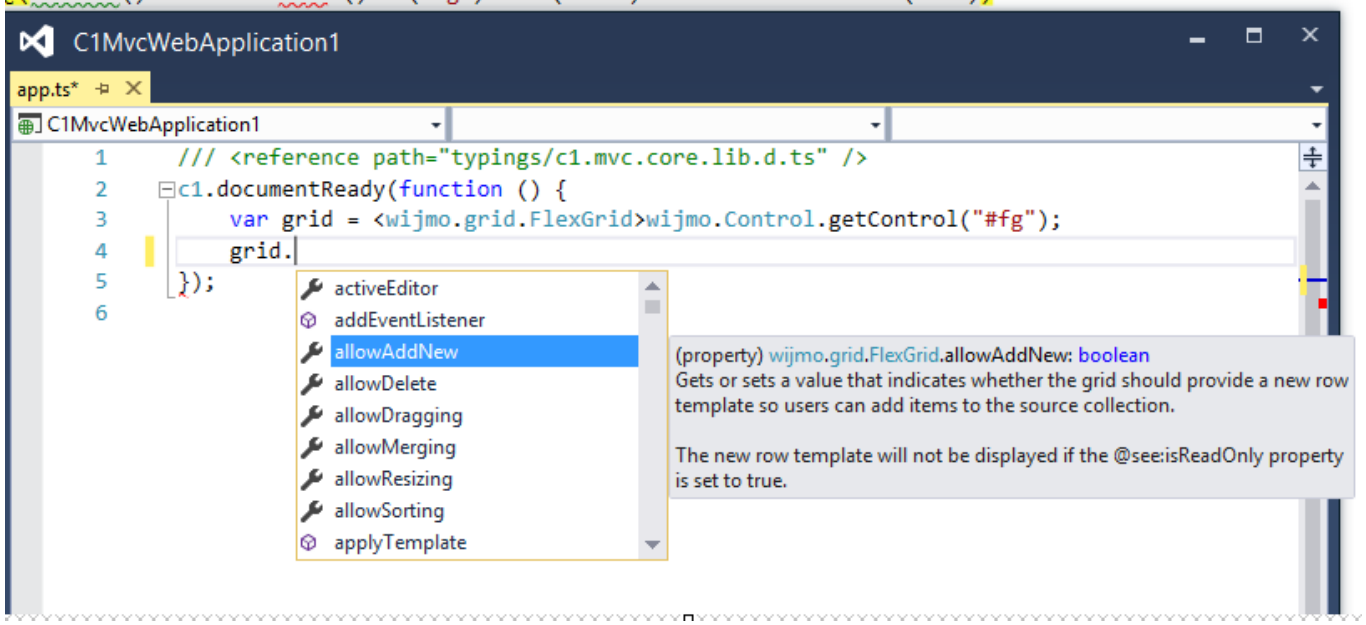


Once you check the **Enable Client IntelliSense** option, it automatically adds **c1.mvc.basic.lib.d.ts** to **Scripts\Typings** folder. Similarly, **c1.mvc.finance.lib.d.ts, c1.mvc.flexsheet.lib.d.ts, c1.mvc.flexviewer.lib.d.ts, c1.mvc.multirow.lib.d.ts** or **c1.mvc.olap.lib.d.ts** files are also automatically added to the folder, in case the user is working on Financial chart, FlexSheet, FlexViewer, MultiRow or OLAP, and have checked the **Include C1 MVC Finance Library** or **Include C1 MVC FlexSheet Library** or **Include C1 MVC FlexViewer Library** or **Include C1 MVC MultiRow Library** or **Include C1 MVC Olap Library** check box in the **ComponentOne ASP.NET MVC Application Wizard**.

Note: In case you are using standard VisualStudio templates to create an application, you can enable TS IntelliSense in your application by including and referencing the type definition files from **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\TsTypings** location.

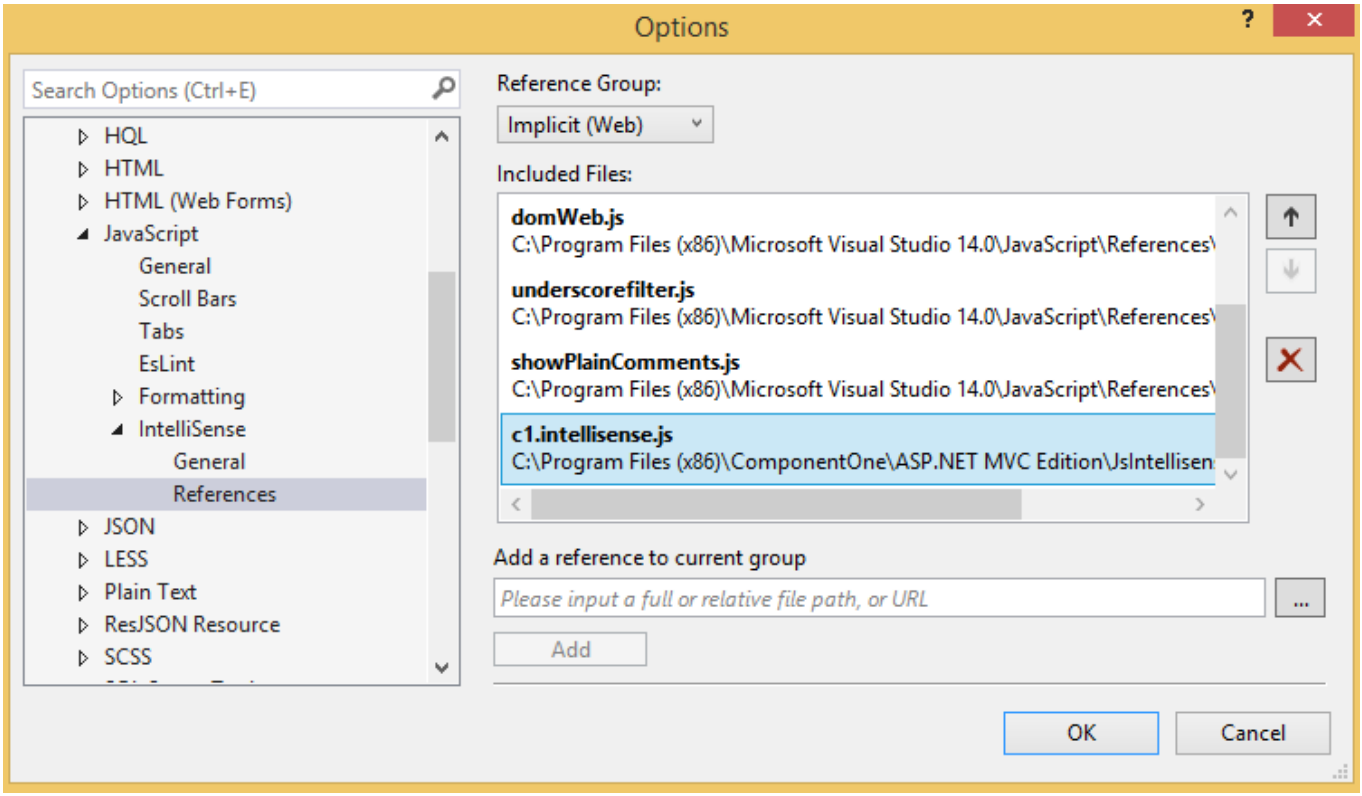
The **Add TS File** option helps you to use C1 MVC IntelliSense while working with TypeScript files. When you select the Add TS File option, it adds a **.ts** file to the Scripts folder that refers to the related **.d.ts** file.

```
@using C1MvcWebApplication1.Models
@model IEnumerable<Sale>
<script src="~/Scripts/app.js"></script>
@{
    ViewBag.Title = "Home Page";
}

@(Html.C1().FlexGrid<Sale>().Id("fg").Bind(Model).AutoGenerateColumns(true))
```

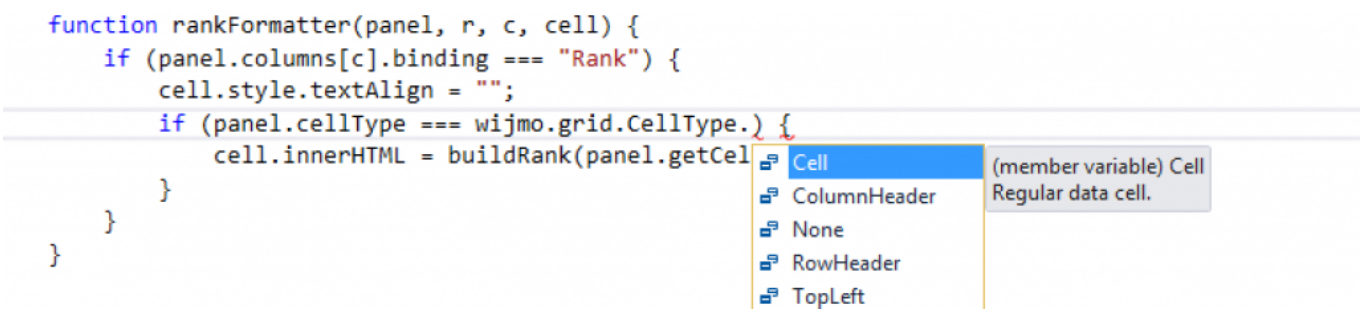


JavaScript IntelliSense

C1Studio MVC Edition introduces JavaScript IntelliSense which enables the developers to quickly write their code for development. When you install C1 MVC Edition using the C1Studio installer, the JS IntelliSense file for MVC is automatically added to Visual Studio.



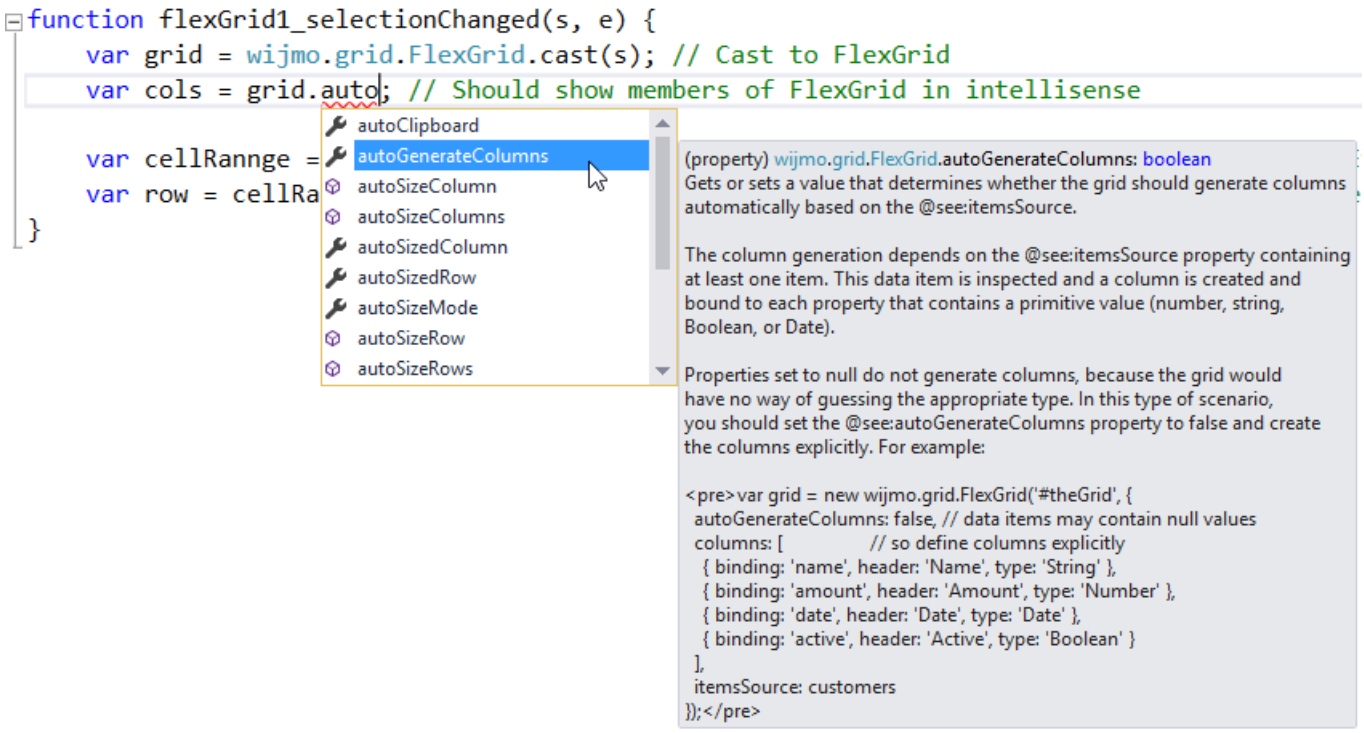
Once the JavaScript IntelliSense support is installed in your Visual Studio, it enables client side IntelliSense whenever you write script code.



Using Cast Method

The **Cast** method is a part of the JavaScript IntelliSense support for C1 MVC ASP.NET controls. In a TypeScript or JavaScript file, you can cast an object to other type, and then the members of the newly created type is available in the IntelliSense. Using the **Cast** method is helpful when you intend to use IntelliSense, while writing JavaScript code using the client-side API.

The example of using Cast method is as follows.

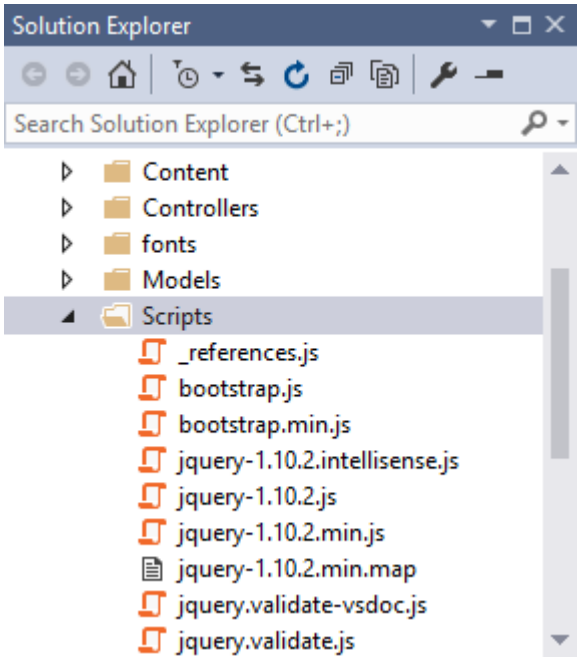


JavaScript IntelliSense in VisualStudio 2017

To enable JavaScript IntelliSense in VisualStudio 2017, you need to add **c1.mvc.basic.lib.d.ts** TypeScript file in your MVC application.

ASP.NET

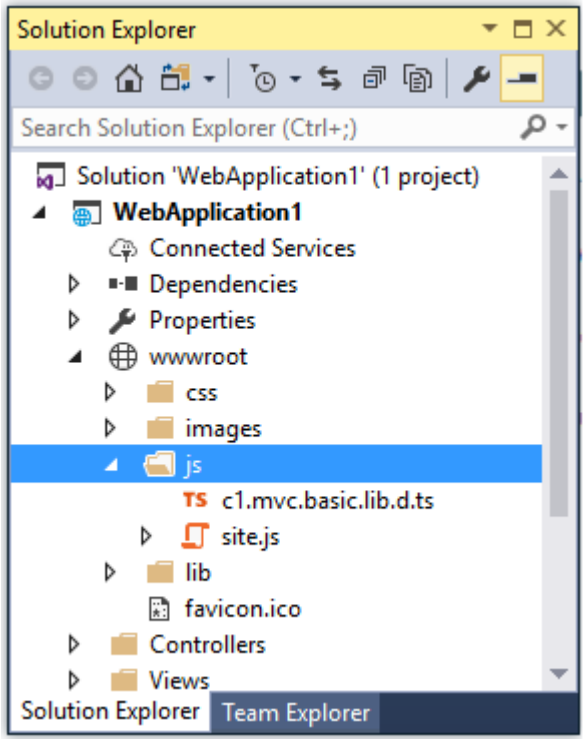
1. From the **Solution Explorer**, right-click the **Scripts** folder, and then select **Add | Existing Item**.
2. In the **Add Existing Item** dialog, select **c1.mvc.basic.lib.d.ts** from **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\TsTypings** location.
3. To use JavaScript IntelliSenese in a JavaScript file or TypeScript file, add **/// <reference path="c1.mvc.basic.lib.d.ts" />** reference in **Scripts_references.js** file or any other js file to use IntelliSense.



Similarly you can add **c1.mvc.finance.lib.d.ts, c1.mvc.flexsheet.lib.d.ts, c1.mvc.flexviewer.lib.d.ts, c1.mvc.multirow.lib.d.ts** or **c1.mvc.olap.lib.d.ts** file to use control specific IntelliSense in your application.

ASP.NET Core

1. From the **Solution Explorer**, expand the **wwwroot** section.
2. Right-click the **js** folder, and then select **Add | Existing Item**.
3. In the **Add Existing Item** dialog, select **c1.mvc.basic.lib.d.ts** from **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\TsTypings** location.
4. To use JavaScript IntelliSenese in a JavaScript file or TypeScript file, add **/// <reference path="c1.mvc.basic.lib.d.ts" />** reference in **wwwroot\js\site.js** file or any other js file to use IntelliSense.



Similarly you can add **c1.mvc.finance.lib.d.ts, c1.mvc.flexsheet.lib.d.ts, c1.mvc.flexviewer.lib.d.ts, c1.mvc.multirow.lib.d.ts** or **c1.mvc.olap.lib.d.ts** file to use control specific IntelliSense in your application.

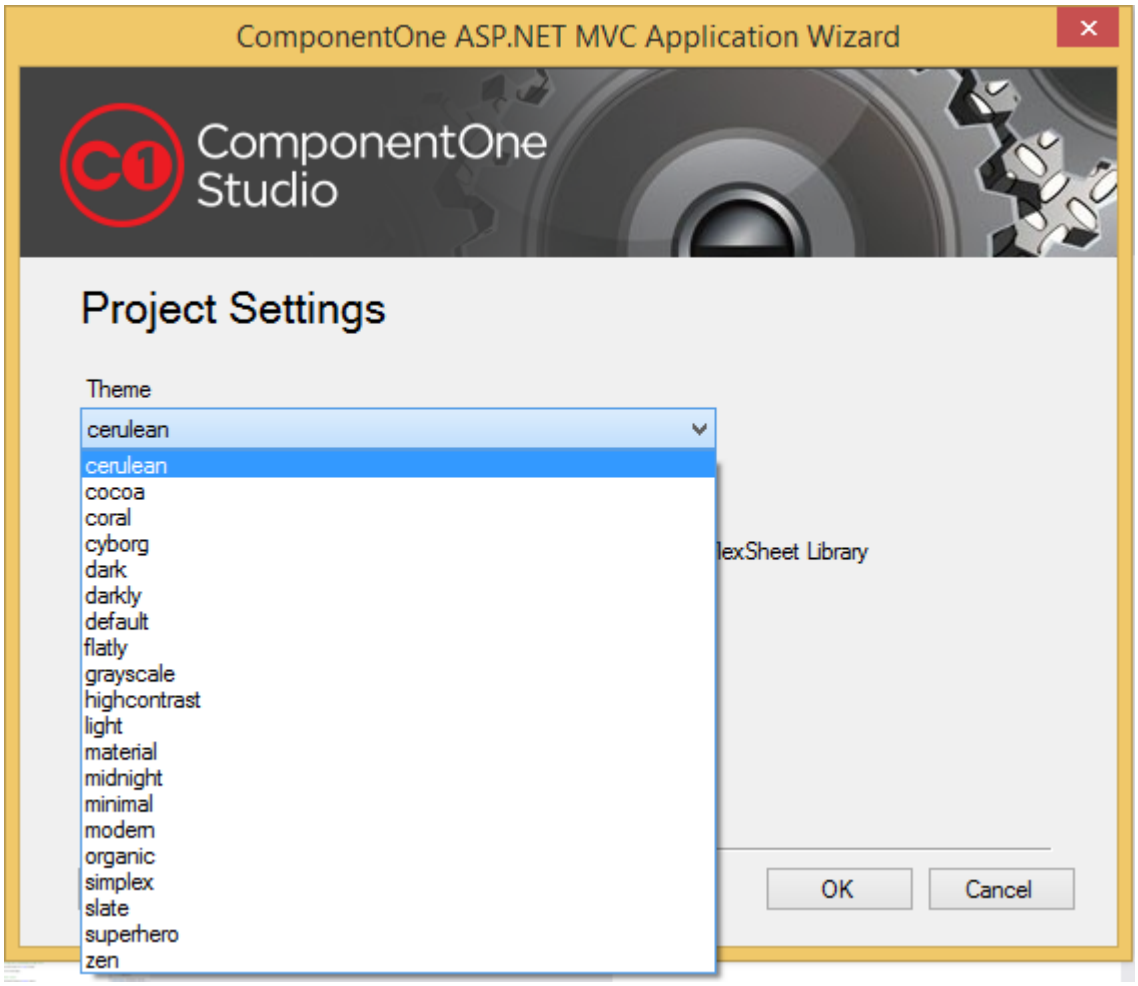
Themes

A theme is a collection of GUI details including colors, fonts, images and skins that you can apply to customize the appearance of your control. MVC Edition supports an array of built-in themes that you can either directly select at the time of creating an MVC application, or set at the time of registering resources. In addition, you can also set the Material themes while registering resources for your MVC application.

By default, when you create an MVC application, it uses a common default theme. The appearance is implemented using a specific C1 theme called "**default**". ASP.NET MVC controls are provided with many different predefined visual themes, that allows the user to provide a consistent appearance to all the MVC controls within your website pages. These predefined themes are defined in C1.Web.Mvc.Themes static class.

There are two ways by which you can apply themes to your MVC control.

1. Using ComponentOne Template - You can use the **ComponentOne ASP.NET MVC Application Wizard** to set a theme to your application using the **Theme** option. In the image below, you can see the list of themes that are available for ASP.NET MVC application.



2. Using Visual Studio Template - You can set the above mentioned themes and material themes manually while you are registering resources for your MVC application. To apply a theme in your MVC application, you need to add the following code in **_Layout.cshtml** file.

HTML Helpers

_Layout.cshtml

```
//Set C1 Theme
@Html.C1().Styles().Theme("cocoa")

//Set using Theme Enum
@Html.C1().Styles().Theme(C1.Web.Mvc.Themes.Cocoa)

//Set Material Theme
@Html.C1().Styles().Theme("material.deep_orange-red")
```

Tag Helpers

_Layout.cshtml

```
//Set C1 Theme
<c1-styles theme="cocoa" />

// Set using Theme Enum
<c1-styles theme="@Themes.Cocoa" />

//Set Material Theme
<c1-styles theme="material.deep_orange-red" />
```

Redistributable Files

ASP.NET MVC 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, free of royalties, the following redistributable files 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 ASP.NET MVC

- C1.Web.Mvc.dll
- C1.Web.Mvc.Finance.dll
- C1.Web.Mvc.FlexSheet.dll
- C1.Web.Mvc.FlexViewer.dll
- C1.Web.Mvc.Olap.dll
- C1.Web.Mvc.MultiRow.dll

For ASP.NET Core MVC

- C1.Web.Mvc NuGet Package
- C1.Web.Mvc.Finance NuGet Package
- C1.Web.Mvc.FlexSheet NuGet Package
- C1.Web.Mvc.FlexViewer NuGet Package

- C1.Web.Mvc.Olap NuGet Package
- C1.Web.Mvc.MultiRow NuGet Package

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.

Razor Pages

C1 ASP.NET MVC controls supports Razor Pages with the introduction of the new **ASP.NET Core 2.0** framework.

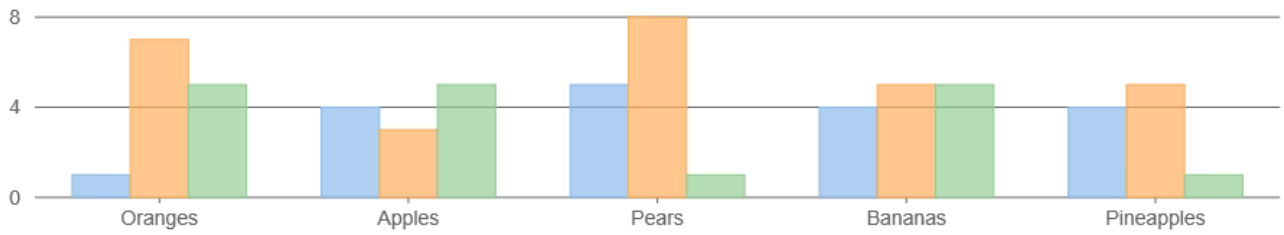
Razor Pages are similar to the view component of the MVC model. It includes the same syntax and functionality as MVC. However, Razor Pages follow a **MVVM (Model-View-ViewModel)** framework, which enables the developers to implement two-way data binding. The folder structure for the new **ASP.NET Core 2.0 Web Application** is more organized and simple to understand the concept of Razor Pages. All the pages associated with the application are inside one single **Pages** folder.

Razor Pages support multiple actions within a single page by using handlers. This would help in scenarios where your page has multiple AJAX call backs, multiple form submission, etc. To implement a Razor Page instead of a view, you need to simply add the **@page** directive at the top of the page. Once, you do that the @page makes the file into an MVC action, which means that it handles the requests directly, without going through a controller.

To understand the implementation of MVC controls using Razor Pages, we will take an example of FlexChart control, and implement it using the following steps.

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for FlexChart**
- **Step 3: Add a FlexChart control**
- **Step 4: Build and Run the Project**

The below image shows how FlexChart appears after completing the above steps.



Step 1: Create an MVC Application

Create a new ASP.NET Core MVC application using the ComponentOne templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for FlexChart

1. Add a new class to the folder **Models** (for example: `Fruit.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to the new model to define the classes that serve as a datasource for the FlexChart control.

C#

Fruit.cs

```
public class Fruit
    {
        public string Name { get; set; }
```

```
        public int MarPrice { get; set; }
        public int AprPrice { get; set; }
        public int MayPrice { get; set; }

        private IEnumerable<FruitSale> _sales = null;
        public IEnumerable<FruitSale> Sales
        {
            get
            {
                if (_sales == null)
                {
                    _sales = GetSales();
                }
                return _sales;
            }
        }

        public static IEnumerable<Fruit> GetFruitsSales()
        {
            var rand = new Random(0);
            var fruits = new[] { "Oranges", "Apples", "Pears", "Bananas",
"Pineapples" };
            var list = fruits.Select((f, i) =>
            {
                int mar = rand.Next(1, 6);
                int apr = rand.Next(1, 9);
                int may = rand.Next(1, 6);
                return new Fruit { Name = f, MarPrice = mar, AprPrice = apr,
MayPrice = may };
            });

            return list;
        }

        private IEnumerable<FruitSale> GetSales()
        {
            var rand = new Random(0);
            var today = DateTime.Now.Date;
            var firstDay = new DateTime(today.Year - 1, 3, 1);
            var dataTimes = new List<DateTime>();
            for (int i = 0; i < 92; i++)
            {
                dataTimes.Add(firstDay.AddDays(i + 1));
            }
            var list = dataTimes.Select((date, i) => {
                FruitSale sale = new FruitSale { Date = date };
                sale.SalesInChina = rand.Next(150, 250);
                if (i % 30 != 0)
                {
                    sale.SalesInUSA = rand.Next(100, 200);
```

```
                sale.SalesInJapan = rand.Next(0, 100);
            }
            else
            {
                sale.SalesInUSA = null;
                sale.SalesInJapan = null;
            }

            return sale;
        });

        return list;
    }
}

public class FruitSale
{
    public DateTime Date { get; set; }
    public int? SalesInUSA { get; set; }
    public int? SalesInChina { get; set; }
    public int? SalesInJapan { get; set; }
}
```

Back to Top

Step 3: Add a FlexChart control

Complete the following steps to initialize a FlexChart control.

1. In the Solution Explorer, right click the folder **Pages**.
2. From the context menu, select **Add | New Item**. The Add New Item dialog appears.
3. In the Add New Item dialog, select **Razor Page** and give a suitable name to the page. In our case, we use **FlexChart** as page name.
4. Click **Add**.

Once, the razor page is added to your project, you will notice two files being generated **FlexChart.cshtml** and it's corresponding **FlexChart.cshtml.cs** To make the implementation easier for the users, controller code can now be added in FlexChart.cshtml.cs file, and then initialize the control in FlexChart.cshtml.

In Code

FlexChart.cshtml.cs

C#

```
public readonly IEnumerable<Fruit> FruitsSales = Fruit.GetFruitsSales();
```

FlexChart.cshtml

C#

```
@page
@model FlexChartModel

<c1-flex-chart id="flexChart" binding-x="Name">
```

```
    <c1-items-source source-collection="@Model.FruitsSales"></c1-items-source>
    <c1-flex-chart-series binding="MarPrice">
    </c1-flex-chart-series>
    <c1-flex-chart-series binding="AprPrice">
    </c1-flex-chart-series>
    <c1-flex-chart-series binding="MayPrice">
    </c1-flex-chart-series>
</c1-flex-chart>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**ChartSample/FlexChart**) in the address bar of the browser to see the view.

Back to Top

Controls

CollectionView

ComponentOne's MVC Edition **CollectionView** is a service that implements the **ICollectionView** interface to display data in data-bound controls, such as **FlexGrid**. CollectionView is a very powerful concept since it provides a layer that wraps the actual data and performs various operations like sorting, filtering, grouping etc. without affecting the actual data. In simple terms, you can think it as **DefaultView** of the datatable.

The **CollectionView** class has the capability of keeping a track of changes made to the data and managing the current record. You can manage editing, adding, and removing items, paging etc. It implements **INotifyCollectionChanged** Interface which notifies listeners of dynamic changes, such as when an item is added and removed or when the collection is sorted, filtered, or grouped.

The CollectionView class implements the following interfaces:

- **ICollectionView**: provides current record management, custom sorting, filtering, and grouping.
- **IEditableCollectionView**: provides methods for editing, adding, and removing items.
- **IPagedCollectionView**: provides paging.

CollectionView supports **Server-Side** and **Client-Side** operations.

Server-Side Operations

The server side CollectionViewHelper is a service that enables collections to have reading, editing, filtering, grouping and sorting ability, this is similar to .Net ICollectionView. On the server, CollectionView internally handles sorting, paging, filtering requests by data bound controls. This section describes how to use **Create**, **Read**, **Update**, **Delete** and **BatchEdit** actions for CRUD operations:

- `Read()`: Retrieves data from the collection.
- `Edit()`: Handles data update request from data-bound controls. See Editing for more information.
- `BatchEdit()`: Allows update of multiple items at a time. See Batch Update for more information.

Client-Side Operations

CollectionView has a powerful client API. CollectionViewHelper internally performs server side operations like **sorting**, **filtering**, **paging** on data for data-bound controls like FlexGrid, FlexChart and other Input controls. However it is possible to explicitly perform these operations on client-side too. The Client-side operations include **Current Record Management**, **Sorting**, **Filtering**, **Grouping** and **Tracking Changes**.

> See Wijmo CollectionView API Documentation for more information on CollectionView's client-side API.

Quick Start

This section describes how to useCollectionView with FlexGrid control in your MVC web application.

This topic comprises of three steps:

- **Step 1: Create an MVC Application**
- **Step 2: Configure the Datasource for your Application**
- **Step 3: Add a FlexGrid control**
- **Step 4: Build and Run the Project**

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

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish
*			

Back to Top

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic..

Step 2: Configure the Datasource for your Application

> The example uses **C1NWind** database. The **C1NWind.mdf** file is available on your system in *C:\Users\<username>\Documents\ComponentOne Samples\ASP.NET MVC\MVC\MvcExplorer\App_Data.*

1. Add **C1NWind.mdf** file to the `AppData` folder in the Solution Explorer.
2. In the Solution Explorer, right-click `Models|Add New Item|Data`, and select **ADO.NET Entity Data Model**.
3. Name the model as **C1NWind**, and click **Add**.
4. In the Entity Data Model Wizard, select **EF Designer from database**, click **Next**. **C1NWind.mdf** database is added to the data connection dropdown.
5. Click **Next** to choose Entity Framework version, and click **Next**.
6. In the **Choose Your Database Objects and Settings**, select `Tables`, and click **Finish**.

If you can see **C1NWind.edmx** added to your project under the **Models** folder, you have successfully configured the datasource for your application.

Back to Top

Step 3: Add a FlexGrid control

Complete the following steps to initialize a FlexGrid control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C# copyCode

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using CollectionView_EN.Models;
using C1.Web.Mvc.Grid;
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using System.Data.Entity.Validation;
using System.Data.Entity;
using System.Data;
```

> **Note:** Replace **CollectionView_EN.Models;** with **<*YourMVCApplicationName*>.Models;** in the references.

5. Replace the method Index() with the following method.

IndexController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View();
}
//Instantiate a JSON request
public ActionResult GridReadCategory([C1JsonRequest]
CollectionViewRequest<Category> requestData)
{
    return this.C1Json(CollectionViewHelper.Read(requestData, db.Categories));
}
```

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view has been added for the controller.
6. In the Solution Explorer, double click `Index.cshtml` to open it.
7. Replace the default code of the **Views\Index.cshtml** file with the one given below to initialize a **FlexGrid** control.

   In this example, the **GridReadCategory** action of controller is assigned to **Bind** property of FlexGrid' ItemSource to populate data.

HTML Helpers

Razor (Index.cshtml) copyCode

```
@(Html.C1().FlexGrid().Id("fGRCView").AutoGenerateColumns(false).AllowAddNew(true)
    .AllowSorting(true).CssClass("grid")
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("500")))
    .Bind(
        ib => ib.Bind(Url.Action("GridReadCategory"))
    )
)
```

Step 4: Build and Run the Project

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

Back to Top

Features

Server-Side Operations

Reading

The server side CollectionViewHelper class defines a **Read** request to retrieve data. This enables displaying of records from the source database by handling **Read** request of CollectionViewHelper.

The following image shows how the FlexGrid appears after the **Read** request is defined.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish

The following code example demonstrates how to use **Read** request of CollectionViewHelper to display records from the datasource in the FlexGrid:

In Code

ReadController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View();
}
//Instantiate a JSON request
public ActionResult GridReadCategory([C1JsonRequest] CollectionViewRequest<Category>
requestData)
{
    return this.C1Json(CollectionViewHelper.Read(requestData, db.Categories));
}
```

In this example, the **GridReadCategory** action of controller is assigned to Bind property of FlexGrid's ItemSource to populate data.

Read.cshtml

HTML Helpers

Razor copyCode

```
@(Html.C1().FlexGrid().Id("fGRCView").AutoGenerateColumns(false).IsReadOnly(true)
    .CssClass("grid")
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("500")))
    .Bind(
        ib => ib.Bind(Url.Action("GridReadCategory"))
    )
)
```

Creating

CollectionView supports various server-side operations, including creating, reading, updating, deleting and editing of the field in data-bound controls, such as FlexGrid.

This example shows how to use **Edit** request of CollectionViewHelper, which allows adding records to source database in data-bound controls, such as FlexGrid. Here, the **GridCreateCategory** action of the controller is assigned to the Create property of FlexGrid's ItemSource.

The following image shows how the FlexGrid appears after the AllowAddNew property is set to true, and CollectionViewEditRequest is used in the controller.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish
✎	9	Abc	
*			

The following code example demonstrates how to use **Create** property of FlexGrid's ItemSource and CollectionViewEditRequest to enable creation of a record in the FlexGrid:

In Code

CreateController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View(db);
}
//Instantiate a JSON request
public ActionResult GridReadCategory([C1JsonRequest] CollectionViewRequest<Category>
requestData)
{
    return this.C1Json(CollectionViewHelper.Read(requestData, db.Categories));
}
public ActionResult
GridCreateCategory([C1JsonRequest]CollectionViewEditRequest<Category> requestData)
{
```

```
    var category = requestData.OperatingItems.First();
    if (category.CategoryName == null)
    {
        category.CategoryName = "";
    }
    return Create(requestData, db.Categories);
}

private ActionResult Create<T>(CollectionViewEditRequest<T> requestData, DbSet<T>
data) where T : class
{
    return this.C1Json(CollectionViewHelper.Edit<T>(requestData, item =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            data.Add(item);
            db.SaveChanges();
        }
        catch (DbEntityValidationException e)
        {
            error = string.Join(",", e.EntityValidationErrors.Select(result =>
            {
                return string.Join(",", result.ValidationErrors.Select(err =>
err.ErrorMessage));
            }));
            success = false;
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<T>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = item
        };
    }, () => data.ToList<T>()));
}
```

In this example, the **GridCreateCategory** action is assigned to the Create property of FlexGrid's ItemSource.

Create.cshtml

HTML Helpers

Razor	copyCode

```
@(Html.C1().FlexGrid().Id("fGRCView").AutoGenerateColumns(false).AllowAddNew(true)
```

```
    .AllowSorting(true).CssClass("grid")
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("500")))
        .Bind(ib =>
         ib.Bind(Model.Categories)
        .Create(Url.Action("GridCreateCategory"))
        )
    )
```

Disable Server Reading

CollectionView allows you to disable server side sorting, paging, filtering or scrolling in a data-bound control, such as FlexGrid by setting the value of DisableServerRead property of FlexGrid's ItemSource to **True**. By default, its value is set to False. On setting the property value to true for any operation, all the current items are transferred to the client side and the server side events get disabled. You can carry out client side operations directly without making any network calls.

This example shows how to disable server reading for Paging operation in the FlexGrid.

The following image shows how the FlexGrid appears after the DisableServerRead property is set to true, and page size is set to **10**.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

CustomerID	CompanyName	ContactName	City	Country	Phone
ALFKI	Alfreds Futter...	Maria Anders	Berlin	Germany	030-0074321
ANATR	Ana Trujillo E...	Ana Trujillo	México D.F.	Mexico	(5) 555-4729
ANTON	Antonio More...	Antonio Moreno	México D.F.	Mexico	(5) 555-3932
AROUT	Around the Horn	Thomas Hardy	London	UK	(171) 555-7788
BERGS	Berglunds sn...	Christina Berg...	Luleå	Sweden	0921-12 34 65
BLAUS	Blauer See D...	Hanna Moos	Mannheim	Germany	0621-08460
BLONP	Blondel père ...	Frédérique Ci...	Strasbourg	France	88.60.15.31
BOLID	Bólido Comid...	Martín Sommer	Madrid	Spain	(91) 555 22 82
BONAP	Bon app'	Laurence Leb...	Marseille	France	91.24.45.40
BOTTM	Bottom-Dollar...	Elizabeth Linc...	Tsawassen	Canada	(604) 555-4729

◀◀ ◀ 1 / 9 ▶ ▶▶

The following code example demonstrates how to set **DisableServerRead** property to disable server side Paging in the FlexGrid:

In Code

To enable Paging, add `_Pager.cshtml` page in the **Views|Shared** folder and replace its content with the following:

_Pager.cshtml

_Pager.cshtml

```
@model string
<div class="wj-control wj-content wj-pager">
    <div class="wj-input-group">
        <span class="wj-input-group-btn">
            <button class="wj-btn wj-btn-default demo-grid-firstpage" type="button">
                <span class="wj-glyph-left"></span>
                <span class="wj-glyph-left"></span>
            </button>
        </span>
        <span class="wj-input-group-btn">
            <button class="wj-btn wj-btn-default demo-grid-previouspage" type="button">
                <span class="wj-glyph-left"></span>
            </button>
        </span>
```

```
        <input type="text" class="wj-form-control demo-grid-currentpage" disabled="">
        <span class="wj-input-group-btn">
            <button class="wj-btn wj-btn-default demo-grid-nextpage" type="button">
                <span class="wj-glyph-right"></span>
            </button>
        </span>
        <span class="wj-input-group-btn">
            <button class="wj-btn wj-btn-default demo-grid-lastpage" type="button">
                <span class="wj-glyph-right"></span>
                <span class="wj-glyph-right"></span>
            </button>
        </span>
    </div>
</div>
<script>
    var pagingGrid = wijmo.Control.getControl('#@(Model)'),
        cv,
        pagerButtons = Array.prototype.slice.call(document.querySelectorAll('.wj-pager button'));
    if (pagingGrid) {
        cv = pagingGrid.collectionView;
    }
    if (!cv) {
        throw "the collectionview is null";
    }
    // update pager when user clicks a button
    pagerButtons.forEach(function (el) {
        el.addEventListener('click', function () {
            var className = el.className;
            if (className.indexOf("firstpage") > -1) {
                cv.moveToFirstPage();
            } else if (className.indexOf("lastpage") > -1) {
                cv.moveToLastPage();
            } else if (className.indexOf("previouspage") > -1) {
                cv.moveToPreviousPage();
            } else if (className.indexOf("nextpage") > -1) {
                cv.moveToNextPage();
            }
        });
    });
    cv.collectionChanged.addHandler(function () {
        updatePager();
    });
    updatePager();
    // disable/enable buttons and update display text for pager
    function updatePager() {
        // get buttons by id
        var display = document.querySelector(".wj-pager .demo-grid-currentpage"),
            next = document.querySelector(".wj-pager .demo-grid-nextpage"),
            previous = document.querySelector(".wj-pager .demo-grid-previouspage"),
            first = document.querySelector(".wj-pager .demo-grid-firstpage"),
            last = document.querySelector(".wj-pager .demo-grid-lastpage"),
            enableBackwards,
            enableForwards;
        // update the pager text
        display.value = (cv.pageIndex + 1) + ' / ' + (cv.pageCount);
        // determine which pager buttons to enable/disable
        enableBackwards = cv.pageIndex <= 0;
        enableForwards = cv.pageIndex >= cv.pageCount - 1;
        // enable/disable pager buttons
        previous.disabled = enableBackwards;
        first.disabled = enableBackwards;
        next.disabled = enableForwards;
        last.disabled = enableForwards;
    }
</script>
```

DisableServerReadController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View(db);
}
//Instantiate a JSON request
public ActionResult GridReadCategory([C1JsonRequest] CollectionViewRequest<Category> requestData)
{
    return this.C1Json(CollectionViewHelper.Read(requestData, db.Categories));
}
```

DisableServerRead.cshtml

HTML Helpers

Razor copyCode

```
@(Html.C1().FlexGrid().Id("fGDisableServerView").IsReadOnly(true).AllowSorting(true).AutoGenerateColumns(false)
    .Bind(b => b.DisableServerRead(true).PageSize(10).Bind(Model.Customers)).CssStyle("height", "100%")
    .Columns(columns => columns
        .Add(c => c.Binding("CustomerID"))
        .Add(c => c.Binding("CompanyName"))
        .Add(c => c.Binding("ContactName"))
        .Add(c => c.Binding("City"))
        .Add(c => c.Binding("Country"))
        .Add(c => c.Binding("Phone"))
        )
)
@Html.Partial("_Pager", "fGDisableServerView")
```

JS

Script copyCode

```
<script>
$(document).ready(function () {
    //Disable Server Reading
    fGDisableServerView = wijmo.Control.getControl('#fGDisableServerView');
});

//Disable Server Read
var fGDisableServerView = null;
</script>
```

Editing

The server side CollectionViewHelper class defines a Edit request to handle updates. This example shows how to define update action which enables updating records in source database by handling **Edit** request of CollectionViewHelper. It lets a user edit the data in the FlexGrid, and update it in the source database.

The following image shows how the FlexGrid appears after the CollectionViewEditRequest is used in the controller.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
✎	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish

The following code example demonstrates how to handle CollectionViewEditRequest to enable editing and updating of a record in the FlexGrid:

In Code

EditController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View(db);
}
//Instantiate a JSON request
public ActionResult
GridUpdateCategory([C1JsonRequest]CollectionViewEditRequest<Category> requestData)
{
    return Update(requestData, db.Categories);
}
private ActionResult Update<T>(CollectionViewEditRequest<T> requestData, DbSet<T>
data) where T : class
{
    return this.C1Json(CollectionViewHelper.Edit<T>(requestData, item =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            db.Entry(item as object).State = EntityState.Modified;
            db.SaveChanges();
        }
        catch (DbEntityValidationException e)
        {
            error = string.Join(",", e.EntityValidationErrors.Select(result =>
            {
                return string.Join(",", result.ValidationErrors.Select(err =>
err.ErrorMessage));
            }));
```

```
            success = false;
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<T>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = item
        };
    }, () => data.ToList<T>()));
}
```

In this example, the **Update** action is assigned to the Update property of FlexGrid's ItemSource.

Edit.cshtml

HTML Helpers

Razor copyCode

```
@(Html.C1().FlexGrid().Id("fGUCView").AutoGenerateColumns(false)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID").IsReadOnly(true))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("500")))
    .Bind(ib =>
        ib.Bind(Model.Categories)
        .Update(Url.Action("GridUpdateCategory"))
        )
)
```

Batch Update

CollectionViewHelper has **BatchEdit** request that lets a user commit multiple changes to the database. The example shows how to define BatchEdit request in controller to update database with CollectionView by using BatchEdit request of CollectionViewHelper.

The following image shows how the FlexGrid appears after setting the BatchEditing request of CollectionViewHelper. A user can commit changes to the server by clicking the **Update** button.

> While performing **BatchEdit** in FlexGrid, if you do sorting, filtering or paging in the grid, an update request is made to the server, which automatically updates the data in source database. Hence, ensure that the **DisableServerRead** property of ItemSource is set to **True**.
>
> On setting the property value to true, batch update can only be performed by explicitly calling the Commit method of CollectionView in the **Update** button as shown in the example below:

Update

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish
	9	Desserts	Cookies, ice-creams, pastries, cakes
*			

The following code example demonstrates how to use **BatchEdit** property and CollectionViewEditRequest to enable updating of multiple records simultaneously in the FlexGrid:

In Code

BatchUpdateController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();

// GET: Home
public ActionResult Index()
{
    return View(db);
}

public ActionResult GridBatchEdit([C1JsonRequest]CollectionViewBatchEditRequest<Category>
requestData)
{
    return this.C1Json(CollectionViewHelper.BatchEdit(requestData, batchData =>
    {
        var itemresults = new List<CollectionViewItemResult<Category>>();
        string error = string.Empty;
        bool success = true;
        try
        {
            if (batchData.ItemsCreated != null)
            {
                batchData.ItemsCreated.ToList().ForEach(st =>
                {
                    db.Categories.Add(st);
                    itemresults.Add(new CollectionViewItemResult<Category>
                    {
                        Error = "",
                        Success = ModelState.IsValid,
                        Data = st
```

```
                    });
                });
            }
            if (batchData.ItemsDeleted != null)
            {
                batchData.ItemsDeleted.ToList().ForEach(category =>
                {
                    var fCategory = db.Categories.Find(category.CategoryID);
                    db.Categories.Remove(fCategory);
                    itemresults.Add(new CollectionViewItemResult<Category>
                    {
                        Error = "",
                        Success = ModelState.IsValid,
                        Data = category
                    });
                });
            }
            if (batchData.ItemsUpdated != null)
            {
                batchData.ItemsUpdated.ToList().ForEach(category =>
                {
                    db.Entry(category).State = EntityState.Modified;
                    itemresults.Add(new CollectionViewItemResult<Category>
                    {
                        Error = "",
                        Success = ModelState.IsValid,
                        Data = category
                    });
                });
            }
            db.SaveChanges();
        }
        catch (DbEntityValidationException e)
        {
            error = string.Join(",", e.EntityValidationErrors.SelectMany(i =>
i.ValidationErrors).Select(i => i.ErrorMessage));
            success = false;
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }

        return new CollectionViewResponse<Category>
        {
            Error = error,
            Success = success,
            OperatedItemResults = itemresults
        };
    }, () => db.Categories.ToList()));
}
```

In this example, the **BatchEdit** request is assigned to BatchEdit property of FlexGrid's ItemsSource.

BatchUpdate.cshtml

HTML Helpers

Razor copyCode

```
<input type="button" value="Update" class="btn" onclick="batchUpdate()" />
@(Html.C1().FlexGrid().Id("fGBECView").AutoGenerateColumns(false)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("500")))
    .Bind(ib =>
ib.Bind(Model.Categories).DisableServerRead(true).BatchEdit(Url.Action("GridBatchEdit")))
    .AllowAddNew(true)
    .AllowDelete(true)
    .CssClass("grid")
)
```

JS

Script copyCode

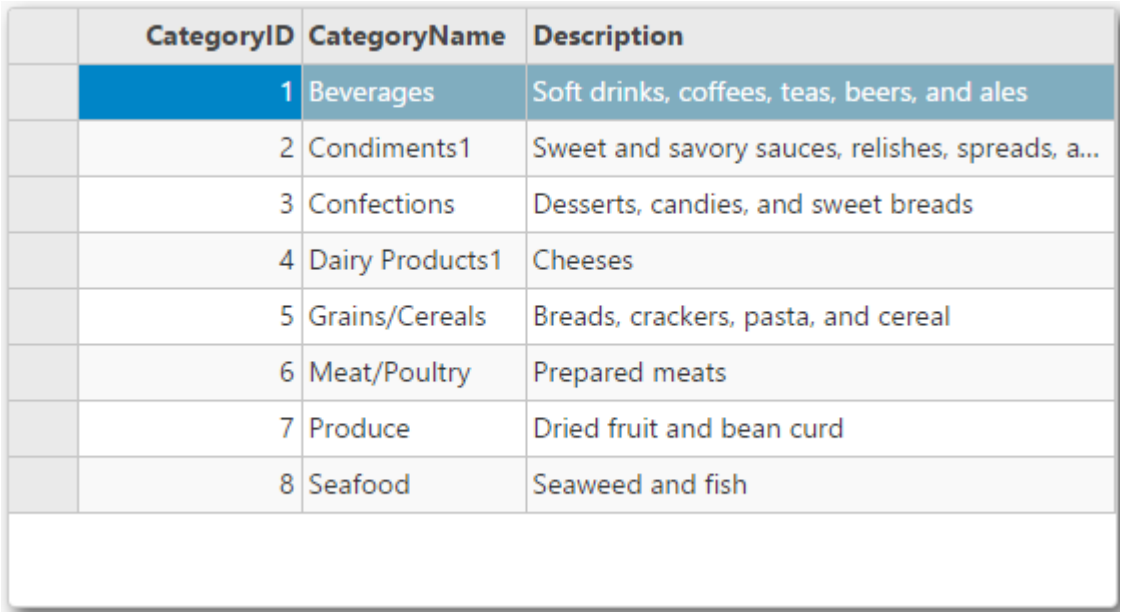
```
<script>
    //Batch Edit
    function batchUpdate() {
        var batchEditGrid = wijmo.Control.getControl('#fGBECView'),
        cv = batchEditGrid.collectionView;
        cv.commit();
    };
</script>
```

Deleting

This example shows how to define action in controller to delete rows from database by handling Edit request of **CollectionViewHelper**.

The following image shows how the FlexGrid appears after the CollectionViewEditRequest is used in the controller.

The example uses **C1NWind** datasource, which was configured in the application in the Quick Start. To **delete** a row, click the row header to select the row, and click **Delete** button on the keyboard. The selected row is deleted and the source database is updated accordingly.



	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments1	Sweet and savory sauces, relishes, spreads, a...
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products1	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish

The following code demonstrates how to handle CollectionViewEditRequest to enable deleting of a record in the FlexGrid:

In Code

DeleteController.cs

C# copyCode

```
//Define datasource for FlexGrid
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View(db);
}
//Instantiate a JSON request
public ActionResult
GridDeleteCategory([C1JsonRequest]CollectionViewEditRequest<Category> requestData)
{
    return Delete(requestData, db.Categories, item => item.CategoryID);
}
private ActionResult Delete<T>(CollectionViewEditRequest<T> requestData, DbSet<T>
data, Func<T, object> getKey) where T : class
{
    return this.C1Json(CollectionViewHelper.Edit<T>(requestData, item =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            var resultItem = data.Find(getKey(item));
            data.Remove(resultItem);
            db.SaveChanges();
        }
        catch (DbEntityValidationException e)
```

```
        {
            error = string.Join(",", e.EntityValidationErrors.Select(result =>
            {
                return string.Join(",", result.ValidationErrors.Select(err =>
err.ErrorMessage));
            }));
            success = false;
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<T>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = item
        };
    }, () => data.ToList<T>()));
}
```

In this example, the **Delete** action is assigned to the Delete property of FlexGrid's ItemSource.

Delete.cshtml

HTML Helpers

Razor copyCode

```
@(Html.C1().FlexGrid().Id("fGDelCView").AutoGenerateColumns(false).IsReadOnly(true)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("*")))
    .AllowDelete(true)
    .Bind(
        ib => ib.Bind(Model.Categories)
        .Delete(Url.Action("GridDeleteCategory"))
    )
)
```

Multiple Controls Data Binding

CollectionView supports multiple controls binding in an MVC application by using CollectionViewService. You can use different data-bound controls, such as FlexGrid and FlexChart and bind them using CollectionView.

First of all, you need to create a CollectionView and set its Id. Initialize the data-bound controls, such as FlexGrid and FlexChart, and assign the CollectionView Id in their ItemsSourceId to bind the controls using a common CollectionView.

CollectionView can be used for multiple items-bound controls, they will have the same itemsSource and if we change

one controls's data, other controls will change immediately. It means we only need create the CollectionView once, and can use it for multiple different controls.

This section describes how to use CollectionViewService with FlexGrid and FlexChart control in your MVC Webforms.

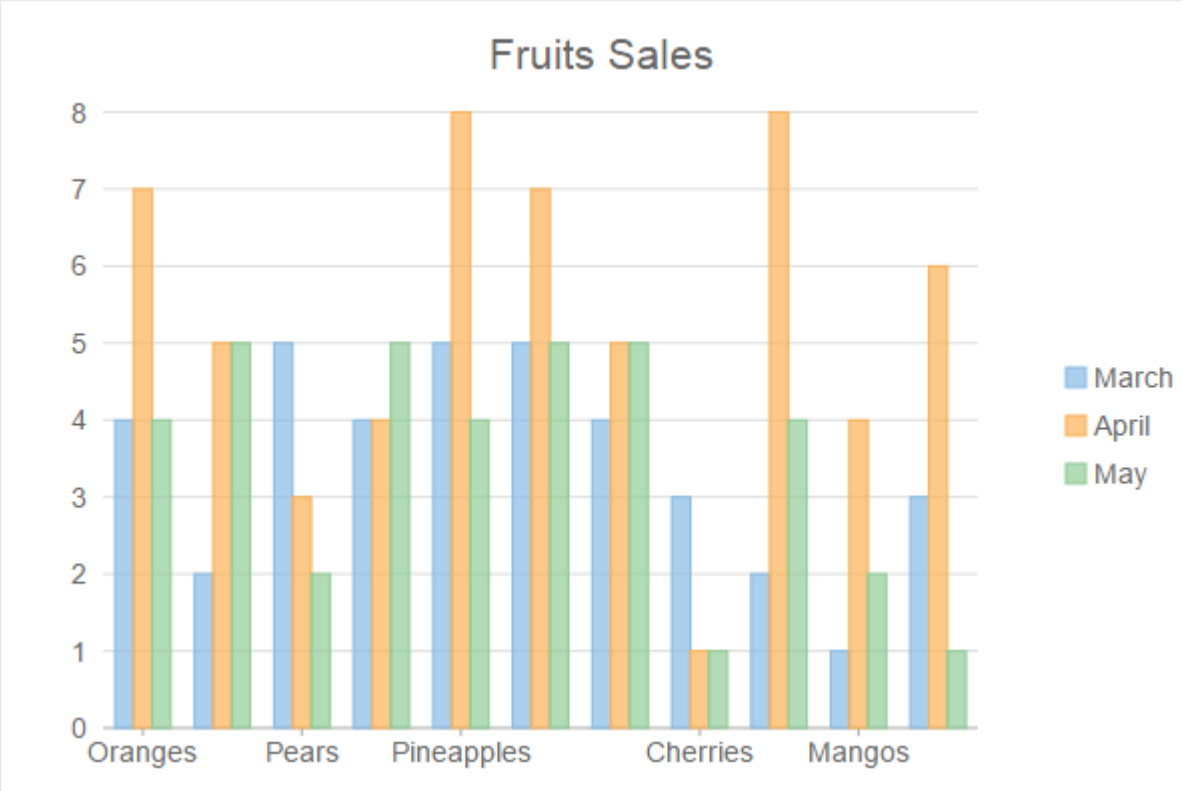
This topic comprises of three steps:

- **Step 1: Configure the Datasource for your Application**
- **Step 2: Initialize a FlexGrid control and set its ItemsSourceId**
- **Step 3: Initialize a FlexChart control and set its ItemsSourceId**

The following image shows how the FlexGrid and FlexChart appear after binding them through CollectionView using **CollectionViewService**.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

	ID	Name	Country	MarPrice	AprPrice	MayPrice
	1	Oranges	France	4	7	4
	2	Apples	Japan	2	5	5
	3	Pears	China	5	3	2
	4	Bananas	US	4	4	5
	5	Pineapples	Canada	5	8	4
	6	Peaches	US	5	7	5
	7	Strawberries	German	4	5	5
	8	Cherries	China	3	1	1
	9	Grapes	German	2	8	4
	10	Mangos	Korea	1	4	2
	11	Lemons	Canada	3	6	1
*						



Step 1: Configure the Datasource for your Application

1. Add a new class, **Fruit.cs** to the folder Models. See Adding controls to know how to add a new model.
2. Replace the following code in the new model to define the classes that serve as a datasource for the FlexGrid and FlexChart controls.

Fruit.cs copyCode

```
public class Fruit
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int MarPrice { get; set; }
    public int AprPrice { get; set; }
    public int MayPrice { get; set; }

    public string Country { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "UK",
"Canada", "Japan", "China", "France", "German", "Italy", "Korea", "Australia" };
    public static IEnumerable<Fruit> GetFruitsSales()
    {
        var rand = new Random(0);
        var fruits = new[] { "Oranges", "Apples", "Pears", "Bananas",
"Pineapples",
            "Peaches", "Strawberries", "Cherries", "Grapes", "Mangos",
"Lemons"};
        var list = fruits.Select((f, i) =>
        {
            int id = i + 1;
            int mar = rand.Next(1, 6);
```

```
            int apr = rand.Next(1, 9);
            int may = rand.Next(1, 6);
            var country = COUNTRIES[rand.Next(0, COUNTRIES.Count - 1)];
            return new Fruit { ID = id, Country = country, Name = f, MarPrice =
mar, AprPrice = apr, MayPrice = may };
        });

        return list;
    }
}
```

Back to Top

The following code example demonstrates how use **CollectionViewService** to enable multiple control binding in the chart and grid controls:

In Code

MultipleControlsBindController.cs

C# copyCode

```
public static List<Fruit> fruitsSales = Fruit.GetFruitsSales().ToList();
public ActionResult Index()
{
   return View(fruitsSales);
}

public ActionResult Update([C1JsonRequest]CollectionViewEditRequest<Fruit>
requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Fruit>(requestData, fruit =>
    {
        string error = string.Empty;
        bool success = true;
        var fSale = fruitsSales.Find(item => item.ID == fruit.ID);
        fSale.Name = fruit.Name;
        fSale.Country = fruit.Country;
        fSale.MayPrice = fruit.MayPrice;
        fSale.MarPrice = fruit.MarPrice;
        fSale.AprPrice = fruit.AprPrice;
        return new CollectionViewItemResult<Fruit>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = fSale
        };
    }, () => fruitsSales));
}

public ActionResult Create([C1JsonRequest]CollectionViewEditRequest<Fruit>
requestData)
{
```

```
    return this.C1Json(CollectionViewHelper.Edit<Fruit>(requestData, fruit =>
    {
        string error = string.Empty;
        bool success = true;
        fruit.ID = getId();
        fruitsSales.Add(fruit);
        return new CollectionViewItemResult<Fruit>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = fruit
        };
    }, () => fruitsSales));
}

public ActionResult Delete([C1JsonRequest]CollectionViewEditRequest<Fruit>
requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Fruit>(requestData, fruit =>
    {
        string error = string.Empty;
        bool success = true;
        var fSale = fruitsSales.Find(item => item.Name == fruit.Name);
        fruitsSales.Remove(fSale);
        return new CollectionViewItemResult<Fruit>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = fruit
        };
    }, () => fruitsSales));
}

private int getId()
{
    int id = 1;
    while (fruitsSales.Find(item => item.ID == id) != null)
    {
        id++;
    }
    return id;
}
```

MultipleControlsBind.cshtml

HTML Helpers

Razor	copyCode

```
@using CollectionView_EN.Models;
@using C1.Web.Mvc;
```

```
@using System.Collections.Generic;
@using C1.Web.Mvc.Fluent;

@model List<Fruit>

        @(Html.C1().CollectionViewService().Id("fruitsSales")
                                .Bind(Model).Update(Url.Action("Update", "Home"))
                                .Delete(Url.Action("Delete", "Home"))
                                .Create(Url.Action("Create", "Home")))

                @(Html.C1().FlexGrid<Fruit>().Id("flexGrid")
                        .Columns(cls =>
                        {
                            cls.Add(col =>
col.Binding("ID").Visible(true).IsReadOnly(true));
                            cls.Add(col => col.Binding("Name").Width("*"));
                            cls.Add(col => col.Binding("Country").Width("*"));
                            cls.Add(col => col.Binding("MarPrice").Width("*"));
                            cls.Add(col => col.Binding("AprPrice").Width("*"));
                            cls.Add(col => col.Binding("MayPrice").Width("*"));
                        })
                    .IsReadOnly(true)
                    .AllowSorting(true)
                    .ItemsSourceId("fruitsSales")
                    .AutoGenerateColumns(false)
                    .IsReadOnly(false)
                    .AllowAddNew(true)
                    .AllowDelete(true)
                    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
                    .Width(600)
                    .Height(400)
                )
                @(Html.C1().FlexChart().Id("flexChart").Header("Fruits Sales")

.ItemsSourceId("fruitsSales").BindingX("Name").Series(sers =>
                    {
                        sers.Add().Binding("MarPrice").Name("March");
                        sers.Add().Binding("AprPrice").Name("April");
                        sers.Add().Binding("MayPrice").Name("May");
                    }).SelectionMode(C1.Web.Mvc.Chart.SelectionMode.Point)
                    .ChartType(C1.Web.Mvc.Chart.ChartType.Column)
                    .Rotated(false)
                    .Height(400)
                    .Width(600)
                )
```

Tag Helpers

HTML

```
@model List<Fruit>
```

```
<c1-items-source id="fruitsSales" source-collection="Model" update-action-
url="/Home/Update"
     delete-action-url="/Home/Delete" create-action-url="/Home/Create">

</c1-items-source>
  <div>
    <div>
    <c1-flex-grid id="flexGrid" is-read-only="false" auto-generate-columns="false"
        allow-add-new="true" allow-delete="true" items-source-id="fruitsSales"
         selection-mode="C1.Web.Mvc.Grid.SelectionMode.Row">
    <c1-flex-grid-column binding="ID" width="*" is-read-only="true"></c1-flex-grid-
column>
    <c1-flex-grid-column binding="Name" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="MarPrice" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="AprPrice" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="MayPrice" width="*"></c1-flex-grid-column>
    </c1-flex-grid>
   </div>
 <div>
   <c1-flex-chart id="flexChart" items-source-id="fruitsSales" binding-x="Name"
         selection-mode="C1.Web.Mvc.Chart.SelectionMode.Point" header="Fruits Sales"
         chart-type="ChartType.Column" rotated="False">
   <c1-flex-chart-series binding="MarPrice" name="March"></c1-flex-chart-series>
   <c1-flex-chart-series binding="AprPrice" name="April"></c1-flex-chart-series>
   <c1-flex-chart-series binding="MayPrice" name="May"></c1-flex-chart-series>
   <c1-flex-chart chart-type="ChartType.Column"></c1-flex-chart>
   </c1-flex-chart>
   </div>
 </div>
```

Client-Side Operations

Current Record Management

CollectionView can manage the current record by using the **ICollectionView** interface.

To obtain the current position of a record in the collection, use **currentPosition** property. We also use the methods **moveCurrentTo(item), moveCurrentToFirst(), moveCurrentToLast(), moveCurrentToNext(), moveCurrentToPosition(index) and moveCurrentToPrevious()** to change the current position. When the current is changed, we use the events **currentChanging** and **currentChanged** to track it. We can cancel the current changing in the event **currentChanging**.

> Make sure that the DisableServerRead property of ItemSource is set to **True** if filtering, paging, sorting is to be performed on data available at client side only.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

The following code example demonstrates how to manage current records in FlexGrid through CollectionView.

In Code

CurrentRecordManagementController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();

public ActionResult Index()
{
    return View(db);
}
```

CurrentRecordManagement.cshtml

HTML Helpers

Razor copyCode

```
<div>
    <button class="btn btn-default" id="btnCRMMoveNext">Move To Next</button>
    <button class="btn btn-default" id="btnCRMMovePre">Move To Previous</button>
    <button class="btn btn-default" id="btnCRMStop4">Stop in 4th Row</button>
    <button class="btn btn-default" id="btnCRMReset">Clear Stopping</button>
</div>
@(Html.C1().FlexGrid().Id("crmGrid").IsReadOnly(true).SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
.Height(300).Width(800).AutoGenerateColumns(true).Bind(b =>
b.DisableServerRead(true).Bind(Model.Customers))
)
```

JS

Script copyCode

```
<script>
    $(document).ready(function () {
        //Current Record Management
        crmGrid = wijmo.Control.getControl('#crmGrid');
        cvCRM = crmGrid.itemsSource; //new wijmo.collections.CollectionView(getData(10)),

        // Add the processes for buttons' click
        // Move to the Next item
        document.getElementById('btnCRMMoveNext').addEventListener('click', function () {
```

```
            cvCRM.moveCurrentToNext();
        });

        // Move to the previous item
        document.getElementById('btnCRMMovePre').addEventListener('click', function () {
            cvCRM.moveCurrentToPrevious();
        });

        // When the current item is the 4th one, restrict any change
        document.getElementById('btnCRMStop4').addEventListener('click', function () {
            cvCRM.currentChanging.addHandler(stopCurrentIn4th);
        });

        // Restore to be able to change
        document.getElementById('btnCRMReset').addEventListener('click', function () {
            cvCRM.currentChanging.removeHandler(stopCurrentIn4th);
        });

        // define the function to forbid the current moving.
        function stopCurrentIn4th(sender, e) {
            // when the current is the 4rd item, stop moving.
            if (sender.currentPosition === 3) {
                e.cancel = true;
            }
        };
    });

    // create collectionview, grid
    var crmGrid = null
        , cvCRM = null;

</script>
```

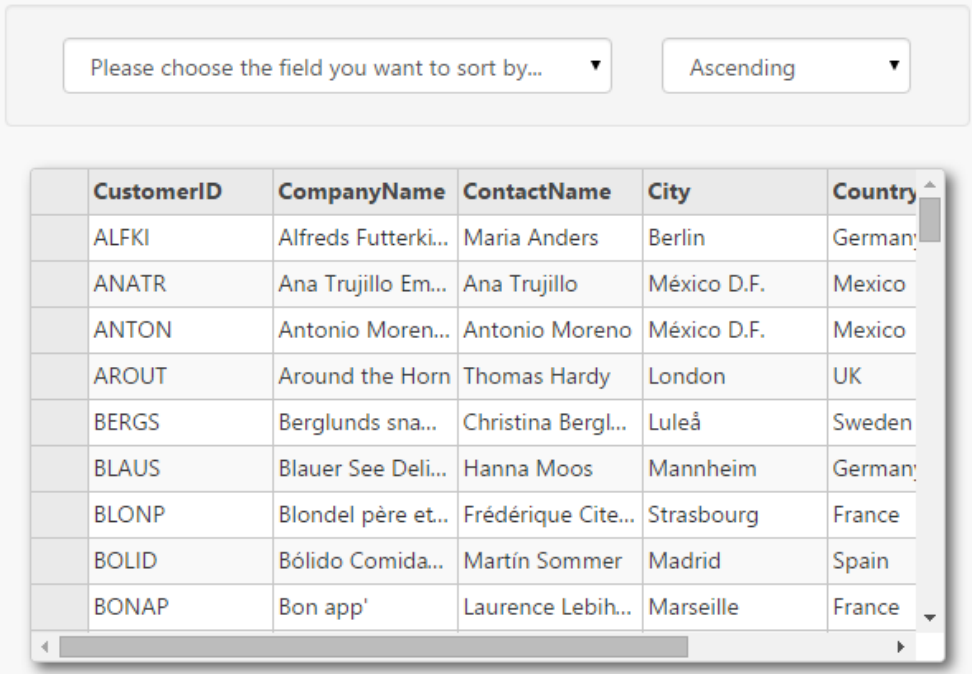
Sorting

The **CollectionView** class supports sorting through the **ICollectionView** interface, similar to .NET. To enable sorting, add one or more **sortDescriptions** objects to the **CollectionView.sortDescriptions** property. The result can be obtained from the **CollectionView.items** property.

SortDescription objects are flexible, allowing you to sort data based on value in ascending or descending order. In the sample below, you can sort the collection based on the corresponding field value chosen in the first list. You can also specify the sorting order in the second list.

> Make sure that the **DisableServerRead** property of ItemSource is set to **True** if filtering, paging, sorting is to be performed on data available at client side only.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:



The following code example demonstrates how to apply sorting in FlexGrid through CollectionView.

In Code

SortingController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();

public ActionResult Index()
{
    return View(db);
}
```

Sorting.cshtml

HTML Helpers

Razor copyCode

```
<div class="row-fluid well row">
    <div class="col-md-8">
        <select id="sortingFieldNameList" class="form-control"></select>
    </div>
    <div class="col-md-4">
        <select id="sortingOrderList" class="form-control">
            <option value="true" selected="selected">Ascending</option>
            <option value="false">Descending</option>
        </select>
    </div>
</div>

@(Html.C1().FlexGrid().Id("sortingGrid").IsReadOnly(true).AllowSorting(false).AutoGenerateColumns(false)
    .Bind(b => b.DisableServerRead(true).Bind(Model.Customers))
    .Columns(columns => columns
        .Add(c => c.Binding("CustomerID"))
        .Add(c => c.Binding("CompanyName"))
        .Add(c => c.Binding("ContactName"))
        .Add(c => c.Binding("City"))
```

```
        .Add(c => c.Binding("Country"))
        .Add(c => c.Binding("Phone"))
        )
)
```

JS

Script copyCode

```
<script>
    function getNames() {
        return ['CustomerID', 'CompanyName', 'ContactName', 'City', 'Country', 'Phone'];
    };

    $(document).ready(function () {
        //Sorting
        sortingGrid = wijmo.Control.getControl('#sortingGrid');
        cvSorting = sortingGrid.itemsSource;
        sortingFieldNameList = document.getElementById('sortingFieldNameList');
        sortingOrderList = document.getElementById('sortingOrderList');
        // initialize the list items for field names and orders.
        sortingFieldNameList.innerHTML += '<option value="" selected="selected">Please choose the field
you want to sort by...</option>';
        for (var i = 0; i < sortingNames.length; i++) {
            sortingFieldNameList.innerHTML += '<option value="' + sortingNames[i] + '">' +
sortingNames[i] + '</option>';
        }

        // track the list change in order to udpate the sortDescriptions property.
        sortingFieldNameList.addEventListener('change', sortGrid);
        sortingOrderList.addEventListener('change', sortGrid);

        //Sorting
        // create collectionview, grid, the jQuery elements, the field name list.
        var cvSorting = null,
            sortingGrid = null,
            sortingFieldNameList = null,
            sortingOrderList = null,
            sortingNames = getNames();

        function sortGrid() {
            var fieldName = sortingFieldNameList.value,
                ascending = sortingOrderList.value,
                sd, sdNew;

            if (!fieldName) {
                return;
            }

            ascending = ascending === 'true';
            sd = cvSorting.sortDescriptions;
            sdNew = new wijmo.collections.SortDescription(fieldName, ascending);

            // remove any old sort descriptors and add the new one
            sd.splice(0, sd.length, sdNew);
        };

    })
</script>
```

Grouping

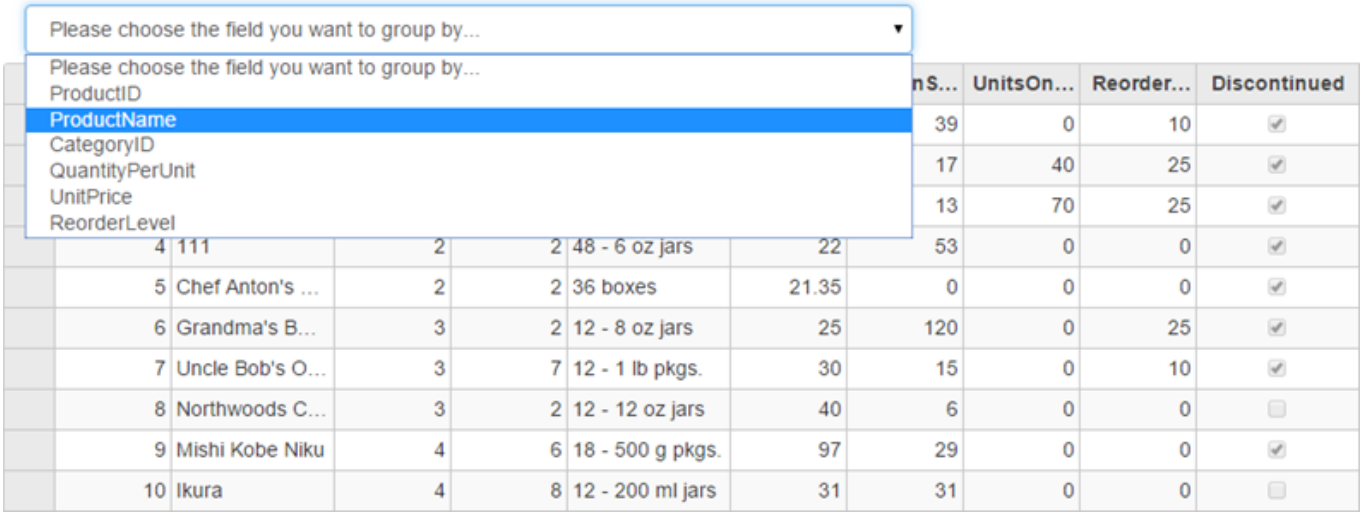
The **CollectionView** class supports grouping through the **ICollectionView** interface, similar to the one in .NET. To enable grouping, add one or more **GroupDescription** objects to the CollectionView.groupDescriptions property, and ensure that the grid's ShowGroups property is set to true when creating the grid instance(the default value is false.).

GroupDescription objects are flexible, allowing you to group data based on value or on grouping functions.

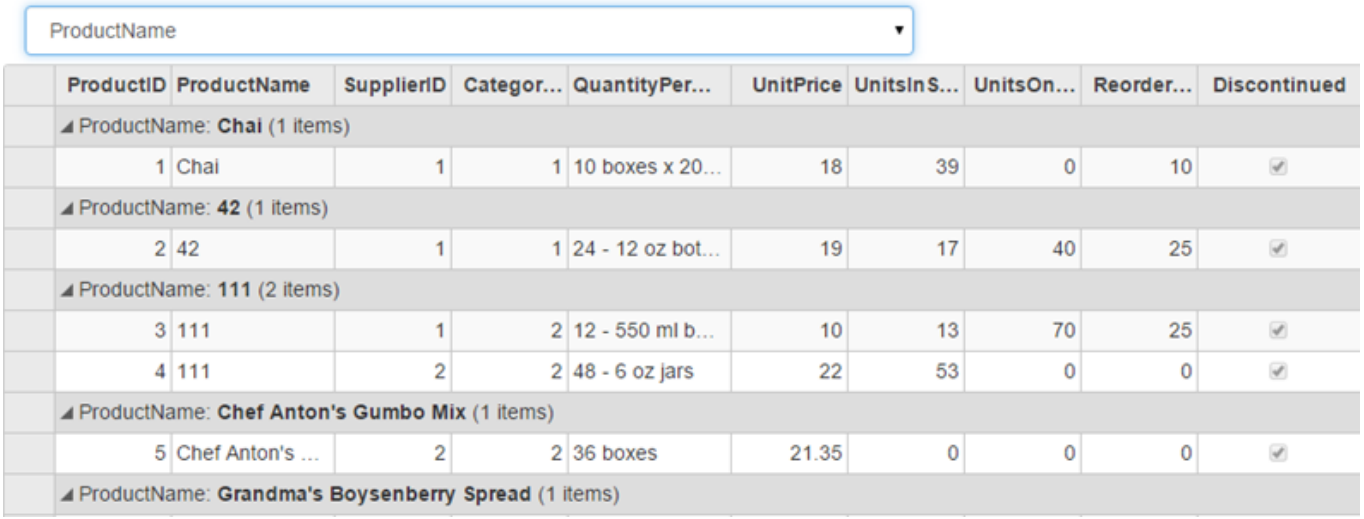
The example below groups the collection by the field which you select from the list. The grid shows not only the items content but also the group information: the group name and the average value of amount in the group.

> Make sure that the **DisableServerRead** property of ItemSource is set to **True** if filtering, paging, sorting is to be performed on data available at client side only.

The example uses C1NWind datasource, which was configured in the application in the Quick Start. The image below shows how the FlexGrid appears after grouping is applied on it:



You can choose the field that you want the FlexGrid to be grouped by from the drop-down.



The following code example demonstrates how to apply grouping in FlexGrid through CollectionView.

In Code

GroupingController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();

public ActionResult Index()
{
    return View(db);
}
```

Grouping.cshtml

HTML Helpers

Razor copyCode

```
<div class="col-md-6">
    <select id="groupingFieldNameList" class="form-control"></select>
</div>

    @(Html.C1().FlexGrid().Id("groupingGrid").IsReadOnly(true).AllowSorting(true)
    .AutoGenerateColumns(true).PageSize(10).Height(500)
    .Bind(b => b.DisableServerRead(true).Bind(Model.Products))
    )
```

JS

Script copyCode

```
<script>
       function getGroupNames() {
           return ['ProductID', 'ProductName', 'CategoryID', 'QuantityPerUnit',
'UnitPrice', 'ReorderLevel'];
       };
       $(document).ready(function () {
           //Grouping

           groupingGrid = wijmo.Control.getControl('#groupingGrid');
           cvGrouping = groupingGrid.itemsSource;
           groupingFieldNameList = document.getElementById('groupingFieldNameList');
           groupingFieldNameList.addEventListener('change', groupGrid);

           // Initialize the list and listen to the list's change.
           groupingFieldNameList.innerHTML += '<option value=""
selected="selected">Please choose the field you want to group by...</option>';
           for (var i = 0; i < groupingNames.length; i++) {
               groupingFieldNameList.innerHTML += '<option value="' +
groupingNames[i] + '">' + groupingNames[i] + '</option>';
           }
```

```
        });
        // Grouping in FlexGrid
        // create collectionview, grid, the select element and the names list.
        var cvGrouping = null,
            groupingGrid = null,
            groupingFieldNameList = null,
            groupingNames = getGroupNames();

        // update the group settings.
        function groupGrid() {
            var gd,
                fieldName = groupingFieldNameList.value;
            gd = cvGrouping.groupDescriptions;
            if (!fieldName) {
                // clear all the group settings.
                gd.splice(0, gd.length);
                return;
            }
            if (findGroup(fieldName) >= 0) {
                return;
            }
            if (fieldName === 'UnitPrice') {
                // when grouping by amount, use ranges instead of specific values
                gd.push(new wijmo.collections.PropertyGroupDescription(fieldName,
function (item, propName) {
                    var value = item[propName]; // UnitPrice
                    if (value > 100) return 'Large Amounts';
                    if (value > 50) return 'Medium Amounts';
                    if (value > 0) return 'Small Amounts';
                    return 'Negative Amounts';
                }));
            }
            else {
                // group by specific property values
                gd.push(new wijmo.collections.PropertyGroupDescription(fieldName));
            }
        };
        // check whether the group with the specified property name already exists.
        function findGroup(propName) {
            var gd = cvGrouping.groupDescriptions;
            for (var i = 0; i < gd.length; i++) {
                if (gd[i].propertyName === propName) {
                    return i;
                }
            }
            return -1;
        };
    </script>
```

Filtering

The **CollectionView** class supports filtering through the **ICollectionView interface**, similar to .NET. To enable filtering, set the **CollectionView.filter** property to a function that determines which objects to be included in the view.

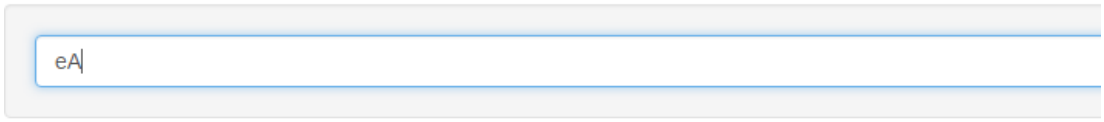
The following image shows how the FlexGrid appears after filtering is applied to the FlexGrid control.

Make sure that the **DisableServerRead** property of ItemSource is set to **True** if filtering, paging, sorting is to be performed on data available at client side only.

The example uses C1NWind datasource, which was configured in the application in the Quick Start:

	CustomerID	CompanyName	ContactName	City	Country	Phone
	ALFKI	Alfreds Futter...	Maria Anders	Berlin	Germany	030-0074321
	ANATR	Ana Trujillo E...	Ana Trujillo	México D.F.	Mexico	(5) 555-4729
	ANTON	Antonio More...	Antonio Moreno	México D.F.	Mexico	(5) 555-3932
	AROUT	Around the Horn	Thomas Hardy	London	UK	(171) 555-7788
	BERGS	Berglunds sn...	Christina Berg...	Luleå	Sweden	0921-12 34 65
	BLAUS	Blauer See D...	Hanna Moos	Mannheim	Germany	0621-08460
	BLONP	Blondel père ...	Frédérique Ci...	Strasbourg	France	88.60.15.31
	BOLID	Bólido Comid...	Martín Sommer	Madrid	Spain	(91) 555 22 82
	BONAP	Bon app'	Laurence Leb...	Marseille	France	91.24.45.40
	BOTTM	Bottom-Dollar...	Elizabeth Linc...	Tsawassen	Canada	(604) 555-4729

Input any character to filter the grid data. For instance, the image below displays how the FlexGrid appears when we enter **eA**.



	CustomerID	CompanyName	ContactName	City	Country	Phone
	EASTC	Eastern Conn...	Ann Devon	London	UK	(171) 555-0297
	GREAL	Great Lakes ...	Howard Snyder	Eugene	USA	(503) 555-7555
	OCEAN	Océano Atlánt...	Yvonne Monc...	Buenos Aires	Argentina	(1) 135-5333
	SAVEA	Save-a-lot Ma...	Jose Pavarotti	Boise	USA	(208) 555-8097

The following code example demonstrates how apply filtering in FlexGrid using CollectionView.

In Code

FilteringController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();

public ActionResult Index()
{
    return View(db);
}
```

Filtering.cshtml

HTML Helpers

Razor copyCode

```
<div>
    <input id="filteringInput" type="text" class="form-control app-pad"
```

```
      placeholder="Please input the character you want filter by Customer ID" />
</div>
@(Html.C1().FlexGrid().Id("filteringGrid").IsReadOnly(true).AllowSorting(false).AutoGenerateColumns(false)
    .Bind(b => b.Bind(Model.Customers).DisableServerRead(true))
    .Columns(columns => columns
        .Add(c => c.Binding("CustomerID"))
        .Add(c => c.Binding("CompanyName"))
        .Add(c => c.Binding("ContactName"))
        .Add(c => c.Binding("City"))
        .Add(c => c.Binding("Country"))
        .Add(c => c.Binding("Phone"))
        )
)
```

JS

Script copyCode

```
<script>

    $(document).ready(function () {
        //Filtering
        // create collectionview, grid, filter with timeout, textbox for inputting filter.
        filteringGrid = wijmo.Control.getControl('#filteringGrid');
        cvFiltering = filteringGrid.itemsSource;
        filteringInput = document.getElementById('filteringInput');
        // apply filter when input
        filteringInput.addEventListener('input', filterGrid);
    });

    //Filtering
    // create collectionview, grid, filter with timeout, textbox for inputting filter.
    var cvFiltering = null,
        filteringGrid = null,
        toFilter,
        filteringInput = null;

    // define the filter function for the collection view.
    function filterFunction(item) {
        var filter = filteringInput.value.toLowerCase();
        if (!filter) {
            return true;
        }

        return item.CustomerID.toLowerCase().indexOf(filter) > -1;
    };

    // apply filter (applied on a 500 ms timeOut)
    function filterGrid() {
        if (toFilter) {
            clearTimeout(toFilter);
        }

        toFilter = setTimeout(function () {
            toFilter = null;
            if (cvFiltering.filter === filterFunction) {
                cvFiltering.refresh();
            }
            else {
                cvFiltering.filter = filterFunction;
            }
        }, 500);
    };
```

```
</script>
```

Tracking Changes

The **CollectionView** class can keep track of changes made to the data. It is useful in situations where you must submit changes to the server. To turn on change tracking, set the **TrackChanges** property to true. Once you do that, the **CollectionView** starts tracking the changes made to the data and exposes them using the following:

- **ItemsEdited**: This list contains items that are edited using the **BeginEdit** and **CommitEdit** methods.
- **ItemsAdded**: This list contains items that are added using the **AddNew** and **CommitNew** methods.
- **ItemsRemoved**: This list contains items that are removed using the **Remove** method.

> Make sure that the **DisableServerRead** property of ItemSource is set to **True** if filtering, paging, sorting is to be performed on data available at client side only.

The example uses **C1NWind** datasource, for which the configuration in the Quick Start application. The following image shows how the FlexGrid appears after the **TrackChanges** property is set to **true**.

Change the data here

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	9	Desserts	Cookies, ice-creams, pastries, cakes
	10	Breads	
*			

See the changes here

Items edited:

	CategoryID	CategoryName	Description
	9	Desserts	Cookies, ice-creams, pastries, cakes

Items added:

	CategoryID	CategoryName	Description
	10	Breads	

Items removed:

	CategoryID	CategoryName	Description
	8	Seafood	Seaweed and fish

The following code example demonstrates how to initialize a FlexGrid, and keep a track of changes by using **TrackChanges** property of the CollectionView.

In Code

TrackChangesController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();
public ActionResult Index()
{
    return View(db);
```

```
}
```

TrackChanges.cshtml

HTML Helpers

Razor copyCode

```
<h5>Change the data here</h5>
@(Html.C1().FlexGrid().Id("tcMainGrid").AutoGenerateColumns(false)
    .AllowAddNew(true).AllowDelete(true)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID"))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("*")))
    .Bind(ib =>
        ib.Bind(Model.Categories).DisableServerRead(true)
        )
)
<h5>See the changes here</h5>
<h6>Items edited:</h6>
@(Html.C1().FlexGrid().Id("tcEditedGrid").AutoGenerateColumns(false)
    .IsReadOnly(true).Height(100)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID").IsReadOnly(true))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("*")))
)
<h6>Items added:</h6>
@(Html.C1().FlexGrid().Id("tcAddedGrid").AutoGenerateColumns(false)
    .IsReadOnly(true).Height(100)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID").IsReadOnly(true))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("*")))
)
<h6>Items removed:</h6>
@(Html.C1().FlexGrid().Id("tcRemovedGrid").AutoGenerateColumns(false)
    .IsReadOnly(true).Height(100)
    .Columns(columns => columns
        .Add(c => c.Binding("CategoryID").IsReadOnly(true))
        .Add(c => c.Binding("CategoryName"))
        .Add(c => c.Binding("Description").Width("*")))
)
```

JS

Script copyCode

```
<script>
```

```
    $(document).ready(function () {
        //Tracking changes
        tcMainGrid = wijmo.Control.getControl('#tcMainGrid');// the flexGrid to edit
the data
        tcEditedGrid = wijmo.Control.getControl('#tcEditedGrid'); // the flexGrid to
record the edited items
        tcAddedGrid = wijmo.Control.getControl('#tcAddedGrid'); // the flexGrid to
record the added items
        tcRemovedGrid = wijmo.Control.getControl('#tcRemovedGrid'); // the flexGrid
to record the removed items
        cvTrackingChanges = tcMainGrid.itemsSource;

        tcEditedGrid.itemsSource = cvTrackingChanges.itemsEdited;
        tcAddedGrid.itemsSource = cvTrackingChanges.itemsAdded;
        tcRemovedGrid.itemsSource = cvTrackingChanges.itemsRemoved;

        // track changes of the collectionview
        cvTrackingChanges.trackChanges = true;
    });

    //Tracking changes
    var tcMainGrid = null,
        tcEditedGrid = null,
        tcAddedGrid = null,
        tcRemovedGrid = null,
        cvTrackingChanges = null;
</script>
```

DateTime Processing

This topic demonstrates how to keep a DateTime property in a UTC format on both server and client when using FlexGrid with Ajax Binding or Editing. To achieve this, we first need to understand how the DateTime values are processed at client side and server side.

On the Server side, every time you use the **DateTime** class, you need to specify the **Kind** property, which indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither. However, when you are working with DateTime object at the Client side, the browser implicitly convert all dates according to the local time when the date is parsed from a **Number** to **Date** object.

For example, when you create a DateTime instance on server side, such as **new DateTime(2017, 0, 25, 7, 0, 0, DateTimeKind.Utc);**. Once you transfer this value from server to client, browsers on different machines which use different TimeZone system settings show different string representations.

To keep time in the UTC format, you need to apply an explicit transformation to the dates on both client and server.

Server

In case of server, you need to convert all the DateTime objects to Unspecified and, then convert them back when necessary using CREATE, UPDATE or DELETE operations. You need to implement the following two steps.

1. Convert to Unspecified format during reading data.
2. Convert Unspecified format back during CREATE, UPDATE and DELETE operations.

To understand the transformation, we will take an example of the FlexGrid control, which include **DateTime** fields with different formats, such as: Utc, Local and Unspecified. Bind the grid with these data values.

Create a new Model

1. Add a new class to the folder **Models** (for example: `DatesData.cs`). For more information on how to add a new model, see Adding Controls.
2. Replace the following code in the new model to define the classes that serve as a datasource for the FlexGrid control.

DatesData.cs

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

namespace DateTimeFields.Models
{
    public class DatesData
    {

        /// The primary key.
        public int Id { get; set; }

        /// A DateTime field which Kind is Utc.
        public DateTime UtcDateTime { get; set; }

        /// A DateTime field which Kind is Unspecified.
        public DateTime UnspecifiedDateTime { get; set; }

        /// A DateTime field which Kind is Local.
        public DateTime LocalDateTime { get; set; }

        /// Get the data.
        /// <param name="total"></param>
        /// <returns></returns>
        public static IEnumerable<DatesData> GetData(int total)
        {
            var rand = new Random(0);
            var dt = DateTime.Now;
            var list = Enumerable.Range(0, total).Select(i =>
            {
                return new DatesData
                {
                    Id = i + 1,
                    UtcDateTime = new DateTime(dt.Year, i % 12 + 1, 25, 7, 0, 0,
DateTimeKind.Utc),
                    UnspecifiedDateTime = new DateTime(dt.Year, i % 12 + 1, 25,
7, 0, 0, DateTimeKind.Unspecified),
                    LocalDateTime = new DateTime(dt.Year, i % 12 + 1, 25, 7, 0,
0, DateTimeKind.Local)
                };
            });
```

```
            return list;
        }
    }
}
```

Controller

In Code - HomeController.cs

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serialization;
using <ApplicationName>.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;

namespace DateTimeFields.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        private static List<DatesData> convertedData = DatesData.GetData(3).ToList();
        private static List<DatesData>
ConvertToUnspecifiedData(IEnumerable<DatesData> sourceData)
        {
            return sourceData.Select(item => new DatesData
            {
                Id = item.Id,
                UnspecifiedDateTime = item.UnspecifiedDateTime,
                UtcDateTime = new DateTime(item.UtcDateTime.Ticks),
                LocalDateTime = new DateTime(item.LocalDateTime.Ticks)
            }).ToList();
        }

        private static void
ConvertUpspecifiedBack(CollectionViewEditRequest<DatesData> requestData)
        {
            // Convert Unspecified DateTime back.
            foreach (var item in requestData.OperatingItems)
            {
                item.LocalDateTime = new DateTime(item.LocalDateTime.Ticks,
DateTimeKind.Local);
                item.UtcDateTime = new DateTime(item.UtcDateTime.Ticks,
DateTimeKind.Utc);
```

```
            }
        }
        public ActionResult Converted_ReadDatesData([C1JsonRequest]
CollectionViewRequest<DatesData> requestData)
        {
            return this.C1Json(CollectionViewHelper.Read(requestData,
ConvertToUnspecifiedData(convertedData)));
        }
        public ActionResult
Converted_UpdateDatesData([C1JsonRequest]CollectionViewEditRequest<DatesData>
requestData)
        {
            ConvertUpspecifiedBack(requestData);
            return Update(requestData, convertedData, ConvertToUnspecifiedData);
        }
        public ActionResult
Converted_CreateDatesData([C1JsonRequest]CollectionViewEditRequest<DatesData>
requestData)
        {
            ConvertUpspecifiedBack(requestData);
            return Create(requestData, convertedData, ConvertToUnspecifiedData);
        }
        public ActionResult
Converted_DeleteDatesData([C1JsonRequest]CollectionViewEditRequest<DatesData>
requestData)
        {
            ConvertUpspecifiedBack(requestData);
            return Delete(requestData, convertedData, ConvertToUnspecifiedData);
        }

        public ActionResult Update(CollectionViewEditRequest<DatesData> requestData,
List<DatesData> sourceData, Func<IEnumerable<DatesData>, List<DatesData>> converter =
null)
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                var error = string.Empty;
                var success = true;
                try
                {
                    var index = sourceData.FindIndex(u => u.Id == item.Id);
                    sourceData.RemoveAt(index);
                    sourceData.Insert(index, item);
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }
                return new CollectionViewItemResult<DatesData>
                {
```

```
                    Error = error,
                    Success = success,
                    Data = item
                };
            }, () => converter != null ? converter(sourceData) : sourceData));
        }

        public ActionResult Create(CollectionViewEditRequest<DatesData> requestData,
List<DatesData> sourceData, Func<IEnumerable<DatesData>, List<DatesData>> converter =
null)
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                var error = string.Empty;
                var success = true;
                try
                {
                    sourceData.Add(item);
                    item.Id = sourceData.Max(u => u.Id) + 1;
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }
                return new CollectionViewItemResult<DatesData>
                {
                    Error = error,
                    Success = success,
                    Data = item
                };
            }, () => converter != null ? converter(sourceData) : sourceData));
        }

        public ActionResult Delete(CollectionViewEditRequest<DatesData> requestData,
List<DatesData> sourceData, Func<IEnumerable<DatesData>, List<DatesData>> converter =
null)
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                var error = string.Empty;
                var success = true;
                try
                {
                    var index = sourceData.FindIndex(u => u.Id == item.Id);
                    sourceData.RemoveAt(index);
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
```

```
                }
                return new CollectionViewItemResult<DatesData>
                {
                    Error = error,
                    Success = success,
                    Data = item
                };
            }, () => converter != null ? converter(sourceData) : sourceData));
        }
    }
}
```

Add View

In Code - Index.cshtml

DatesData.cs

```
@(Html.C1().FlexGrid()
    .Id("convertedGrid")
    .AllowAddNew(true)
    .AllowDelete(true)
    .AutoGenerateColumns(false)
    .Bind(cvb => cvb.Bind(Url.Action("Converted_ReadDatesData"))
                    .Create(Url.Action("Converted_CreateDatesData"))
                    .Update(Url.Action("Converted_UpdateDatesData"))
                    .Delete(Url.Action("Converted_DeleteDatesData")))
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("Id").IsReadOnly(true).Visible(false));
        columns.Add(column => column.Binding("UtcDateTime").Format("dd/MM/yyyy
HH:mm:ss").Width("*"));
        columns.Add(column => column.Binding("LocalDateTime").Format("dd/MM/yyyy
HH:mm:ss").Width("*"));
        columns.Add(column =>
column.Binding("UnspecifiedDateTime").Format("dd/MM/yyyy HH:mm:ss").Width("*"));
    })
    .Filterable(f => f.DefaultFilterType(FilterType.Both)))
```

Client

In case of Client, C1 MVC components provide two client events **OnClientReponseTextParsing** and **OnClientRequestDataStringifying** to make explicit transformations.

- **OnClientReponseTextParsing** - When the data is retrieved on the client from the server, all data is serialized into a JSON text and the text will be retrieved on client. On the client, the text will be parsed into JavaScript objects. You need to perform transformation during parsing. It supports the following event arguments.
  - Key - Name of the item text to be parsed.
  - Value: The text of the item to be parsed.
  - Result: Specifies its value with what you want the text to be parsed to.
  - Cancel: If you don't want the default parsing, specify its value to true.
- **OnClientRequestDataStringifying** - When the data is sent back to the server from the client. The data will be

serialized into a text and sent to the server. On the server the text will be de-serialized into an object. The JavaScript Date object is always to be serialized into a text with UTC format. If we don't customize the serialization, a DateTime object in UTC format is retrieved after de-serialization. This is not the correct way of implementing, you need to do transformation during parsing, using the OnClientRequestDataStringifying event.

- Key - Name of the item text to be serialized.
- Value: Name of the item to be serialized.
- Result: Specifies the text what you want the object to be serialized.
- Cancel: If you don't want the default serialization, specify it to true.

To understand the transformation, we will take an example of the FlexGrid control, which include **DateTime** fields with different formats, such as: Utc, Local and Unspecified. Bind the grid with these data values

Add a new Model

1. Add a new data class to the **Models** folder**.** The example uses the same model **DatesData.cs** created in the Server section.

Add a new Controller

In Code - HomeController.cs

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serialization;
using <ApplicationName>.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;

namespace DateTimeFields.Controllers {
 public class HomeController: Controller {
  public ActionResult Index() {
   return View();
  }

  private static List <DatesData> customData = DatesData.GetData(3).ToList();
  public ActionResult Custom_ReadDatesData([C1JsonRequest] CollectionViewRequest
<DatesData> requestData) {
   return this.C1Json(CollectionViewHelper.Read(requestData, customData));
  }

  public ActionResult Custom_UpdateDatesData([C1JsonRequest]
CollectionViewEditRequest <DatesData> requestData) {
   return Update(requestData, customData);
  }

  public ActionResult Custom_CreateDatesData([C1JsonRequest]
CollectionViewEditRequest <DatesData> requestData) {
   return Create(requestData, customData);
  }
```

```
  public ActionResult Custom_DeleteDatesData([C1JsonRequest]
CollectionViewEditRequest <DatesData> requestData) {
   return Delete(requestData, customData);
  }

  public ActionResult Update(CollectionViewEditRequest <DatesData> requestData, List
<DatesData> sourceData, Func <IEnumerable <DatesData> , List <DatesData>> converter =
null) {
   return this.C1Json(CollectionViewHelper.Edit(requestData, item => {
    var error = string.Empty;
    var success = true;
    try {
     var index = sourceData.FindIndex(u => u.Id == item.Id);
     sourceData.RemoveAt(index);
     sourceData.Insert(index, item);
    } catch (Exception e) {
     error = e.Message;
     success = false;
    }
    return new CollectionViewItemResult <DatesData> {
     Error = error,
     Success = success,
     Data = item
    };
   }, () => converter != null ? converter(sourceData) : sourceData));
  }

  public ActionResult Create(CollectionViewEditRequest <DatesData> requestData, List
<DatesData> sourceData, Func <IEnumerable <DatesData> , List <DatesData>> converter =
null) {
   return this.C1Json(CollectionViewHelper.Edit(requestData, item => {
    var error = string.Empty;
    var success = true;
    try {
     sourceData.Add(item);
     item.Id = sourceData.Max(u => u.Id) + 1;
    } catch (Exception e) {
     error = e.Message;
     success = false;
    }
    return new CollectionViewItemResult <DatesData> {
     Error = error,
     Success = success,
     Data = item
    };
   }, () => converter != null ? converter(sourceData) : sourceData));
  }

  public ActionResult Delete(CollectionViewEditRequest <DatesData> requestData, List
<DatesData> sourceData, Func <IEnumerable <DatesData> , List <DatesData>> converter =
null) {
```

```
   return this.C1Json(CollectionViewHelper.Edit(requestData, item => {
    var error = string.Empty;
    var success = true;
    try {
     var index = sourceData.FindIndex(u => u.Id == item.Id);
     sourceData.RemoveAt(index);
    } catch (Exception e) {
     error = e.Message;
     success = false;
    }
    return new CollectionViewItemResult <DatesData> {
     Error = error,
     Success = success,
     Data = item
    };
   }, () => converter != null ? converter(sourceData) : sourceData));
  }
 }
}
```

Add a JavaScript file.

You need to add a JavaScript file (For example: app.js) to define the **responseTextParsing** and **requestDataStringfying** function.

app.js

```
// The RegExp object which is used to tell a DateTime text.
var dateJsonRegx = /^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d*)?(Z|[\+\-
]\d{2}:\d{2}|)$/;
function reponseTextParsing(sender, args) {
    var dateText = args.value;
    // check whether it is a valid DateTime text.
    var matched = dateJsonRegx.exec(dateText);
    if (!matched) {
        return;
    }
    var timeZoneText = matched[2];
    var dateKind = getDateKind(timeZoneText);
    // only customize the parsing for the Date object in Utc or Local format.
    if (dateKind == c1.mvc.DateKind.Unspecified) {
        return;
    }
    if (typeof dateText === 'string' && matched) {
        var index = dateText.indexOf(timeZoneText);
        // remove the time zone text and create a Date object.
        var date = new Date(dateText.substr(0, index));
        // Don't forget to set the dateKind for the Date object parsed.
        // It could be used in OnClientRequestDataStringifying.
        date.dateKind = dateKind;
        args.result = date;
        args.cancel = true;
```

```
    }
}

function getDateKind(timeZoneText) {
    if (!timeZoneText) {
        return c1.mvc.DateKind.Unspecified;
    }

    if (timeZoneText.toLowerCase() === 'z') {
        return c1.mvc.DateKind.Utc;
    }

    return c1.mvc.DateKind.Local;
}

function requestDataStringifying(sender, args) {
    if (args.value instanceof Date || args.parent[args.key] instanceof Date) {
        var date = args.value instanceof Date ? args.value : args.parent[args.key];
        // only customize the serialization for the Date object in Utc format.
        if (!date.dateKind || date.dateKind == c1.mvc.DateKind.Unspecified) {
            return;
        }

        args.result = c1.mvc.Utils.formatNumber(date.getFullYear(), 4) + '-' +
                c1.mvc.Utils.formatNumber(date.getMonth() + 1, 2) + '-' +
                c1.mvc.Utils.formatNumber(date.getDate(), 2) + 'T' +
                c1.mvc.Utils.formatNumber(date.getHours(), 2) + ':' +
                c1.mvc.Utils.formatNumber(date.getMinutes(), 2) + ':' +
                c1.mvc.Utils.formatNumber(date.getSeconds(), 2) + '.' +
                c1.mvc.Utils.formatNumber(date.getMilliseconds(), 3)
                + (date.dateKind == c1.mvc.DateKind.Utc ? 'Z' :
getLocalTimeZoneText());
        args.cancel = true;
    }
}

function getLocalTimeZoneText() {
    var date = new Date();
    var timeoffset = date.getTimezoneOffset();
    var result = '';
    if (timeoffset > 0) {
        result += '-';
    } else {
        result += '+';
        timeoffset *= -1;
    }
    var hour = Math.floor(timeoffset / 60);
    result += formatNumber(hour, 2);
    result += ":";
    result += formatNumber(timeoffset - hour * 60, 2);
    return result;
```

```
}

function formatNumber(n, k) {
    // Format integers to have at least k digits.
    var text = n.toString();
    while (text.length < k) {
        text = '0' + text;
    }
    return text;
}
```

Add View

In Code - Index.cshtml

DatesData.cs

```
@(Html.C1().FlexGrid()
      .Id("customGrid")
      .AllowAddNew(true)
      .AllowDelete(true)
      .AutoGenerateColumns(false)
      .Bind(cvb => cvb.Bind(Url.Action("Custom_ReadDatesData"))
                  .Create(Url.Action("Custom_CreateDatesData"))
                  .Update(Url.Action("Custom_UpdateDatesData"))
                  .Delete(Url.Action("Custom_DeleteDatesData"))
                  .OnClientReponseTextParsing("reponseTextParsing")
                  .OnClientRequestDataStringifying("requestDataStringifying"))
      .Columns(columns =>
          {
            columns.Add(column =>
column.Binding("Id").IsReadOnly(true).Visible(false));
            columns.Add(column => column.Binding("UtcDateTime").Format("dd/MM/yyyy
HH:mm:ss").Width("*"));
            columns.Add(column => column.Binding("LocalDateTime").Format("dd/MM/yyyy
HH:mm:ss").Width("*"));
            columns.Add(column =>
column.Binding("UnspecifiedDateTime").Format("dd/MM/yyyy HH:mm:ss").Width("*"));
          })
       .Filterable(f => f.DefaultFilterType(FilterType.Both)))
```

CollectionView ASP.NET Core Tags

CollectionView supports the following ASP.NET Core Tags:

CollectionView Class

- c1-items-source
- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url

- disable-server-read
- group-by
- c1-property-group-description
- id
- initial-items-count
- order-by
- page-index
- page-size
- query-data
- read-action-url
- c1-sort-description
- source-collection
- update-action-url
- client-converter
- property-name

Excel

The **Excel** control enables you to use a single command to load or save a workbook – you don't even need to have Microsoft Excel installed! Create or load XLS files. Excel control supports OpenXml format, which allows you to save smaller, compressed XLSX files.

Excel for .NET allows you to create Microsoft Excel files directly from your application's code. The possibilities are endless with an easy-to-manipulate Excel object model exposed in your code. You can create Excel files from data that doesn't directly support Excel exportation like any grid, schedule or chart.

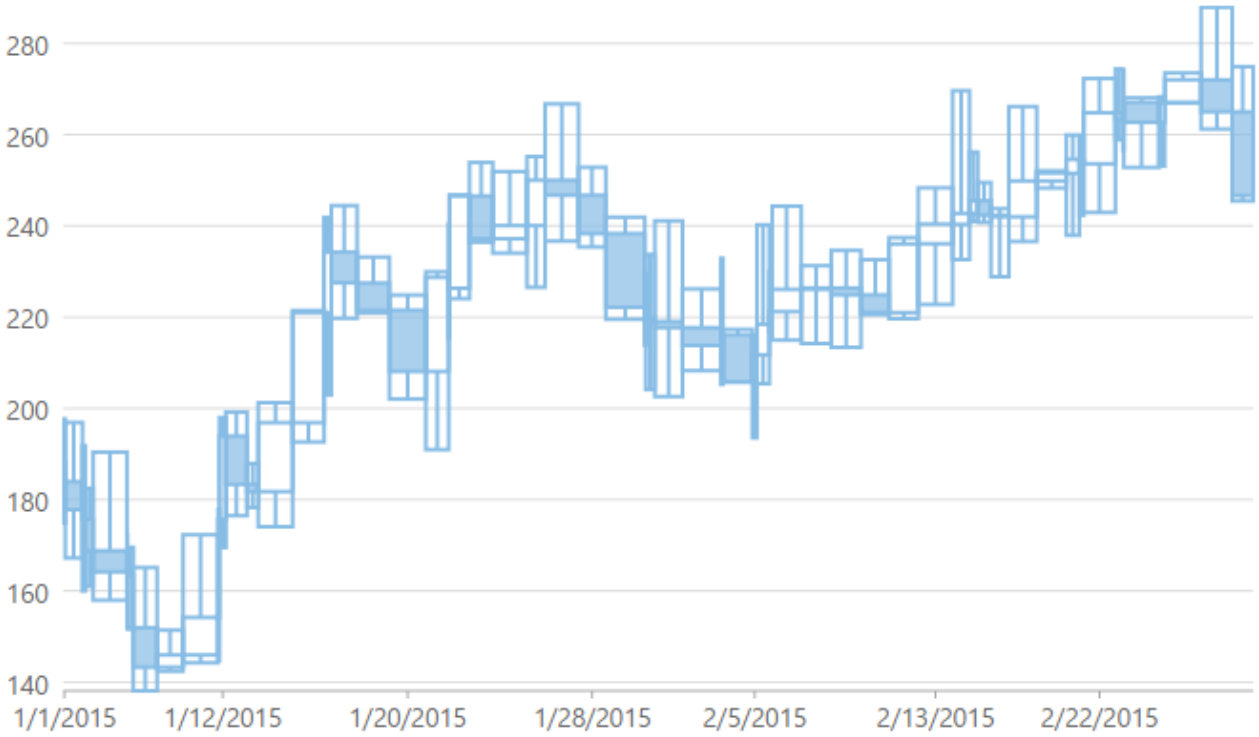
Key Features

- **Save or Load a Workbook with One Command:** Use a single command to load or save a workbook and manipulate sheets.
- **No Need for Microsoft Excel:** Read and write XLS (Excel 97 and later) and XLSX (OpenXml format) files without a dependency on Microsoft Excel.
- **Total Microsoft Excel Compatibility:** Just set one property, C1XLBook.CompatibilityMode, to specify which version of Microsoft Excel you want your workbooks to be compatible with. There are three options: Excel2003 (allows up to 64k rows and 256 columns), Excel2007 (up to 1 million rows and 18k columns), and NoLimits.
- **Format Data:** Format cells to ensure that end-users enter correct data. The format associated with each cell is as easy to access as the data stored in the cell.
- **Cell Formulas:** The XLCell.Formula property allows you to specify a formula for the cell.
- **Grouping and Subtotals:** Calculate subtotals for rows and columns. Declare outline level grouping in code to best display totals and subtotals.

For more information, refer to Excel for .NET.

Financial Charts

ComponentOne's MVC Edition offers **Financial Charts**, the newest range of charts that lets you create advanced, stock trending visualizations. Use these charts to analyze data trends with trend lines, filters, range selectors, and annotations with minimal coding. With various built-in features and different chart types, **FinancialChart** control itself is optimized and targeted for the finance industry.



Key Features

- **Chart Type**: Change a line chart to a bar chart or any other chart type by setting a single property. Provides fifteen different chart types to choose from.
- **Multiple Areas**: Allows you to add multiple series and areas on a single chart without data overlapping.
- **Markers**: Displays a text area on the FinancialChart.
- **Range Selector**: Adjusts the FinancialChart's visible range of data at runtime.
- **Trend Lines**: Visualizes trends in data and analyze the problems of prediction.
- **Tooltips**: Display chart values using tooltips.
- **Header and Footer**: Use simple properties to set a title and footer text.
- **Legend**: Change position of the legend as needed.
- **Annotations**: Mark important events or news attached to a specific data point on financial charts.
- **Fibonacci Series**: Calculate and mark various alert trends on financial charts for stock analysis.
- **Moving Average**: Analyze data points through a series of averages of different subsets of entire data set.
- **Function Series**: Create analytical financial charts using different formulae for calculating y.
- **Parametric Series**: Create analytical financial charts using different formulae for calculating x and y.
- **Hit Test**: Determine coordinates and index of points hovered on chart.
- **Indicators**: Analyze and predict trends in price and volume momentum of trading instruments.

Quick Start: Add Data to FinancialChart

This section describes how to add a FinancialChart control to your ASP.NET MVC and ASP.NET Core MVC web application, and add data to it.

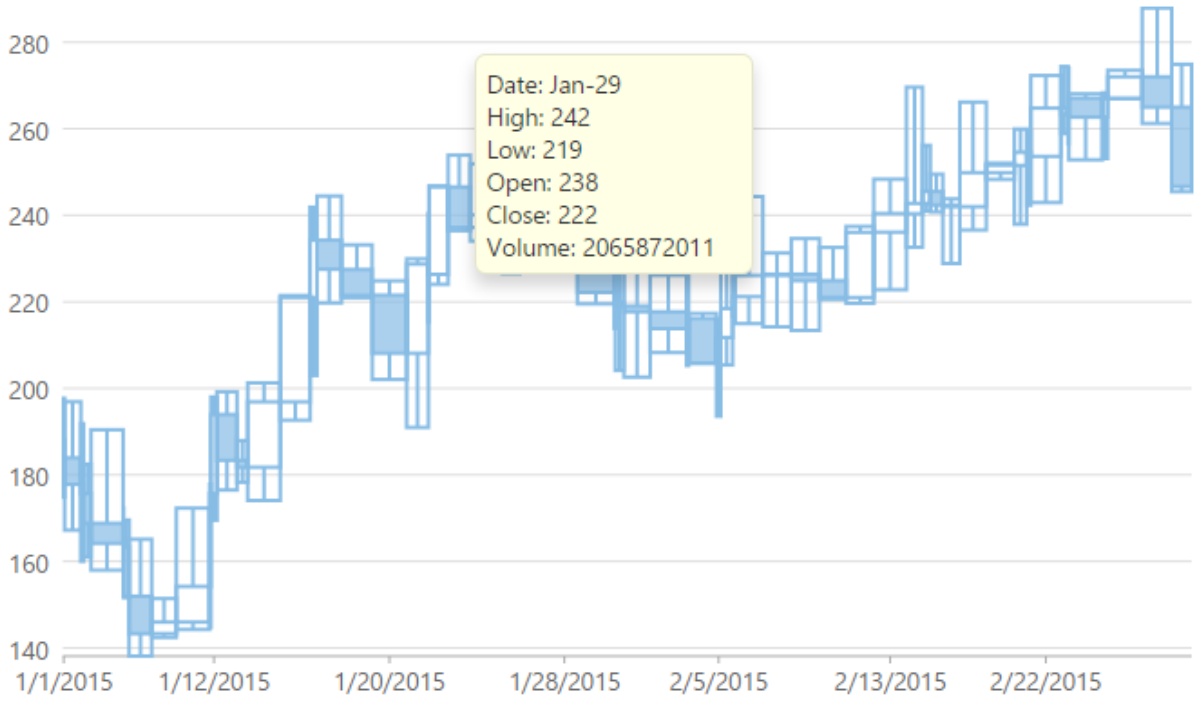
This topic comprises of five steps:

- **Step 1: License your application**
- **Step 2: Add the relevant references to your application**
- **Step 3: Configure the application to use FinancialChart**
- **Step 4: Register Resources**
- **Step 5: Create a Datasource for FinancialChart**

- **Step 6: Add a FinancialChart control**
- **Step 7: Build and Run the Project**

> **Note:** The **ComponentOne template** for ASP.NET MVC Edition automatically registers the required resources, and adds the relevant references and packages to your application. Therefore, you need not follow the Steps 1 to 3 above if your application is created using ComponentOne template.

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



Step 1: License your application

1. In the **Solution Explorer**, double-click the project name (for example, **MVCFinancialChart**) and expand node **Properties.**
2. Double-click the **licenses.licx** file to open it.
3. In the **licenses.licx** file, add the following:

licenses.licx

```
C1.Web.Mvc.LicenseDetector, C1.Web.Mvc
C1.Web.Mvc.Finance.LicenseDetector, C1.Web.Mvc.Finance
```

For more information on how to add license to your application, refer to Licensing.

Back to Top

Step 2: Add the relevant references to your application

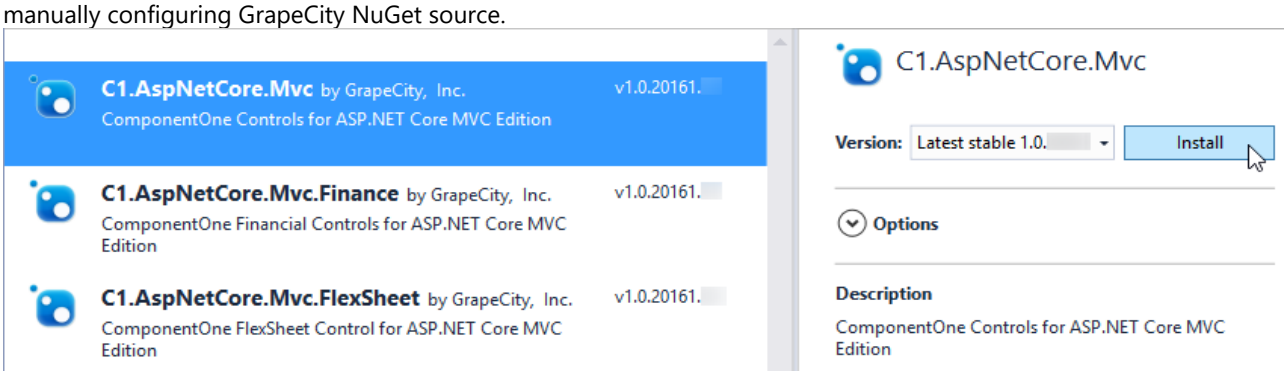
ASP.NET

Complete the following steps to add the ASP.NET MVC Edition references and Financial Chart references to your project.

1. In the **Solution Explorer**, right click **References** and select **Add Reference.**
2. Browse to the location- **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin.**
3. Select **C1.Web.Mvc.dll** and **C1.Web.Mvc.Finance.dll**, and click **Add**.
4. Set the **Copy Local** property of the **C1.Web.Mvc.dll** and **C1.Web.Mvc.Finance.dll** to **True**.

ASP.NET Core

1. Add the ASP.NET MVC Edition references to the project. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**. In **NuGet Package Manager**, select **GrapeCity** as the Package source. Search for **C1.AspNetCore.Mvc** package, and click **Install**. Refer to Configuring NuGet Package Sources for information on manually configuring GrapeCity NuGet source.



> **Note:** "C1.AspNetCore.Mvc" gets added under the "dependencies" within project.json file of your project once you restore the packages.

2. To work with **Financial Chart** control in your application, add **C1.AspNetCore.Mvc.Finance** package. Once you restore the packages, "C1.AspNetCore.Mvc.Finance" gets added under the "dependencies" in project.json file.

Back to Top

Step 3: Configure the application to use FinancialChart

ASP.NET

1. From the **Solution Explorer**, expand the folder **Views** and double click the `web.config` file to open it.
2. Add the following markups in <namespaces></namespaces> tags, within the `<system.web.webPages.razor>` `</system.web.webPages.razor>` tags.

HTML

```
<add namespace="C1.Web.Mvc" />
<add namespace="C1.Web.Mvc.Finance" />
<add namespace="C1.Web.Mvc.Finance.Fluent" />
```

ASP.NET Core

1. From the **Solution Explorer**, expand the folder **Views** and double click the _ViewImports.cshtml file to open it.
2. Add the following references to work with Financial Chart control in your ASP.NET Core application,

_ViewImports

```
@addTagHelper *, C1.AspNetCore.Mvc

@addTagHelper *, C1.AspNetCore.Mvc.Finance
```

Back to Top

Step 4: Register Resources

Complete the following steps to register the required resources for using ASP.NET MVC FinancialChart control:

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click `_Layout.cshtml` to open it.
3. Add the following code between the `<head></head>` tags.

Using HTML Helpers

_Layout.cshtml

```
@Html.C1().Styles()
@Html.C1().Scripts().Basic().Finance()
```

Using Tag Helpers

_Layout.cshtml

```
<c1-styles/>
<c1-scripts>
        <c1-finance-scripts />
</c1-scripts>
```

For more information on how to register resources for **FinancialChart**, refer to Registering Resources.

Back to Top

Step 5: Create a Datasource for FinancialChart

1. Add a new class to the folder **Models** (for example: `FinanceData.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes that serve as a datasource for the FinancialChart control.

C# copyCode

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVCFinancialChart.Models
{
    public class FinanceData
    {
        public DateTime X { get; set; }
        public double High { get; set; }
        public double Low { get; set; }
        public double Open { get; set; }
        public double Close { get; set; }
        public double Volume { get; set; }
    }

}
```

Back to Top

Step 6: Add a FinancialChart control

Complete the following steps to initialize a FinancialChart control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
```

5. Replace the method `Index()` with the following method.

C# copyCode

```
public ActionResult Index()
{

    var model = GenerateFinanceData();
    return View(model);
}

private List<FinanceData> GenerateFinanceData(int count = 60)
{
    List<FinanceData> financeDatas = new List<FinanceData>() { };

    DateTime startTime = new DateTime(2015, 1, 1);
    var rand = new Random();
    double high, low, open, close, volume;
    for (int i = 0; i < count; i++)
    {
        DateTime dt = startTime.AddDays(i);

        if (i > 0)
            open = financeDatas[i - 1].Close;
        else
            open = 188;

        high = open + rand.NextDouble() * 30;
        low = open - rand.NextDouble() * 20;

        close = low + rand.NextDouble() * (high - low);
        volume = rand.Next();

        financeDatas.Add(new FinanceData { X = dt, High = high, Low = low, Open =
open, Close = close, Volume = volume });
    }

    return financeDatas;
}
```

Add a View for the Controller

1. Within the Controller, which was added in the above step, place the cursor inside the method `Index()`.
2. Right click and select **Add View**. The **Add View** dialog appears.
3. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
4. Click **Add**. A view is added for the controller.
5. Instantiate a FinancialChart control in the view QuickStart as shown below.

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
                              'High: ' + item.High.toFixed() + '<br/>' +
                              'Low: ' + item.Low.toFixed() + '<br/>' +
                              'Open: ' + item.Open.toFixed() + '<br/>' +
                              'Close: ' + item.Close.toFixed() + '<br/>' +
                              'Volume: ' + item.Volume.toFixed();
        }
    };
</script>

@*Initialize FinancialChart control*@
@(Html.C1().FinancialChart()
.Bind(Model)
        //Set the height and width of the chart
.Height(400)
.Width(700)
.BindingX("X")
        //Set ChartType of the chart
.ChartType(C1.Web.Mvc.Finance.ChartType.ArmsCandleVolume)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close,Volume");
    })
.Tooltip(t => t.Content("tooltipContent")))
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Chart;
<script type="text/javascript">
var tooltipContent = function (ht) {
var item = ht.series.collectionView.items[ht.pointIndex];
return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
```

```
                    'High: ' + item.High.toFixed() + '<br/>' +
                    'Low: ' + item.Low.toFixed() + '<br/>' +
                    'Open: ' + item.Open.toFixed() + '<br/>' +
                    'Close: ' + item.Close.toFixed();
};
</script>
<c1-financial-chart binding-x="X" chart-
type="C1.Web.Mvc.chart.Finance.ChartType.HighLowOpenClose">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close"></c1-financial-chart-series>
<c1-flex-chart-tooltip content="tooltipContent"></c1-flex-chart-tooltip>
</c1-financial-chart>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**QuickStart/Index**) in the address bar of the browser to see the view.

Back to Top

Financial Chart Types

You can change the type of the FinancialChart control depending on your requirement. Chart type can be changed by setting the ChartType property of the FinancialChart control. In this case, if multiple series are added to the FinancialChart, all of them are of the same chart type.

Supported Chart types:

Area Chart	**ArmsCandleVolume Chart**	**Candlestick Chart**	**CandleVolume Chart**
Column Chart	**ColumnVolume Chart**	**EquiVolume Chart**	**Heikin-Ashi Chart**
HighLowOpenClose Chart	**Kagi Chart**	**Line Chart**	**Line Break Chart**
Line Symbols Chart	**Renko Chart**	**Scatter chart**	

In Code

HTML Helpers

Razor

```
.ChartType(C1.Web.Mvc.Finance.ChartType.Renko)
```

Tag Helpers

HTML

```
chart-type="C1.Web.Mvc.Finance.ChartType.Renko"
```

Area chart

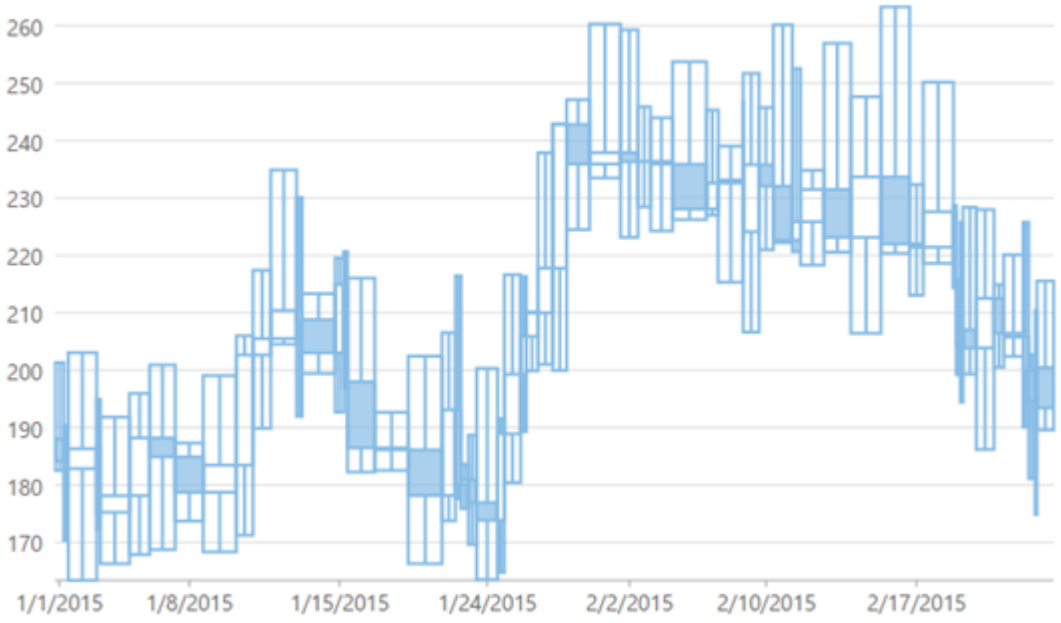
An Area chart draws each series as connected points of data and the area below the connected points is filled with color to denote volume. Each new series is drawn on top of the preceding series. The series can either be drawn independently or stacked.

These charts are commonly used to show trends between associated attributes over time.

Back to Top

ArmsCandleVolume chart

Arms CandleVolume charts are a combination of EquiVolume and CandleVolume charts.



Back to Top

Candlestick chart

A Candlestick chart is a financial chart that shows the opening, closing, high and low prices of a given stock. It is a special type of HiLoOpenClose chart that is used to show the relationship between open and close as well as high and low. Candle chart uses price data (high, low, open, and close values) and it includes a thick candle-like body that uses the color and size of the body to reveal additional information about the relationship between the open and close values. for example:, long transparent candles show buying pressure and long filled candles show selling pressure.

Elements of a Candlestick chart

The Candlestick chart is made up of the following elements: **candle**, **wick**, and **tail**.

- **Candle**: The candle or the body (the solid bar between the opening and closing values) represents the change in stock price from opening to closing.
- **Wick and Tail**: The thin lines, wick and tail, above and below the candle depict the high/low range.
- **Hollow Body:** A hollow candle or transparent candle indicates a rising stock price (close was higher than open). In a hollow candle, the bottom of the body represents the opening price and the top of the body represents the closing price.
- **Filled Body:** A filled candle indicates a falling stock price (open was higher than close). In a filled candle the top of the body represents the opening price and the bottom of the body represents the closing price.

In a Candlestick there are five values for each data point in the series.

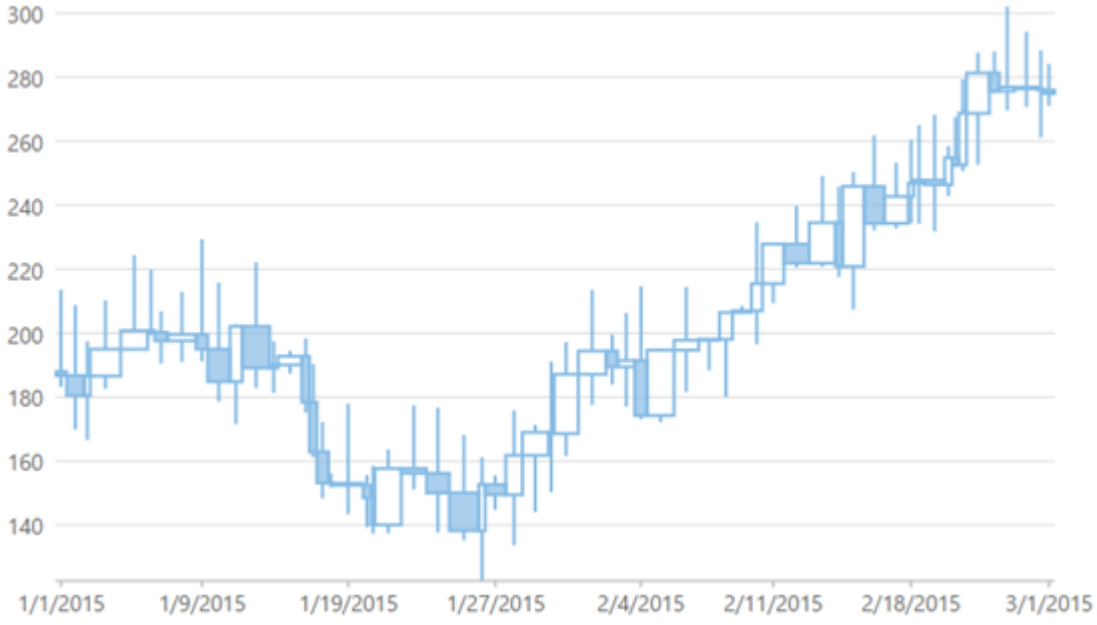
- **x:** Determines the date position along the x axis.
- **high:** Determines the highest price for the day, and plots it as the top of the candle along the y axis.
- **low:** Determines the lowest price for the day, and plots it as the bottom of the candle along the y axis.
- **open:** Determines the opening price for the day.
- **close:** Determines the closing price for the day.

The following image shows a candlestick chart displaying stock prices.

Back to Top

CandleVolume chart

CandleVolume charts are identical to standard Candlestick charts. The only difference is that the Volume determines the width of each bar in CandleVolume charts.



Back to Top

Column chart

A Column chart represents each series in the form of bars of the same color and width, whose length is determined by its value. Each new series is plotted in the form of bars next to the bars of the preceding series. In a column chart, bars are arranged vertically. Column charts can be either grouped or stacked.

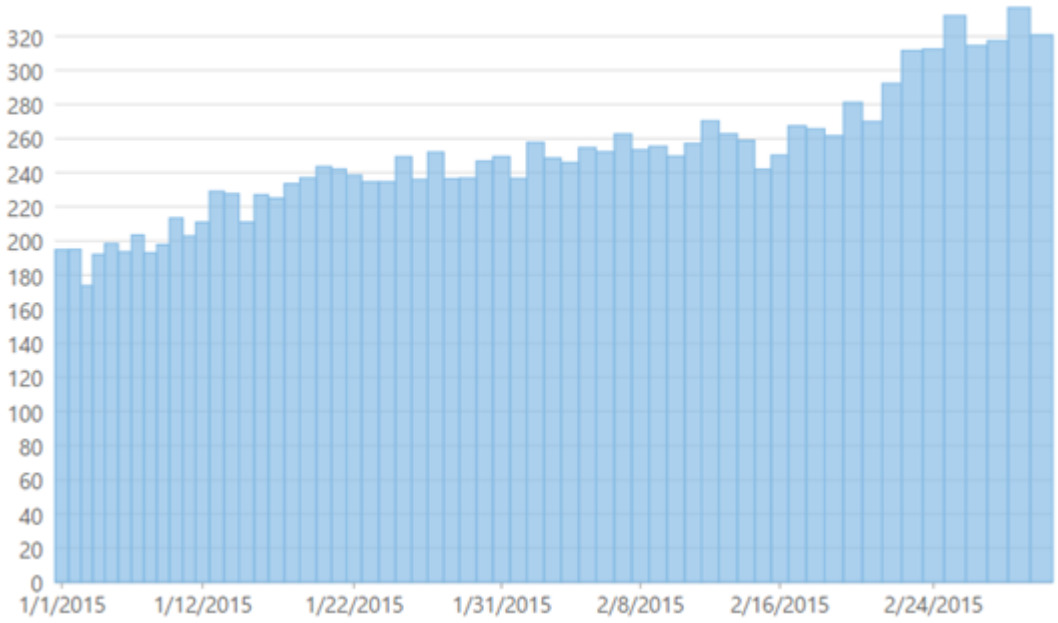
These charts are commonly used to visually represent data that is grouped into discrete categories, for example: age

groups, months, etc.

Back to Top

ColumnVolume chart

ColumnVolume charts are similar to Column charts, but width in these charts are determined by **Volume** value.

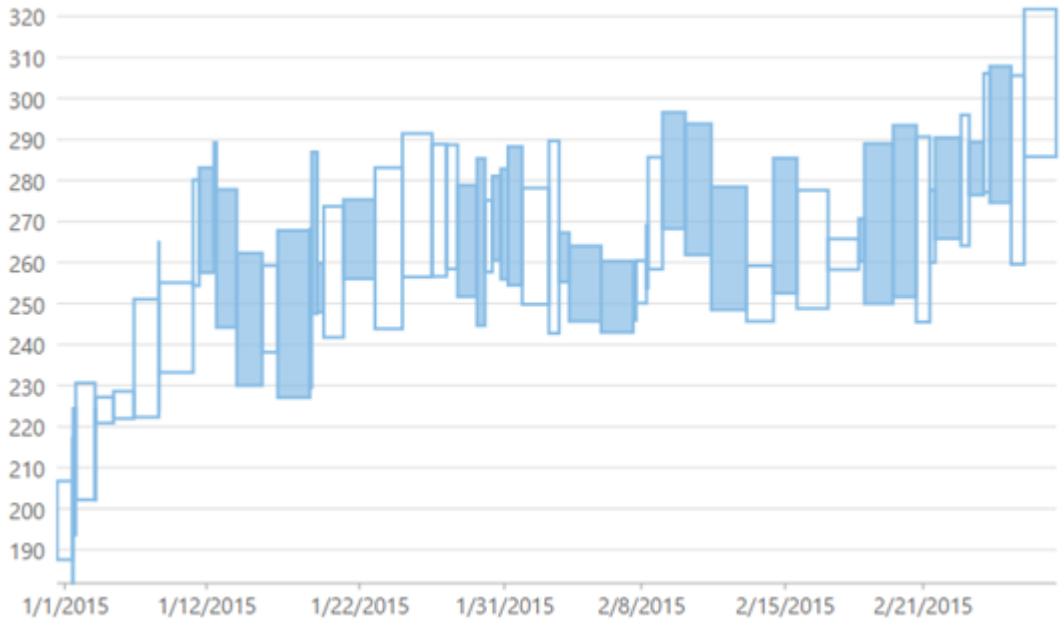


Back to Top

EquiVolume chart

EquiVolume charts are similar to candlestick charts, but the candlesticks in these charts are replaced with rectangular boxes of varying width (and no wicks). An EquiVolume box includes high and low price components with a third dimension, **Volume** that determines the width of each box. Color represents whether the close number is higher or lower than the previous box's close.

To know more about **EquiVolume charts**, read at StockCharts.com.



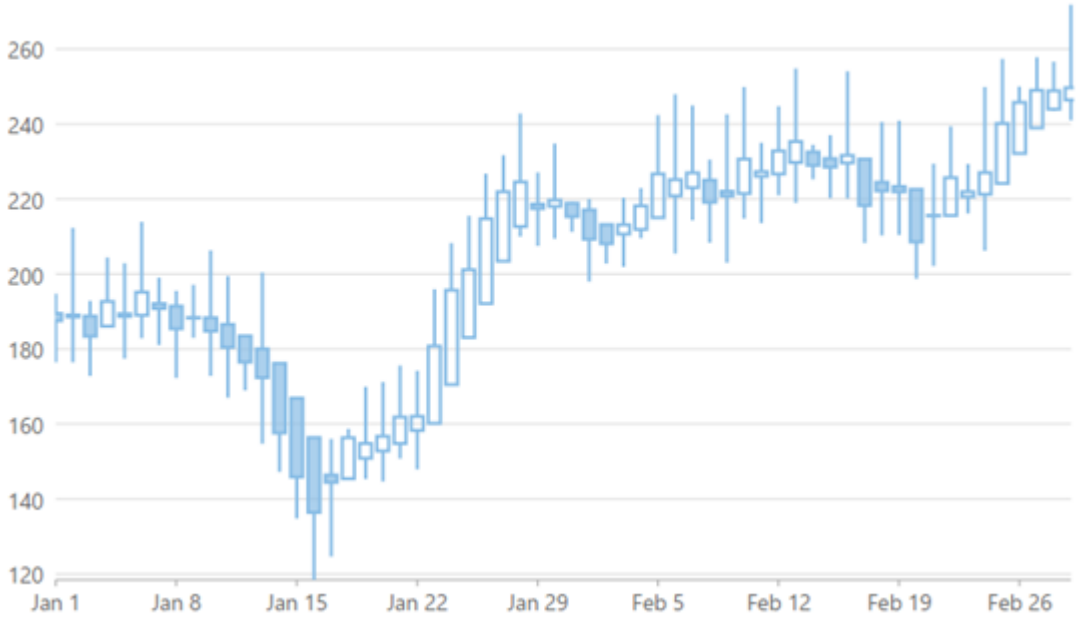
Back to Top

Heikin-Ashi chart

Heikin-Ashi charts are a variation of Japanese candlestick charts designed to remove noise from candlesticks and behave like a moving average. Heikin-Ashi charts are similar to Candlestick charts, but the method to calculate and plot candles on the chart is different for Heikin-Ashi. Each candle in Heikin-Ashi chart uses information presented by the previous candle. These charts are easier to read and interpret data.

These charts can be used to identify trends, potential reversal points, and other technical analysis patterns.

To know more about **Heikin-Ashi charts**, refer to What is Heikin-Ashi and How to Trade With It.



Back to Top

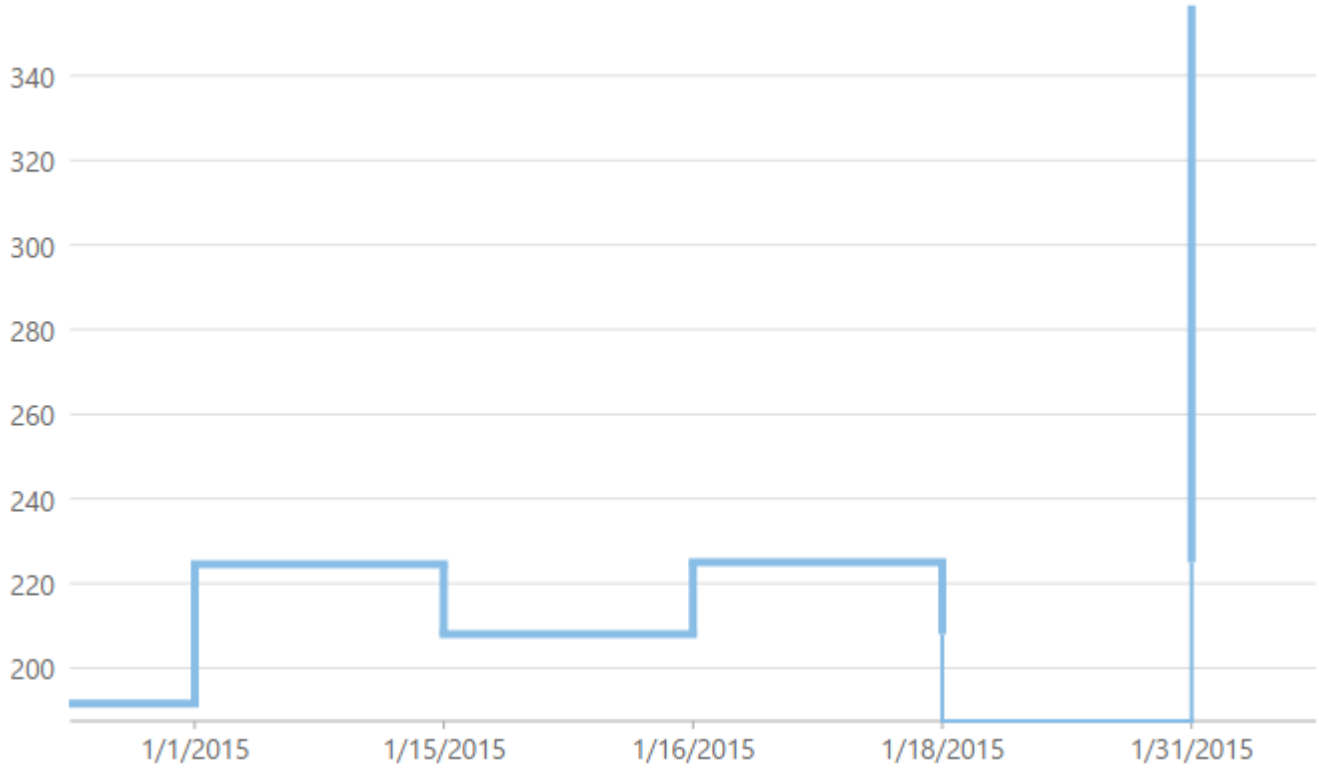
HighLowOpenClose chart

HiLoOpenClose are financial charts that combine four independent values to supply high, low, open, and close data for a point in a series. In addition to showing the high and low value of a stock, the Y2 and Y3 array elements represent the stock's opening and closing price, respectively.

Back to Top

Kagi chart

A Kagi chart displays supply and demand trends using a sequence of linked vertical lines. The thickness and direction of the lines vary depending on the price movement. If closing prices go in the direction of the previous Kagi line, then that Kagi line is extended. However, if the closing price reverses by the preset reversal amount, a new Kagi line is charted in the next column in the opposite direction. Thin lines indicate that the price breaks the previous low (supply) while thick lines indicate that the price breaks the previous high (demand).



Back to Top

Line chart

A Line chart draws each series as connected points of data, similar to area chart except that the area below the connected points is not filled. The series can be drawn independently or stacked.

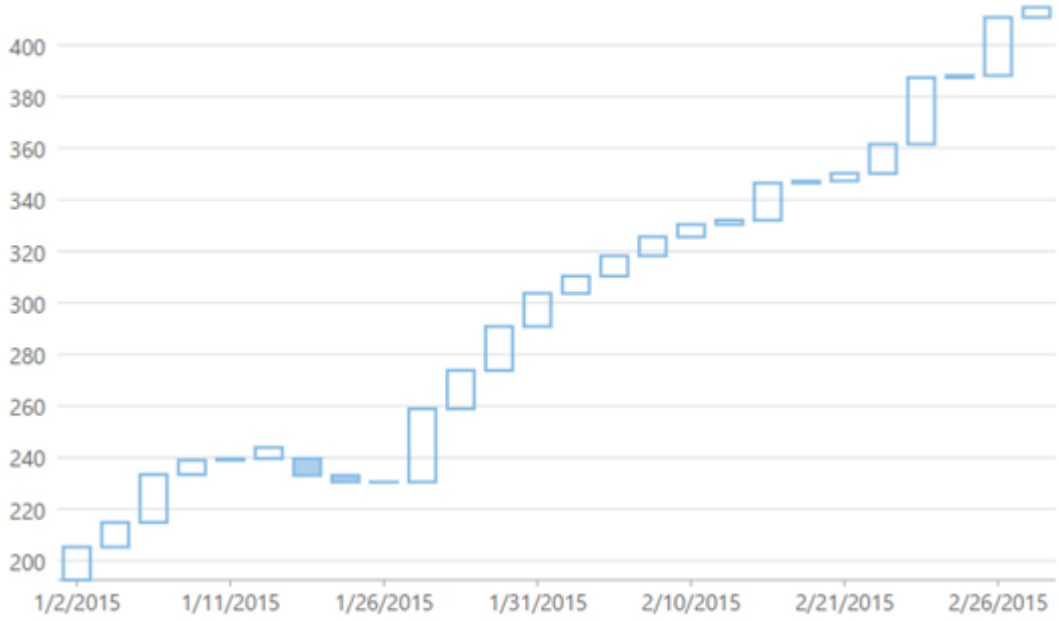
These charts are commonly used to show trends and performance over time.

Back to Top

Line Break chart

A Line Break or Three Line Break chart uses vertical boxes or lines to illustrate the price changes of an asset or market.

Movements are depicted with box colors and styles; movements that continue the trend of the previous box are colored similarly while movements that trend oppositely are indicated with a different color and/or style. The opposite trend is only drawn if its value exceeds the extreme value of the previous n number of boxes or lines, which is determined by the newLineBreaks option.



Back to Top

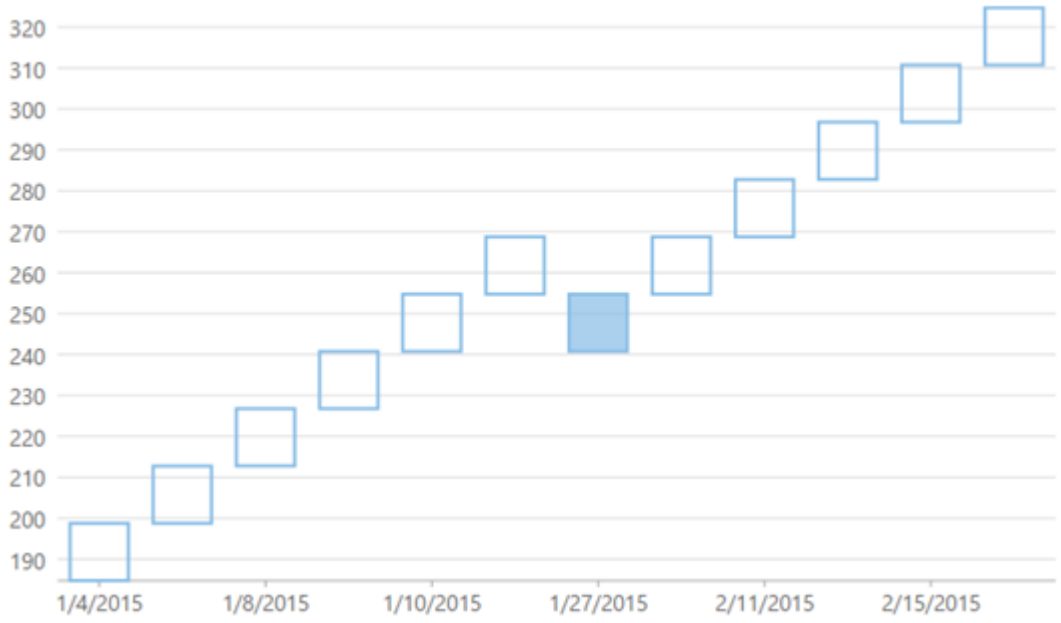
Line Symbols chart

LineSymbol chart is similar to line chart except that it represents data points using symbols. It is the most effective way of denoting changes in value between different groups of data.

Back to Top

Renko chart

The Renko chart uses bricks of uniform size to chart the price movement. When a price moves to a greater or lesser value than the preset boxSize option required to draw a new brick, a new brick is drawn in the succeeding column. The change in box color and direction signifies a trend reversal.
Renko chart ignores time, and solely focuses on price change.

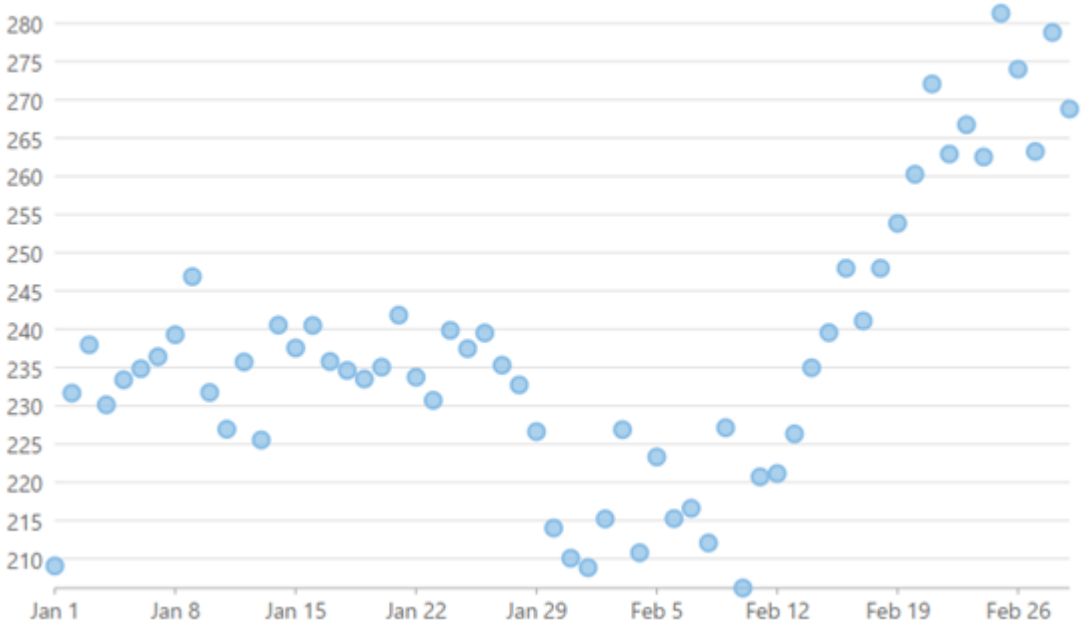


Back to Top

Scatter chart

A Scatter chart represents a series in the form of points plotted using their X and Y axis coordinates. The X and Y axis coordinates are combined into single data points and displayed in uneven intervals or clusters.

These charts are commonly used to determine the variation in data point density with varying x and y coordinates.



Back to Top

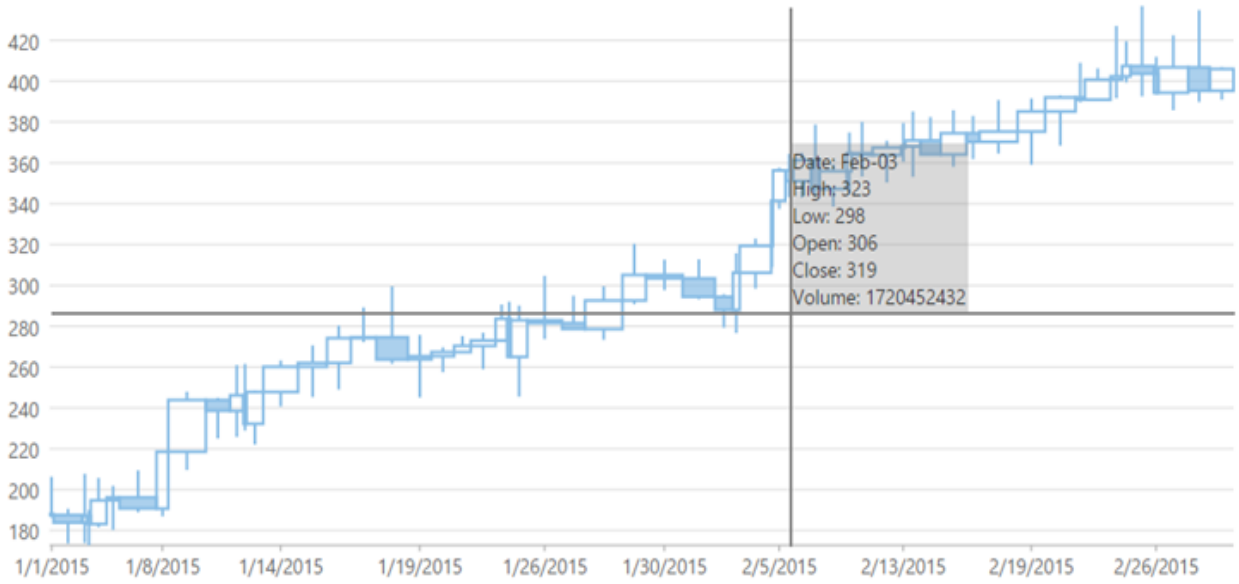
Features

Financial Charts Line Markers

Markers are the symbols used to display data points when the mouse is hovered over the data series. In FinancialChart, you can add line markers using **AddLineMarker** method. **LineMarkerLines** property allows you to set line markers to:

- **Vertical**: Shows vertical line markers.
- **Horizontal**: Shows horizontal line markers.
- **Both**: Sets vertical as well as horizontal line markers.
- **None**: Shows no line.

The image below shows how FinancialChart appears when the LineMarkerLines property is set to **Both (for a cross-hair effect)** to display the value of data points on the FinancialChart.



The following code example demonstrates how to add line markers and marker content to the FinancialChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    function lineMarkerContent(ht, pt) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
                              'High: ' + item.High.toFixed() + '<br/>' +
                              'Low: ' + item.Low.toFixed() + '<br/>' +
```

```
                                'Open: ' + item.Open.toFixed() + '<br/>' +
                                'Close: ' + item.Close.toFixed() + '<br/>' +
                                'Volume: ' + item.Volume.toFixed();
        }
    }
</script>
@*Initialize FinancialChart control.*@
@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
//Set chart type.
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.CandleVolume)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close,Volume");
    })
.Tooltip(tp => tp.Content(""))
//Add a line marker to FinancialChart.
.AddLineMarker(lm => lm
        .Alignment(C1.Web.Mvc.Chart.LineMarkerAlignment.Auto)
        .Lines(C1.Web.Mvc.Chart.LineMarkerLines.Both)
        .DragContent(true)

.Interaction(C1.Web.Mvc.Chart.LineMarkerInteraction.Move).Content("lineMarkerContent")))
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Chart;
@model List<FinanceData>


<script type="text/javascript">
    function lineMarkerContent(ht, pt) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
                                'High: ' + item.High.toFixed() + '<br/>' +
                                'Low: ' + item.Low.toFixed() + '<br/>' +
                                'Open: ' + item.Open.toFixed() + '<br/>' +
                                'Close: ' + item.Close.toFixed() + '<br/>' +
                                'Volume: ' + item.Volume.toFixed();
        }
    }
</script>
<c1-financial-chart binding-x="X" chart-
type="C1.Web.Mvc.chart.Finance.ChartType.CandleVolume">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close,Volume"></c1-financial-chart-
series>
<c1-flex-chart-tooltip content=""></c1-flex-chart-tooltip>
<c1-line-marker id="LineMarker" alignment="LineMarkerAlignment.Auto"
```

```
lines="LineMarkerLines.Both" drag-content="true"
interaction="LineMarkerInteraction.Move" content="lineMarkerContent"></c1-line-marker>
</c1-financial-chart>
```

Financial Charts Range Selector

FinancialChart's **RangeSelector** lets a user select a specific range of data to be displayed on the chart. A user can easily bind the RangeSelector with various types of financial charts. It is mostly used by finance industry to perform stock analysis on different data ranges.

The RangeSelector has a left thumb (for minimum value) and right thumb (for maximum value) that lets you scroll through particular time periods on the chart. Users can change the minimum and maximum values of the RangeSelector, and adjust the visible range of data on the chart by dragging these thumbs to left or right. On dragging the thumb towards left on the range bar, you reduce its value, and dragging it towards the right increases its value on the range bar.

To set the minimum and maximum range in code, use the **Min** and **Max** properties of the RangeSelector.

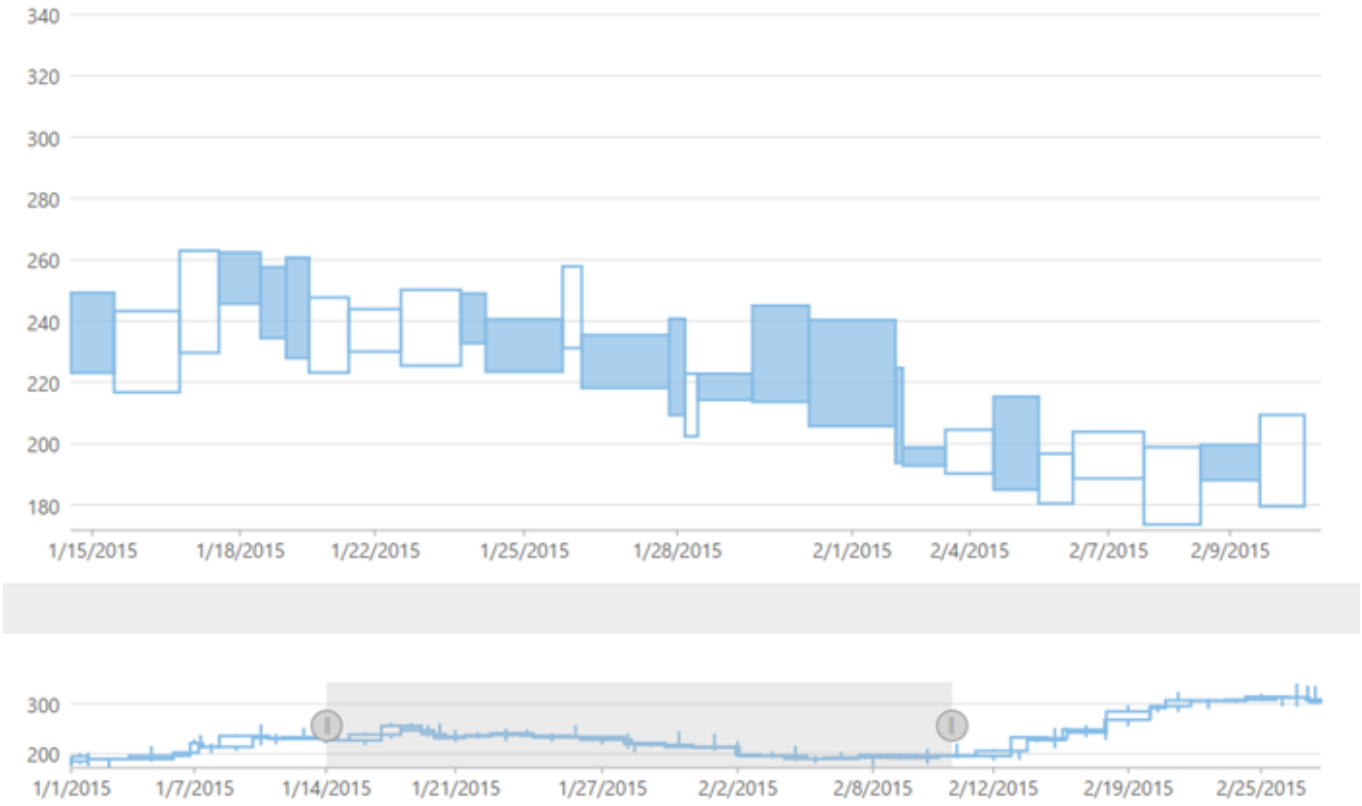
You can change the position of the thumbs by setting the value of RangeSelector's **Orientation** property to **X** (default) or **Y**.

C#

```
.Orientation(C1.Web.Mvc.Chart.Orientation.Y)
```

On setting the orientation to **Y**, the left and right thumbs get aligned as the lower and upper thumb on the RangeSelector, and can be dragged up or down to increase or decrease the values, and specify the visible range of data on the chart.

The image below shows how the FinancialChart appears after a RangeSelector is added to it.



The following code example demonstrates how to display thumb values using a range selector on the FinancialChart.

In Code

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>' +
             'Volume: ' + item.Volume.toFixed() + '<br/>'
        }
    };

    function rangeChanedHandler(sender, e) {
        var stChart = wijmo.Control.getControl("#FinancialChart"),
            rs = wijmo.Control.getControl("#rs").extenders[0];
```

```
        if (stChart && rs) {
            stChart.axisX.min = rs.min;
            stChart.axisX.max = rs.max;
            stChart.invalidate();
        }
    }
</script>
@*Initialize Financialchart control*@
@(Html.C1().FinancialChart()
.Id("FinancialChart")
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.EquiVolume)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close,Volume");
    })
.Tooltip(t => t.Content("tooltipContent")))
<p></p>
<br />
@(Html.C1().FinancialChart()
.Id("rs")
.Height("100px")
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.CandleVolume)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close,Volume");
    })
    //Add a range selector.
.AddRangeSelector(rs => rs.OnClientRangeChanged("rangeChanedHandler")
.Orientation(C1.Web.Mvc.Chart.Orientation.Y))
.Tooltip(t => t.Content("")))
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Chart;
@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>' +
```

```
            'Volume: ' + item.Volume.toFixed() + '<br/>'
        }
    };

    function rangeChangedHandler(sender, e) {
        var stChart = wijmo.Control.getControl("#FinancialChart"),
            rs = c1.getExtender(wijmo.Control.getControl("#rs"), "RangeSelector"),
            yRange;
        if (stChart && rs) {
            // update main chart's x & y range
            yRange = findRenderedYRange(stChart.collectionView.sourceCollection,
                                        rs.min, rs.max);
            stChart.axisX.min = rs.min;
            stChart.axisX.max = rs.max;
            stChart.axisY.min = yRange.min;
            stChart.axisY.max = yRange.max;
            stChart.invalidate();
        }
    }

</script>

<c1-financial-chart id="FinancialChart"  binding-x="X" chart-
type="C1.Web.Mvc.Finance.ChartType.CandleVolume">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close,Volume"></c1-financial-chart-
series>
<c1-flex-chart-tooltip content="tooltipContent"></c1-flex-chart-tooltip>

</c1-financial-chart>
<c1-financial-chart id="rs" binding-x="X" chart-
type="C1.Web.Mvc.Finance.ChartType.CandleVolume" height="150px">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close,Volume"></c1-financial-chart-
series>
<c1-range-selector range-changed="rangeChangedHandler" id="RangeSelector"></c1-range-
selector>
<c1-flex-chart-tooltip content=""></c1-flex-chart-tooltip>
</c1-financial-chart>
```

Financial Charts Trendline

Trendlines are used to represent trends in data and to examine problems of prediction. Trendlines are commonly used with price charts or financial charts, but they can also be used with a variety of technical analysis charts such as MACD (moving average convergence/divergence) which is a trading indicator used in technical analysis of stock prices, or RSI (relative strength index) which is a technical indicator used in the analysis of financial markets.

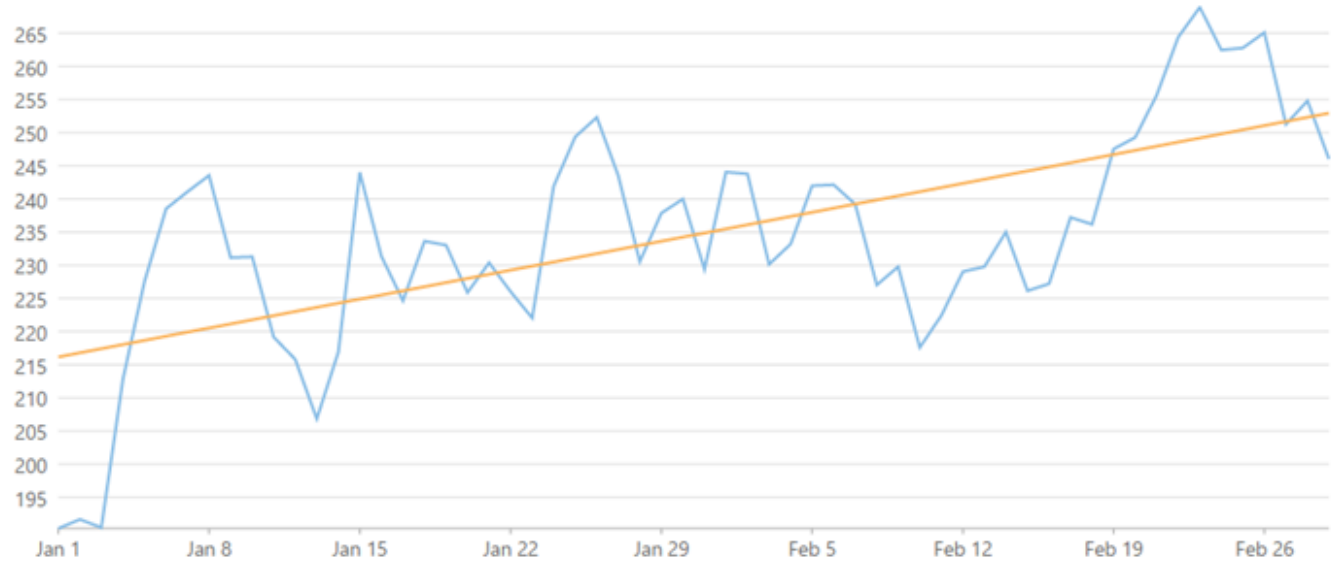
Types of Trendlines

The following table displays the supported FitTypes. Each trend type is drawn based on the calculation formula of its type.

Type	Description
Average X	Calculates the average value of X from the chart data and draws a trendline.
Average Y	Calculates the average value of Y from the chart data and draws a trendline.
Exponential	A curved line that is convenient to use when data values rise or fall at increasingly higher rates. You cannot create an exponential trendline if your data contains zero or negative values.
Fourier	A way to display a wave like function as a combination of simple sine waves. It is created by using the fourier series formula.
Linear	A linear trendline is a best-fit straight light. Your data is linear if the data point pattern resembles a line, and a linear trendline is a good fit if the R-squared value is at or near 1.
Logarithmic	A best fit curved line used for better visualization of data. Used when the rate of change in the data increases or decreases quickly and then levels out. It can also use positive and negative values.
Max X	Takes the maximum value of X from the chart and draws a trendline using it.
Max Y	Takes the maximum value of Y from the chart and draws a trendline using it.
Min X	Takes the minimum value of X from the chart and draws a trendline using it.
Min Y	Takes the minimum value of Y from the chart and draws a trendline using it
Polynomial	A twisted line that is used when data oscillates. It is useful for analyzing gains and losses over a large data set.
Power	A curved line that is best used with data sets that compare calculation that increase at a peculiar rate. For example, the acceleration of a vehicle at one-second intervals.

You can add trendlines to your charts to display trends of data. Using this, you can easily analyze the increase or decrease of your Y data over your X data.

The image below shows how trendline, with **FitType** set to Line, appears on the FinancialChart.



The following code example demonstrates how to add a trendline to the FinancialChart. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
        }
    };

</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.Line)
.Series(sers =>
    {
        sers.Add().Binding("Close");
```

```
        sers.AddTrendLine().Binding("Close");
    })
.Tooltip(t => t.Content("tooltipContent")))
```

Tag Helpers

HTML

```
@using C1.Web.Mvc.Chart;
@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
        }
    };
</script>

<c1-financial-chart binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Line">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-financial-chart-series binding="Close"></c1-financial-chart-series>
    <c1-flex-chart-trendline binding="Close" fit-type="TrendLineFitType.Linear"></c1-
flex-chart-trendline>
    <c1-flex-chart-tooltip content="tooltipContent"></c1-flex-chart-tooltip>
</c1-financial-chart>
```

Financial Charts Moving Average

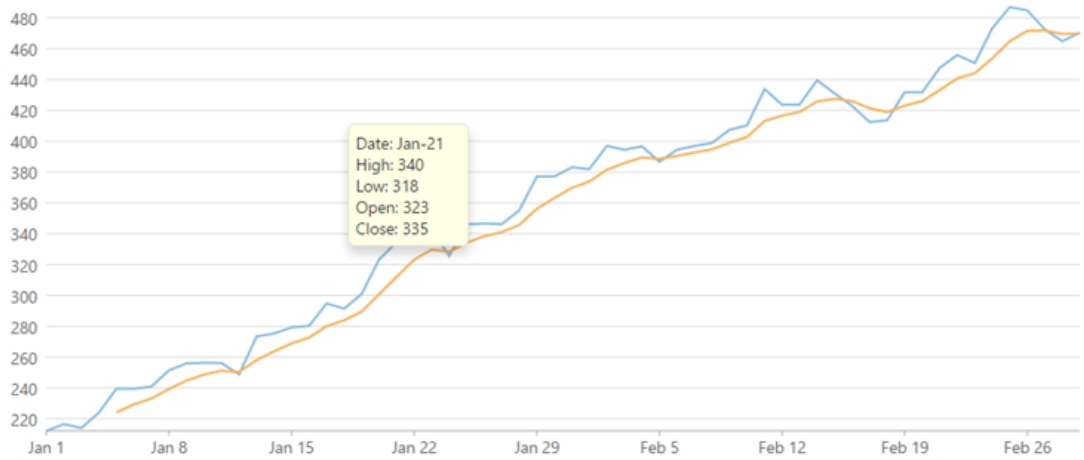
FinancialChart's MovingAerage represents a moving average trendline for a financial chart. It is a calculation to analyze data points by creating a series of averages of different subsets of the full data set. You may define a different type on each **MovingAverage** object by setting the **MovingAverageType** property to:

- **Exponential**: Weighted average of the last n values, where the weightage decreases exponentially with each previous value.
- **Simple**: An average of the last n values.
- **Triangular**: Weighted average of the last n values, whose result is equivalent to a double smoothed simple moving average.
- **Weighted**: Weighted average of the last n values, where the weightage decreases by 1 with each previous value.

In this example, MovingAverageType property value is set as **Exponential**.

Use the **Period** property of the **MovingAverage** class to set the number of periods for computing the average value. Its value should be set to integer value **greater than 1**. In this example, the value of **Period** property is assigned an integer value 5.

The image below shows how FinancialChart appears when the MovingAverage is used to calculate and display average on the chart, the **MovingAverageType** value is set to **Exponential** and the value of **Period** property to **5**.



The following code example demonstrates how to use MovingAverage to analyze data on FinancialChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
        }
    };

</script>
@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.Line)
.Series(sers =>
    {
        sers.Add().Binding("Close");

sers.AddMovingAverage().Binding("Close").Period(5).Type(C1.Web.Mvc.Chart.MovingAverageType.Exponential);
    })
.Tooltip(t => t.Content("tooltipContent")))
```

Tag Helpers

HTML

```
 @using C1.Web.Mvc.Chart;
@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
```

```
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
        }
    };

</script>
<c1-financial-chart binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Line">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="Close"></c1-financial-chart-series>
<c1-flex-chart-movingaverage binding="Close" period="5" type="MovingAverageType.Exponential"></c1-flex-
chart-movingaverage>
<c1-flex-chart-tooltip content=""></c1-flex-chart-tooltip>
</c1-financial-chart>
```

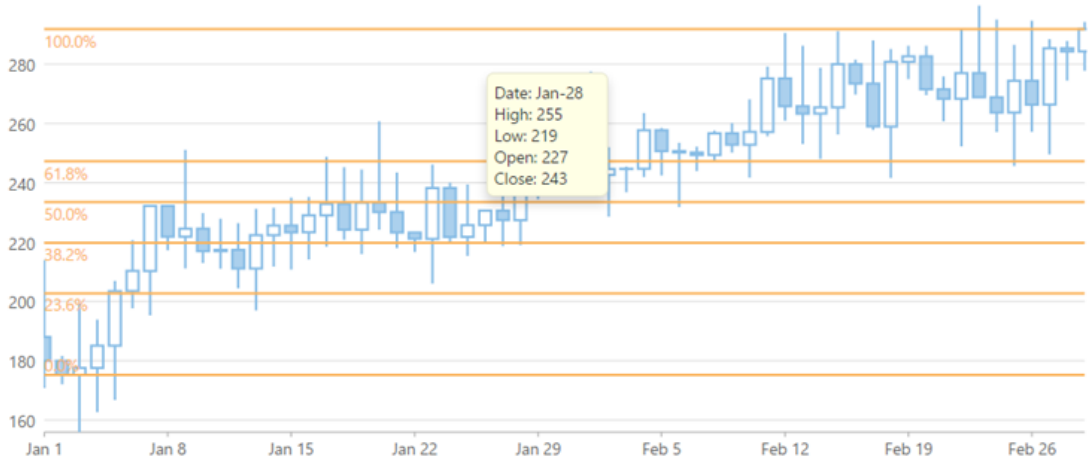
Financial Charts Fibonacci Tool

Fibonacci tool enables the calculation and plotting of various alert levels that are useful in financial charts. This topic explains how to initialize a Fibonacci tool in a FinancialChart, and use it for performing stock analysis.

To display uptrend or downtrend using Fibonacci series, set the value of Uptrend property to true (default) or false. The Uptrend value, when set to true, plots uptrending values, and setting it to false indicates and plots the downtrending value. Set the position of the label by setting the LabelPosition property to:

- **Bottom**: The labels appear below the data points.
- **Center**: The labels appear centered on the data points.
- **Left**: The labels appear to the left of the data points.
- **None**: No data labels appear.
- **Right**: The labels appear to the right of the data points.
- **Top**: The labels appear above the data points.

The image below shows how FinancialChart appears when the Fibonacci series is added to the chart, and **Uptrend** value is set to true with **LabelPosition** set to Left. The image shows Fibonacci Retracements.



The following code example demonstrates how to add Fibonacci series and tooltip content to the FinancialChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor	copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script type="text/javascript">
    var tooltipContent = function (ht) {
        var item = ht.series.collectionView.items[ht.pointIndex];
        if (item) {
            return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
        }
    };

</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Chart.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close");

sers.AddFibonacci().Binding("Close").Uptrend(true).LabelPosition(C1.Web.Mvc.Chart.LabelPosition.Left);
    })
.Tooltip(t => t.Content("tooltipContent")))
```

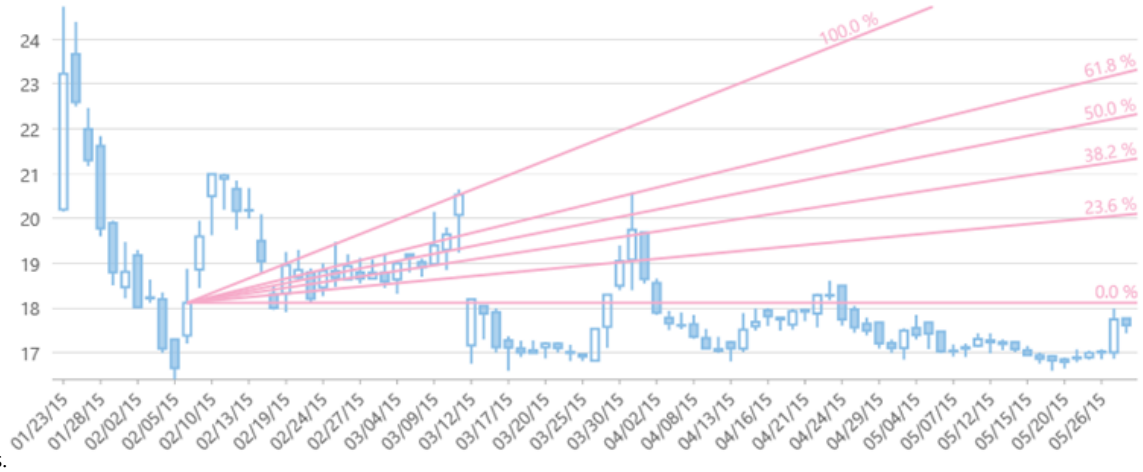
Tag Helpers

HTML

```
@using C1.Web.Mvc.Chart;
@model List<FinanceData>
<script type="text/javascript">
var tooltipContent = function (ht) {
var item = ht.series.collectionView.items[ht.pointIndex];
if (item) {
return 'Date: ' + wijmo.Globalize.format(ht.x, 'MMM-dd') + '<br/>' +
             'High: ' + item.High.toFixed() + '<br/>' +
             'Low: ' + item.Low.toFixed() + '<br/>' +
             'Open: ' + item.Open.toFixed() + '<br/>' +
             'Close: ' + item.Close.toFixed() + '<br/>'
}
};
</script>
<c1-financial-chart binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Candlestick">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close"></c1-financial-chart-series>
<c1-flex-chart-fibonacci binding="Close" uptrend="true" label-position="LabelPosition.Left"></c1-flex-
chart-fibonacci>
<c1-flex-chart-tooltip content="tooltipContent"></c1-flex-chart-tooltip>
</c1-financial-chart>
```

Fibonacci Fans

Financial charts in MVC Edition support Fibonacci Fan lines, which are trend lines that are based on Fibonacci retracement points. The rising Fibonacci Fan lines can be used to predict support levels or reversal zones, while falling fan lines can help predict resistance levels or reversal



zones.

In Code

HTML Helpers

Razor

```
@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close").Name("BOX");
        sers.AddFibonacciFans().Binding("Close").Start(new DataPoint(10, 18.12)).End(new DataPoint(32,
20.53)).LabelPosition(C1.Web.Mvc.Chart.LabelPosition.Top);

    })
```

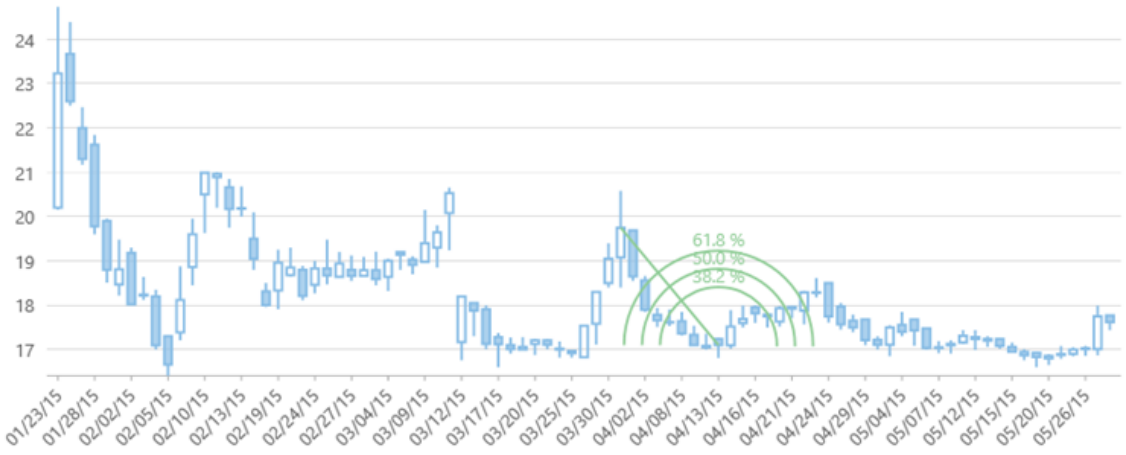
Tag Helpers

HTML

```
<c1-financial-chart binding-x="X"
chart-type="C1.Web.Mvc.Finance.ChartType.Candlestick">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close" name="BOX"></c1-financial-chart-series c1-
flex-chart-fibonacci-fans binding="Close" start="new DataPoint(10, 18.12)" end="new DataPoint(32,
20.53)" label-position="LabelPosition.Top"></c1-flex-chart-fibonacci-fans>
/c1-financial-chart>
```

Fibonacci Arcs

These are the half circles extending out from a trend line. Fibonacci arcs help predict reversal zones or resistance for counter trend bounce situations after decline.



In Code

HTML Helpers

Razor

```
@(Html.C1().FinancialChart()
.Id(demoSettingsModel.ControlId)
.Bind(Model)
.BindingX("X")
.SymbolSize(6)
.ChartType(C1.Web.Mvc.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close").Name("BOX");
        sers.AddFibonacciArcs().Binding("Close").Start(new DataPoint(46, 19.75)).End(new DataPoint(54,
17.10)).LabelPosition(C1.Web.Mvc.Chart.LabelPosition.Left);
    })
```

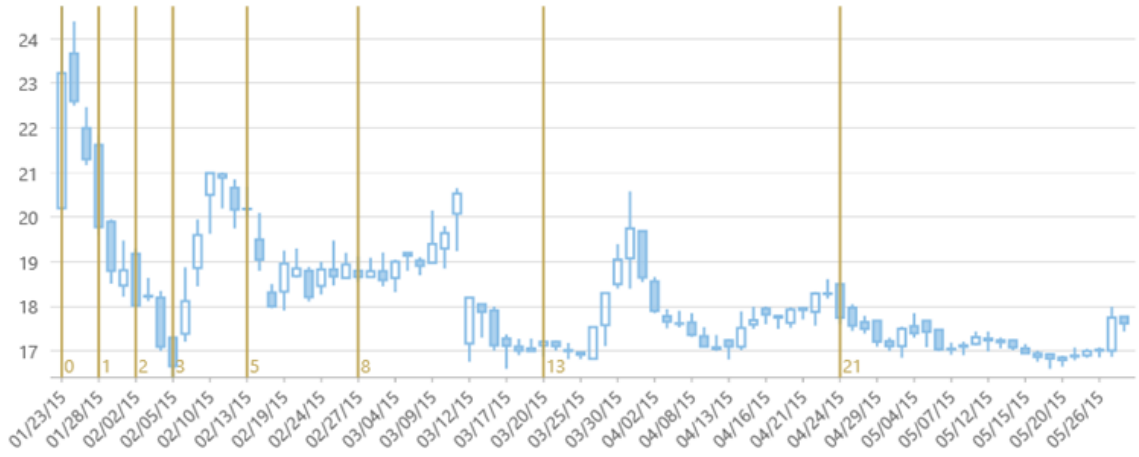
Tag Helpers

HTML

```
<c1-financial-chart binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Candlestick">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-financial-chart-series binding="High,Low,Open,Close" name="BOX"></c1-financial-chart-series>
<c1-flex-chart-fibonacci-arcs binding="Close" start="new DataPoint(46, 19.75)" end="new DataPoint(54,
17.10)" label-position="LabelPosition.Left"></c1-flex-chart-fibonacci-arcs>
</c1-financial-chart>
```

Fibonacci Time Zones

Fibonacci Time Zones are based on Fibonacci sequence and represented as vertical lines, which are used to predict reversal points in future. MVC Edition supports Fibonacci Time Zones for Financial Charts.



In Code

HTML Helpers

Razor

```
@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.ChartType(C1.Web.Mvc.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close").Name("BOX");
        sers.AddFibonacciTimeZones().Binding("Close").StartX(0).EndX(3);
    })
```

Tag Helpers

HTML

```
<c1-financial-chart binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Candlestick">
        <c1-items-source source-collection="Model"></c1-items-source>
        <c1-financial-chart-series binding="High,Low,Open,Close" name="BOX"></c1-financial-chart-
series>
        <c1-flex-chart-fibonacci-time-zones binding="Close" start-x="0" end-x="3"></c1-flex-chart-
fibonacci-time-zones>
        </c1-financial-chart>
```

Financial Charts Indicators

Indicators or technical indicators are mathematical calculations or a series of data points, that are based on a trading instrument's historic and current price and the volume activity. These data points are derived by applying formulas to price of financial instruments. Indicators aim to forecast financial market direction and form a basis for trading decisions.

MVC Edition supports technical indicators for its financial chart control, to be easily used in financial applications. These financial indicators are plotted as chart patterns and form a basis for technical analysis. Note that the indicators are generally plotted separately from original price or volume data, as Y axis scales for technical indicators differ from that of price or volume chart data. The following sections discuss various financial chart indicators that MVC Edition supports.

Moving Average Convergence Divergence

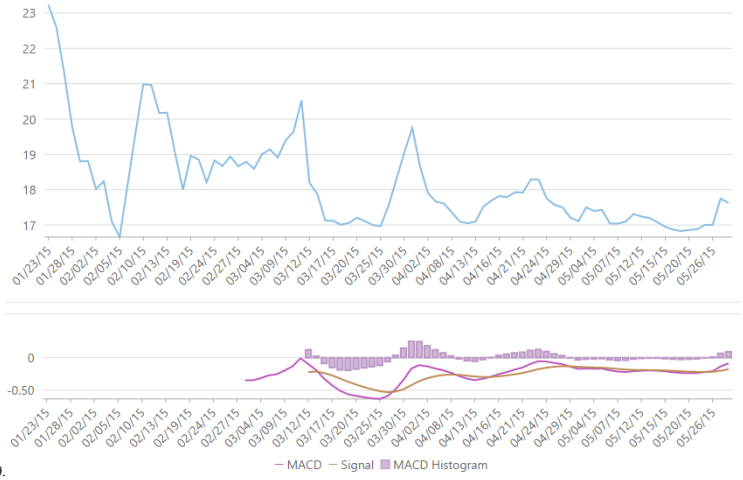
Moving Average Convergence Divergence (MACD) indicator for the FinancialChart is a trend-following momentum indicator. It reveals changes in the strength, direction, duration and momentum of price of an asset. MACD is efficient in helping the users spot short-term price momentum.

MACD momentum oscillator displays a relationship between 26 day exponential moving average and 12 day exponential moving average. As the two moving averages converge, cross and diverge, they make the MACD oscillator to fluctuate above and below the zero line. A "signal line" is plotted on the top of the oscillator. It is a 9 day exponential moving average of MACD, which serves as a trigger for buy and sell signals.

MACD Histogram

MACD Histogram is an oscillator which measures the difference between the fast MACD line and the signal line. Just like MACD indicator, histogram also fluctuates above and below zero line. A positive histogram indicates that MACD is above its signal line, while MACD going below its signal line makes a negative histogram.

The image below shows how FinancialChart appears when Moving Average Convergence Divergence indicator and Moving Average Convergence Divergence histogram oscillator are used, with Fast Period set to 12,



Slow Period set to 26 and Signal Smoothing Period set to 9.

The following code example demonstrates how to add Moving Average Convergence Divergence indicator and Moving Average Convergence Divergence Histogram Oscillator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path, System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open, Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit organization committed to accelerating the transition to a sustainable world
powered by clean energy, is leveraging the Box platform to power a cloud-based document submission and review process as part of the Clean Vehicle Rebate
Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate in the J.P. Morgan Technology, Media and Telecom Conference in Boston on
Monday, May 18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™ Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery and Fewer Flow Interruptions including Silent Occlusions Infusion Set
Developed in Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX) has declared a quarterly dividend of 60.0 cents per common share payable on
```

```
June ...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first quarter which ended April 30, 2015, following the close of the market on
Wednesday, June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
     "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
     "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target
of $14. The analysts believe that the company's dividend ...",
     "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File Sharing</b>"+
     "<br>Box today announced that it is working with the U.S. Department of Justice to deliver file sharing and information management to the agency's
workforce. After rigorous assessment, Box will receive an agency Authorization to Operate this week, allowing the DOJ to leverage the platform across all its
component agencies.",
     "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
     "<br>Box today announced that the ASPCA© , the first humane society established in North America, is bringing Box's cloud content sharing and
collaboration platform to its employees."
     };

        return _annotationToolTips;
    }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json


```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23, "volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6, "volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3, "volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78, "volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8, "volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81, "volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02, "volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24, "volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1, "volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66, "volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12, "volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6, "volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99, "volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96, "volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17, "volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18, "volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05, "volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18, "volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96, "volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85, "volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21, "volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83, "volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67, "volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94, "volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66, "volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79, "volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59, "volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19, "volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14, "volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91, "volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4, "volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64, "volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53, "volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2, "volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88, "volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13, "volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12, "volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01, "volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06, "volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21, "volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11, "volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17, "volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97, "volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54, "volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3, "volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05, "volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75, "volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65, "volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9, "volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66, "volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61, "volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36, "volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1, "volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05, "volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1, "volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52, "volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69, "volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82, "volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79, "volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93, "volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92, "volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29, "volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28, "volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75, "volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57, "volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5, "volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21, "volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11, "volume": 391040 },
```

```
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5, "volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4, "volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43, "volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04, "volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04, "volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1, "volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31, "volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24, "volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2, "volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08, "volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95, "volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87, "volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83, "volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86, "volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88, "volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17, "volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01, "volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75, "volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62, "volume": 338482 }
]
```

MACDController.cs

C# copyCode

```
// GET: /MACD/
public ActionResult MACDIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }

    function chartRendered(sender, args) {
        // set min/max for indicator chart to match main chart
        var indicatorChart = wijmo.Control.getControl('#indicator'),
            chart = wijmo.Control.getControl('#financeChart');

        if (chart && indicatorChart) {
            indicatorChart.axisX.min = chart.axisX.actualMin;
            indicatorChart.axisX.max = chart.axisX.actualMax;
        }
    };
</script>

@(Html.C1().FinancialChart().Id("financeChart")
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
.OnClientRendered("chartRendered")
)

@(Html.C1().FinancialChart().Height(200).Id("indicator")
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
```

```
sers.AddMacd().Binding("Close").FastPeriod(12).SlowPeriod(26).SmoothingPeriod(9).SetMacdLineStyle("#bfa554").SetSignalLineStyle("#bf8c54").Name("MACD,Signal");
})
.Tooltip(t => t.Content("tooltip"))
.OnClientRendered("chartRendered")
)
```

Tag Helpers

HTML copyCode

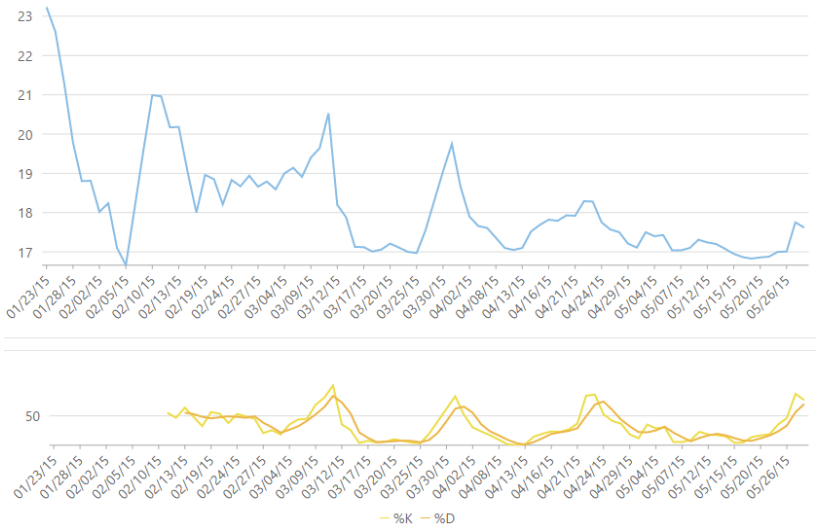
```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }

    function chartRendered(sender, args) {
        // set min/max for indicator chart to match main chart
        var indicatorChart = wijmo.Control.getControl('#indicator'),
            chart = wijmo.Control.getControl('#financeChart');

        if (chart && indicatorChart) {
            indicatorChart.axisX.min = chart.axisX.actualMin;
            indicatorChart.axisX.max = chart.axisX.actualMax;
        }
    };
</script>

                <c1-financial-chart binding-x="X" rendered="chartRendered" id="financeChart">
                    <c1-items-source source-collection="Model"></c1-items-source>
                    <c1-financial-chart-series binding="Close"></c1-financial-chart-series>
                    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
                </c1-financial-chart>

                <c1-financial-chart height="200px" binding-x="X" id="indicator" rendered="chartRendered">
                    <c1-items-source source-collection="Model"></c1-items-source>
                    <c1-flex-chart-macd binding="Close" fast-period="12" slow-period="26" smoothing-period="9">
                        <c1-flex-chart-line-style c1-property="macdLine" stroke="#bfa554"></c1-flex-chart-line-style>
                        <c1-flex-chart-line-style c1-property="signalLine" stroke="#bf8c54"></c1-flex-chart-line-style>
                    </c1-flex-chart-macd>
                    <c1-flex-chart-macd-histogram binding="Close" fast-period="12" slow-period="26" smoothing-period="9"></c1-flex-chart-macd-histogram>
                    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
                </c1-financial-chart>
```

Stochastic Oscillator Indicator

Stochastic Oscillator indicator for Financial Charts is a momentum indicator to foreshadow price turning points. It follows momentum of a price, and can be used to anticipate future reversals by identifying bull and bear set-ups.

A Stochastic Oscillator indicator series can be used for fast (default), slow and full stochastic oscillators. To create a slow or full stochastic oscillator, SmoothingPeriod is set to an integer value greater than one, whereas, slow stochastic oscillators generally use a fixed SmoothingPeriod of three. To create to a fast stochastic oscillator, the SmoothingPeriod is set to an integer value of one.

The image below shows how FinancialChart appears when the Stochastic Oscillator indicator is added to the chart, with K Period set to 14 , D Period set to 3 and Smoothing Period set to 1.



The following code example demonstrates how to add Stochastic Oscillator indicator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path, System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open, Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit organization committed to accelerating the transition to a
sustainable world powered by clean energy, is leveraging the Box platform to power a cloud-based document submission and review process as part
of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate in the J.P. Morgan Technology, Media and Telecom Conference
in Boston on Monday, May 18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™ Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery and Fewer Flow Interruptions including Silent Occlusions
Infusion Set Developed in Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX) has declared a quarterly dividend of 60.0 cents per common
share payable on June ...",
```

```
    "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
    "<br>Box today announced that it will report financial results for its first quarter which ended April 30, 2015, following the close of the
market on Wednesday, June 10, 2015.",
    "<b>Making Money With Charles Payne: 05/21/15</b>",
    "<b>Best Buy to move sideways: Analyst</b>",
    "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
    "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with
a price target of $14. The analysts believe that the company's dividend ...",
    "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File Sharing</b>"+
    "<br>Box today announced that it is working with the U.S. Department of Justice to deliver file sharing and information management to the
agency’s workforce. After rigorous assessment, Box will receive an agency Authorization to Operate this week, allowing the DOJ to leverage the
platform across all its component agencies.",
    "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
    "<br>Box today announced that the ASPCA® , the first humane society established in North America, is bringing Box’s cloud content sharing
and collaboration platform to its employees."
    };

        return _annotationToolTips;
    }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json


```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23, "volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6, "volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3, "volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78, "volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8, "volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81, "volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02, "volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24, "volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1, "volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66, "volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12, "volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6, "volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99, "volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96, "volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17, "volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18, "volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05, "volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18, "volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96, "volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85, "volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21, "volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83, "volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67, "volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94, "volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66, "volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79, "volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59, "volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19, "volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14, "volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91, "volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4, "volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64, "volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53, "volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2, "volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88, "volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13, "volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12, "volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01, "volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06, "volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21, "volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11, "volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17, "volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97, "volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54, "volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3, "volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05, "volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75, "volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65, "volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9, "volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66, "volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61, "volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36, "volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1, "volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05, "volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1, "volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52, "volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69, "volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82, "volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79, "volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93, "volume": 487017 },
```

```
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92, "volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29, "volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28, "volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75, "volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57, "volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5, "volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21, "volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11, "volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5, "volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4, "volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43, "volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04, "volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04, "volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1, "volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31, "volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24, "volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2, "volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08, "volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95, "volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87, "volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83, "volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86, "volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88, "volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17, "volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01, "volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75, "volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62, "volume": 338482 }
]
```

StochasticOscillatorController.cs

C# copyCode

```
// GET: /Stochastic/
public ActionResult StochasticIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
)
```

```
@(Html.C1().FinancialChart().Height(200)
.Bind(Model)
.BindingX("X")
.Series(sers =>
{

sers.AddStochastic().Binding("High,Low,Open,Close").KPeriod(14).DPeriod(3).SmoothingPeriod(1).SetKLineStyle("#eddd46").SetDLineStyle("#edb747");
})
.Tooltip(t => t.Content("tooltip"))
)
```

Tag Helpers

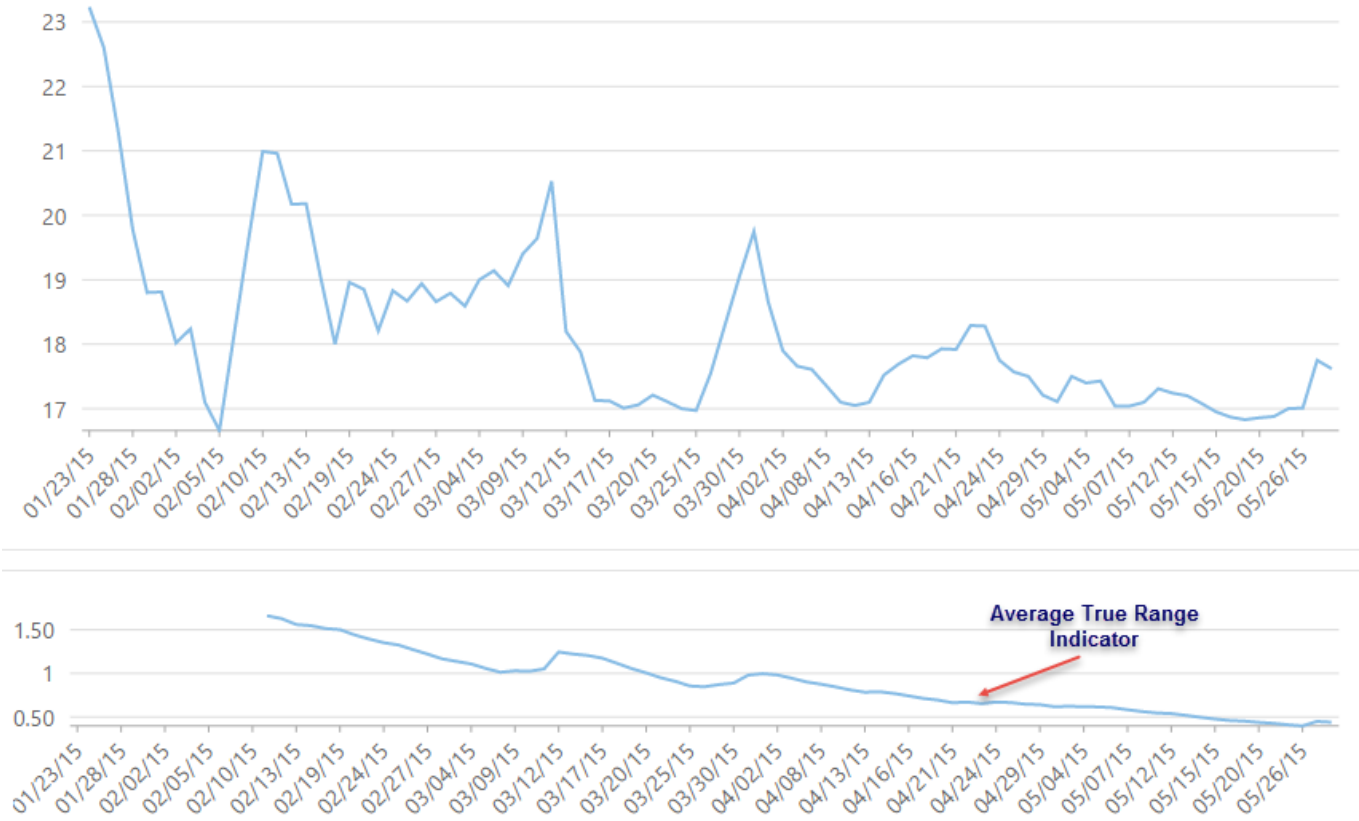
HTML copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

               <c1-financial-chart binding-x="X" rendered="chartRendered">
                   <c1-items-source source-collection="Model"></c1-items-source>
                   <c1-financial-chart-series binding="Close"></c1-financial-chart-series>
                   <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
               </c1-financial-chart>

               <c1-financial-chart height="200px" binding-x="X">
                   <c1-items-source source-collection="Model"></c1-items-source>
                   <c1-flex-chart-stochastic binding="High,Low,Open,Close" k-period="14" d-period="3" smoothing-period="1">
                       <c1-flex-chart-line-style c1-property="kLine" stroke="#eddd46"></c1-flex-chart-line-style>
                       <c1-flex-chart-line-style c1-property="dLine" stroke="#edb747"></c1-flex-chart-line-style>
                   </c1-flex-chart-stochastic>
                   <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
               </c1-financial-chart>
```

Average True Range Indicator

Average True Range (ATR) indicator for FinancialChart is a technical indicator for price volatility of an asset. It is typically based on 14 periods, and could be calculated intra daily, daily, weekly or monthly basis. It gives information about the degree of price volatility, and gives no indication of price trends. Stocks having high volatility will have a higher ATR, whereas low volatility stocks will have a lower ATR.

The image below shows how FinancialChart appears when the Average True Range indicator is added to the chart, and Period is set to 14.



The following code example demonstrates how to add Average True Range Index indicator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path,
System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
```

```
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open,
Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean
Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit
organization committed to accelerating the transition to a sustainable world powered
by clean energy, is leveraging the Box platform to power a cloud-based document
submission and review process as part of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom
Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate
in the J.P. Morgan Technology, Media and Telecom Conference in Boston on Monday, May
18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™
Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery
and Fewer Flow Interruptions including Silent Occlusions Infusion Set Developed in
Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX)
has declared a quarterly dividend of 60.0 cents per common share payable on June
...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first
quarter which ended April 30, 2015, following the close of the market on Wednesday,
June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
     "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
     "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on
```

```
Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14.
The analysts believe that the company's dividend ...",
     "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File
Sharing</b>"+
     "<br>Box today announced that it is working with the U.S. Department of Justice
to deliver file sharing and information management to the agency’s workforce. After
rigorous assessment, Box will receive an agency Authorization to Operate this week,
allowing the DOJ to leverage the platform across all its component agencies.",
     "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
     "<br>Box today announced that the ASPCA® , the first humane society established
in North America, is bringing Box’s cloud content sharing and collaboration platform
to its employees."
     };

         return _annotationToolTips;
     }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json

```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23,
"volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6,
"volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3,
"volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78,
"volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8,
"volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81,
"volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02,
"volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24,
"volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1,
"volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66,
"volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12,
"volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6,
"volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99,
"volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96,
"volume": 1698365 },
```

```
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17,
"volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18,
"volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05,
"volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18,
"volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96,
"volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85,
"volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21,
"volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83,
"volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67,
"volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94,
"volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66,
"volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79,
"volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59,
"volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19,
"volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14,
"volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91,
"volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4,
"volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64,
"volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53,
"volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2,
"volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88,
"volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13,
"volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12,
"volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01,
"volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06,
"volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21,
```

```
"volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11,
"volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17,
"volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97,
"volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54,
"volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3,
"volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05,
"volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75,
"volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65,
"volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9,
"volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66,
"volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61,
"volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36,
"volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1,
"volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05,
"volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1,
"volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52,
"volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69,
"volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82,
"volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79,
"volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93,
"volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92,
"volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29,
"volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28,
"volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75,
"volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57,
"volume": 437294 },
```

```
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5,
"volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21,
"volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11,
"volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5,
"volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4,
"volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43,
"volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04,
"volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04,
"volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1,
"volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31,
"volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24,
"volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2,
"volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08,
"volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95,
"volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87,
"volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83,
"volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86,
"volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88,
"volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17,
"volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01,
"volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75,
"volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62,
"volume": 338482 }
]
```


ATRController.cs

C# copyCode

```
// GET: /ATR/
public ActionResult ATRIndex()
{
```

```
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
)
```

```
@(Html.C1().FinancialChart().Height(200)
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.AddATR().Binding("High,Low,Open,Close").Period(14).Name("ATR");
})
.Tooltip(t => t.Content("tooltip"))
)
```

Tag Helpers

HTML copyCode

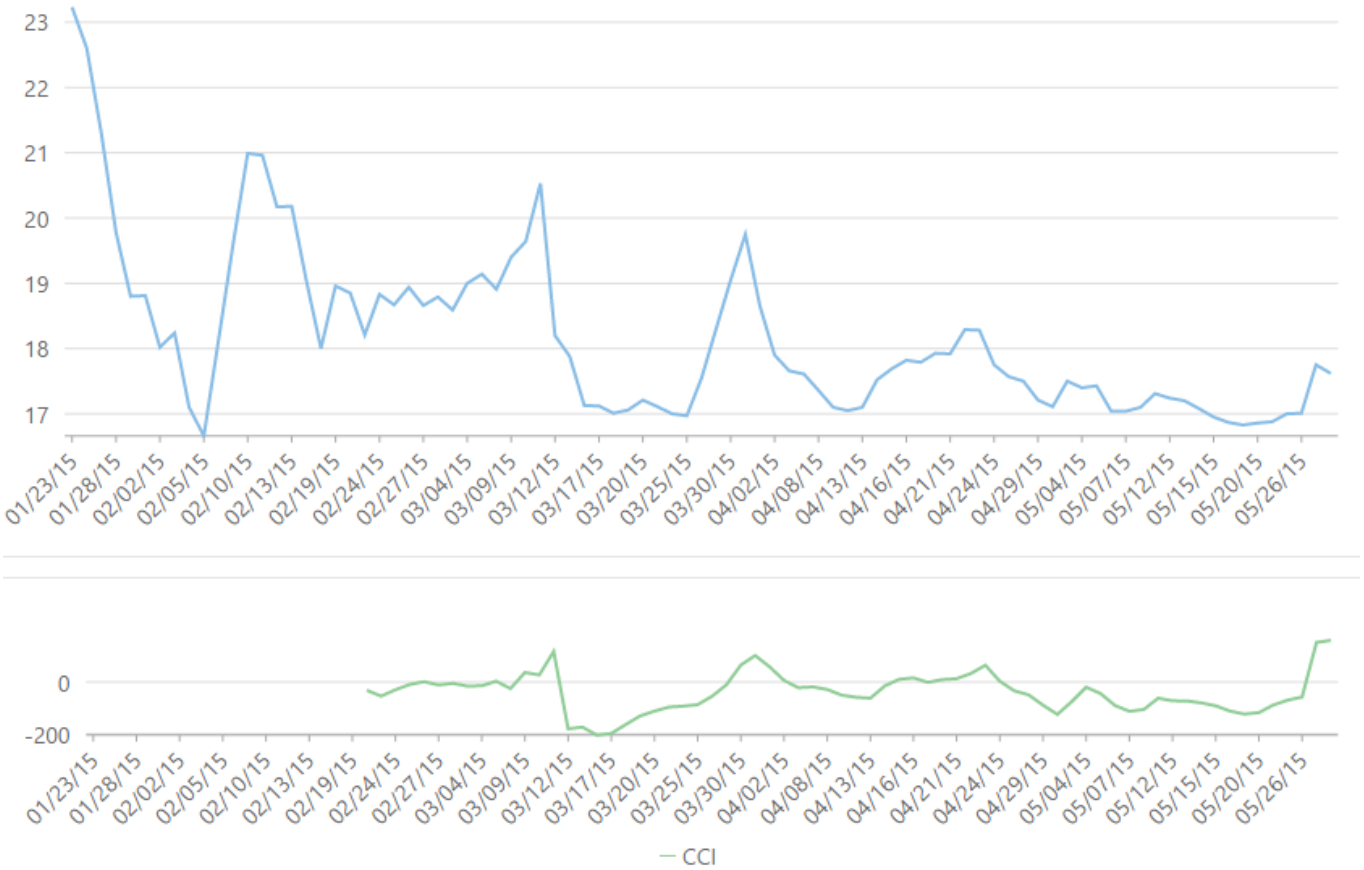
```
@using MVCFinancialChart.Models

@model List<financedata>

    <script>
        // function used for displaying XYV data in tooltips
        function tooltip(ht) {
            var date, content;

            if (!ht || !ht.item) {
                return '';
            }

            date = ht.item.X ? ht.item.X : null;

            if (wijmo.isDate(date)) {
                date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
            }

            content =
                    '<b>' + ht.name + '</b><br/>' +
                    'Date: ' + date + '<br/>' +
                    'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
            if (ht.item.Volume) {
                content +=
                '<br/>' +
                'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
            }

            return content;
        }
    </script>

                <c1-financial-chart binding-x="X" rendered="chartRendered">
                    <c1-items-source source-collection="Model"></c1-items-source>
```

```
                <c1-financial-chart-series binding="Close"></c1-financial-chart-
series>
                <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
            </c1-financial-chart>

            <c1-financial-chart height="200px" binding-x="X">
                <c1-items-source source-collection="Model"></c1-items-source>
                <c1-flex-chart-atr binding="High,Low,Open,Close" period="14"></c1-
flex-chart-atr>
                <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
            </c1-financial-chart>
```

Commodity Channel Index Indicator

Commodity Channel Index (CCI) indicator for FinancialChart is an oscillator, which displays a stock's variation from its average price. It is used to determine a new trend or to warn about extreme conditions. Typically, it measures the current price levels of an asset to its average price over a particular period.

The image below shows how FinancialChart appears when the Commodity Channel Index indicator is added to the chart, and the Period is set to 20.



The following code example demonstrates how to add Commodity Channel Index indicator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path,
System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open,
Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean
Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit
organization committed to accelerating the transition to a sustainable world powered
by clean energy, is leveraging the Box platform to power a cloud-based document
submission and review process as part of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom
Conference</b>"+
```

```
        "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate
in the J.P. Morgan Technology, Media and Telecom Conference in Boston on Monday, May
18, 2015.",
        "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™
Technology to Enhance the Use of Insulin Pumps</b>"+
        "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery
and Fewer Flow Interruptions including Silent Occlusions Infusion Set Developed in
Collaboration with JDRF and Helmsley Charitable ...",
        "<b>Wall Street tech debutant Box dismisses doubters</b>",
        "<b>BD Board Declares Dividend</b>"+
        "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX)
has declared a quarterly dividend of 60.0 cents per common share payable on June
...",
        "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
        "<br>Box today announced that it will report financial results for its first
quarter which ended April 30, 2015, following the close of the market on Wednesday,
June 10, 2015.",
        "<b>Making Money With Charles Payne: 05/21/15</b>",
        "<b>Best Buy to move sideways: Analyst</b>",
        "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
        "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on
Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14.
The analysts believe that the company's dividend ...",
        "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File
Sharing</b>"+
        "<br>Box today announced that it is working with the U.S. Department of Justice
to deliver file sharing and information management to the agency’s workforce. After
rigorous assessment, Box will receive an agency Authorization to Operate this week,
allowing the DOJ to leverage the platform across all its component agencies.",
        "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
        "<br>Box today announced that the ASPCA® , the first humane society established
in North America, is bringing Box’s cloud content sharing and collaboration platform
to its employees."
        };

            return _annotationToolTips;
        }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json

```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23,
"volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6,
"volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3,
"volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78,
```

```
"volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8,
"volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81,
"volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02,
"volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24,
"volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1,
"volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66,
"volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12,
"volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6,
"volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99,
"volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96,
"volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17,
"volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18,
"volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05,
"volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18,
"volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96,
"volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85,
"volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21,
"volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83,
"volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67,
"volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94,
"volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66,
"volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79,
"volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59,
"volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19,
"volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14,
"volume": 486149 },
```

```
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91,
"volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4,
"volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64,
"volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53,
"volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2,
"volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88,
"volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13,
"volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12,
"volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01,
"volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06,
"volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21,
"volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11,
"volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17,
"volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97,
"volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54,
"volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3,
"volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05,
"volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75,
"volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65,
"volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9,
"volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66,
"volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61,
"volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36,
"volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1,
"volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05,
"volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1,
```

```
"volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52,
"volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69,
"volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82,
"volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79,
"volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93,
"volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92,
"volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29,
"volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28,
"volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75,
"volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57,
"volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5,
"volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21,
"volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11,
"volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5,
"volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4,
"volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43,
"volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04,
"volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04,
"volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1,
"volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31,
"volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24,
"volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2,
"volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08,
"volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95,
"volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87,
"volume": 418513 },
```

```
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83,
"volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86,
"volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88,
"volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17,
"volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01,
"volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75,
"volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62,
"volume": 338482 }
]
```

CCIController.cs

C# copyCode

```
// GET: /CCI/
public ActionResult CCIIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
```

```
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
)

@(Html.C1().FinancialChart().Height(200)
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.AddCCI().Binding("High,Low,Open,Close").Period(20).Name("CCI");
})
.Tooltip(t => t.Content("tooltip"))
)
```

Tag Helpers

HTML copyCode

```
@using MVCFinancialChart.Models

@model List<financedata>

    <script>
        // function used for displaying XYV data in tooltips
        function tooltip(ht) {
            var date, content;

            if (!ht || !ht.item) {
                return '';
            }
```

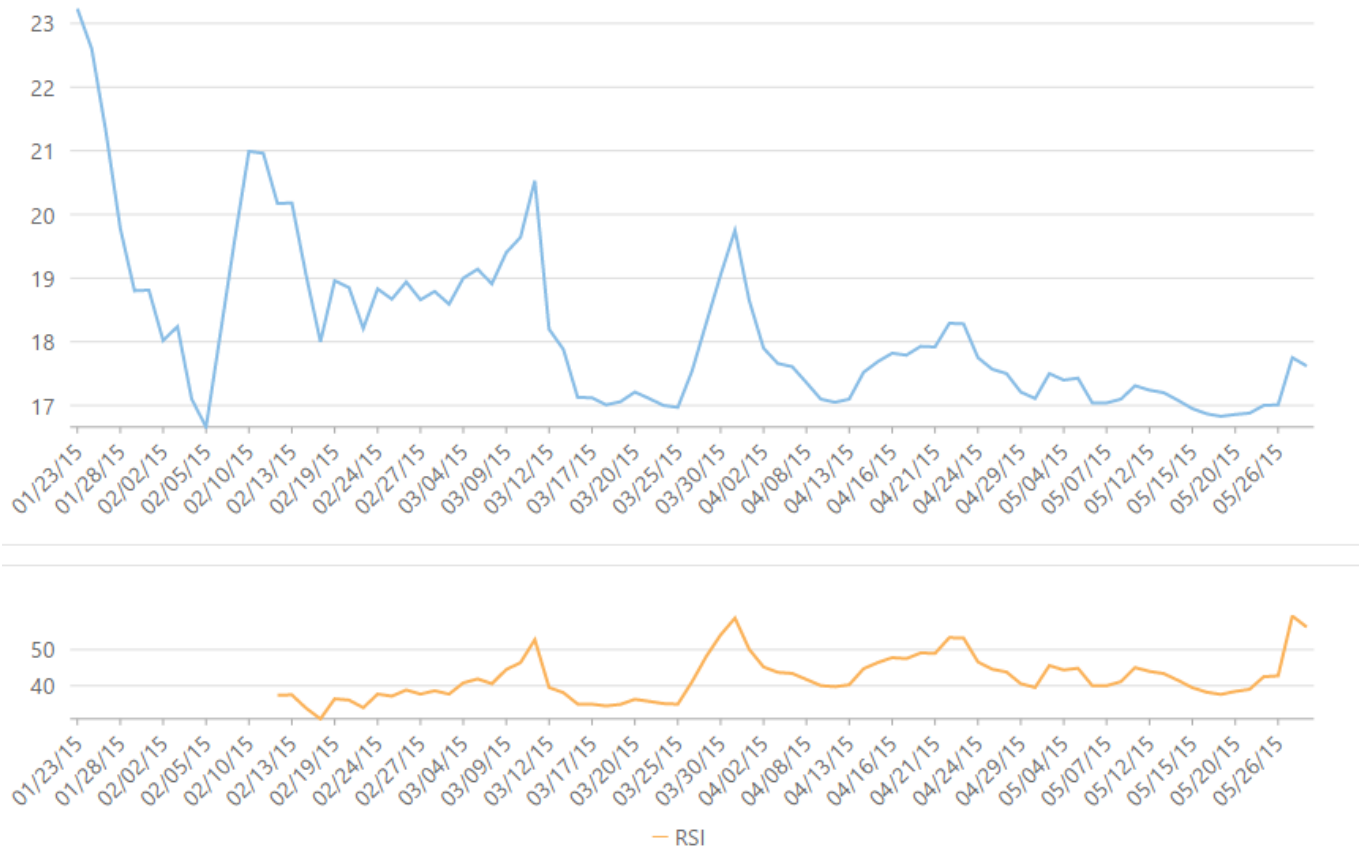
```
            date = ht.item.X ? ht.item.X : null;

            if (wijmo.isDate(date)) {
                date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
            }

            content =
                    '<b>' + ht.name + '</b><br/>' +
                    'Date: ' + date + '<br/>' +
                    'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
            if (ht.item.Volume) {
                content +=
                '<br/>' +
                'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
            }

            return content;
        }
    </script>

               <c1-financial-chart binding-x="X" rendered="chartRendered">
                   <c1-items-source source-collection="Model"></c1-items-source>
                   <c1-financial-chart-series binding="Close"></c1-financial-chart-
series>
                   <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
               </c1-financial-chart>

               <c1-financial-chart height="200px" binding-x="X">
                   <c1-items-source source-collection="Model"></c1-items-source>
                   <c1-flex-chart-cci binding="High,Low,Open,Close" period="20"></c1-
flex-chart-cci>
                   <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
               </c1-financial-chart>
```

Relative Strength Index Indicator

Relative Strength Index (RSI) indicator for FinancialChart is a momentum oscillator, which measures velocity and magnitude of price movements. It compares the upward movements in closing price of an asset to the downward movements over a trading period, and intends to determine strength or weakness of a stock. It fluctuates between 0 and 100. The stocks with strong positive changes have a higher RSI than the stocks with strong negative changes.

It finds application in comparing the magnitude of recent gains to recent losses, to determine the overbought and oversold conditions of an asset. Stocks are considered overbought when RSI is above 70, and oversold when below 30.

The image below shows how FinancialChart appears when the Relative Strength Index indicator is added to the chart, and Period is set to 14.



The following code example demonstrates how to add Relative Strength Index indicator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path,
System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
```

```
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open,
Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean
Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit
organization committed to accelerating the transition to a sustainable world powered
by clean energy, is leveraging the Box platform to power a cloud-based document
submission and review process as part of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom
Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate
in the J.P. Morgan Technology, Media and Telecom Conference in Boston on Monday, May
18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™
Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery
and Fewer Flow Interruptions including Silent Occlusions Infusion Set Developed in
Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX)
has declared a quarterly dividend of 60.0 cents per common share payable on June
...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first
quarter which ended April 30, 2015, following the close of the market on Wednesday,
June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
     "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
```

```
    "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on
Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14.
The analysts believe that the company's dividend ...",
    "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File
Sharing</b>"+
    "<br>Box today announced that it is working with the U.S. Department of Justice
to deliver file sharing and information management to the agency’s workforce. After
rigorous assessment, Box will receive an agency Authorization to Operate this week,
allowing the DOJ to leverage the platform across all its component agencies.",
    "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
    "<br>Box today announced that the ASPCA® , the first humane society established
in North America, is bringing Box’s cloud content sharing and collaboration platform
to its employees."
    };

        return _annotationToolTips;
    }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json

```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23,
"volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6,
"volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3,
"volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78,
"volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8,
"volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81,
"volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02,
"volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24,
"volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1,
"volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66,
"volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12,
"volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6,
"volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99,
"volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96,
```

```
"volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17,
"volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18,
"volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05,
"volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18,
"volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96,
"volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85,
"volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21,
"volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83,
"volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67,
"volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94,
"volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66,
"volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79,
"volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59,
"volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19,
"volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14,
"volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91,
"volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4,
"volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64,
"volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53,
"volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2,
"volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88,
"volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13,
"volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12,
"volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01,
"volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06,
"volume": 536427 },
```

```
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21,
"volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11,
"volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17,
"volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97,
"volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54,
"volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3,
"volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05,
"volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75,
"volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65,
"volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9,
"volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66,
"volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61,
"volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36,
"volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1,
"volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05,
"volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1,
"volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52,
"volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69,
"volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82,
"volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79,
"volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93,
"volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92,
"volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29,
"volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28,
"volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75,
"volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57,
```

```
"volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5,
"volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21,
"volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11,
"volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5,
"volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4,
"volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43,
"volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04,
"volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04,
"volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1,
"volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31,
"volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24,
"volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2,
"volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08,
"volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95,
"volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87,
"volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83,
"volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86,
"volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88,
"volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17,
"volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01,
"volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75,
"volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62,
"volume": 338482 }
]
```

RSIController.cs

C# copyCode

```
// GET: /RSI/
```

```
public ActionResult RSIIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
```

```
)

@(Html.C1().FinancialChart().Height(200)
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.AddRSI().Binding("Close").Period(14).Name("RSI");
})
.Tooltip(t => t.Content("tooltip"))
)
```

Tag Helpers

HTML copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

                <c1-financial-chart binding-x="X" rendered="chartRendered">
                    <c1-items-source source-collection="Model"></c1-items-source>
```

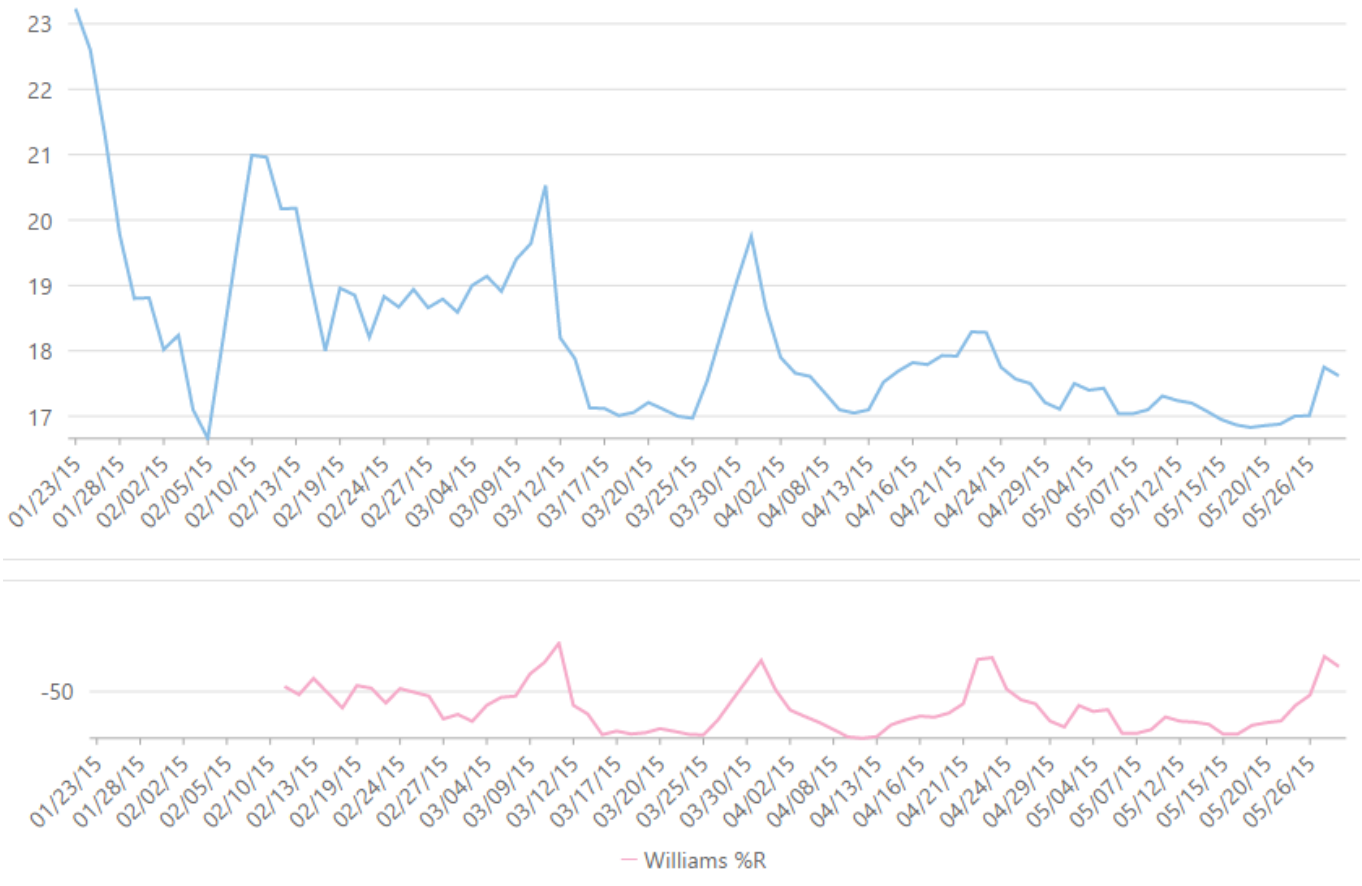
```
                <c1-financial-chart-series binding="Close"></c1-financial-chart-
series>
                <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
            </c1-financial-chart>

            <c1-financial-chart height="200px" binding-x="X">
                <c1-items-source source-collection="Model"></c1-items-source>
                <c1-flex-chart-rsi binding="Close" period="14"></c1-flex-chart-
rsi>
                <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
            </c1-financial-chart>
```

Williams %R Indicator

Williams %R indicator for the FinancialChart is a momentum indicator, which compares the current asset price to the highest price over the look back period. Its look-back is typically 14 periods. The indicator fluctuates between 0 and -100. It is the inverse of a fast Stochastic Oscillator. While the Williams %R displays the level of a stock's close relative to the highest high for the look-back period, the Stochastic Oscillator shows the level of a stock's close relative to the lowest low. Both the indicators show same lines, however scaling is different.

It finds application in determining Overbought/Oversold levels, providing buy and sell signals, and momentum confirmations.

The image below shows how FinancialChart appears when the Williams %R indicator is added to the chart, and Period is set to 14.



The following code example demonstrates how to add Williams %R indicator to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path,
System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open,
Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean
Vehicle Rebate Project</b>"
```

```
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit
organization committed to accelerating the transition to a sustainable world powered
by clean energy, is leveraging the Box platform to power a cloud-based document
submission and review process as part of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom
Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate
in the J.P. Morgan Technology, Media and Telecom Conference in Boston on Monday, May
18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™
Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery
and Fewer Flow Interruptions including Silent Occlusions Infusion Set Developed in
Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX)
has declared a quarterly dividend of 60.0 cents per common share payable on June
...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first
quarter which ended April 30, 2015, following the close of the market on Wednesday,
June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
     "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
     "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on
Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14.
The analysts believe that the company's dividend ...",
     "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File
Sharing</b>"+
     "<br>Box today announced that it is working with the U.S. Department of Justice
to deliver file sharing and information management to the agency’s workforce. After
rigorous assessment, Box will receive an agency Authorization to Operate this week,
allowing the DOJ to leverage the platform across all its component agencies.",
     "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
     "<br>Box today announced that the ASPCA® , the first humane society established
in North America, is bringing Box’s cloud content sharing and collaboration platform
to its employees."
     };

         return _annotationToolTips;
     }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json

```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23,
```

```
"volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6,
"volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3,
"volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78,
"volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8,
"volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81,
"volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02,
"volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24,
"volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1,
"volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66,
"volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12,
"volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6,
"volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99,
"volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96,
"volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17,
"volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18,
"volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05,
"volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18,
"volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96,
"volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85,
"volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21,
"volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83,
"volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67,
"volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94,
"volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66,
"volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79,
"volume": 519605 },
```

```
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59,
"volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19,
"volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14,
"volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91,
"volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4,
"volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64,
"volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53,
"volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2,
"volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88,
"volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13,
"volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12,
"volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01,
"volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06,
"volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21,
"volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11,
"volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17,
"volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97,
"volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54,
"volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3,
"volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05,
"volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75,
"volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65,
"volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9,
"volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66,
"volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61,
"volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36,
```

```
"volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1,
"volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05,
"volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1,
"volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52,
"volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69,
"volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82,
"volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79,
"volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93,
"volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92,
"volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29,
"volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28,
"volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75,
"volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57,
"volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5,
"volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21,
"volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11,
"volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5,
"volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4,
"volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43,
"volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04,
"volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04,
"volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1,
"volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31,
"volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24,
"volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2,
"volume": 244617 },
```

```
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08,
"volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95,
"volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87,
"volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83,
"volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86,
"volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88,
"volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17,
"volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01,
"volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75,
"volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62,
"volume": 338482 }
]
```

WilliamsRController.cs

C# copyCode

```
// GET: /WilliamsR/

public ActionResult WilliamsRIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;
```

```
        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.Add().Binding("Close");
})
.Tooltip(t => t.Content("tooltip"))
)

@(Html.C1().FinancialChart().Height(200)
.Bind(Model)
.BindingX("X")
.Series(sers =>
{
    sers.AddWilliamsR().Binding("High,Low,Open,Close").Period(14);
})
.Tooltip(t => t.Content("tooltip"))
)
```

Tag Helpers

HTML copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
```

```
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

                <c1-financial-chart binding-x="X" rendered="chartRendered">
                    <c1-items-source source-collection="Model"></c1-items-source>
                    <c1-financial-chart-series binding="Close"></c1-financial-chart-
series>
                    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
                </c1-financial-chart>

                <c1-financial-chart height="200px" binding-x="X">
                    <c1-items-source source-collection="Model"></c1-items-source>
                    <c1-flex-chart-williams-r binding="High,Low,Open,Close"
period="14"></c1-flex-chart-williams-r>
                    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
                </c1-financial-chart>
```

Financial Charts Overlays

Overlays are the indicators that use the same scale as price or volume chart data and are, therefore, plotted along with the price bars. MVC Edition supports overlays in its financial chart control, for momentum predictions in financial market. The following sections demonstrate how to use Overlays in Financial Charts for technical analysis.

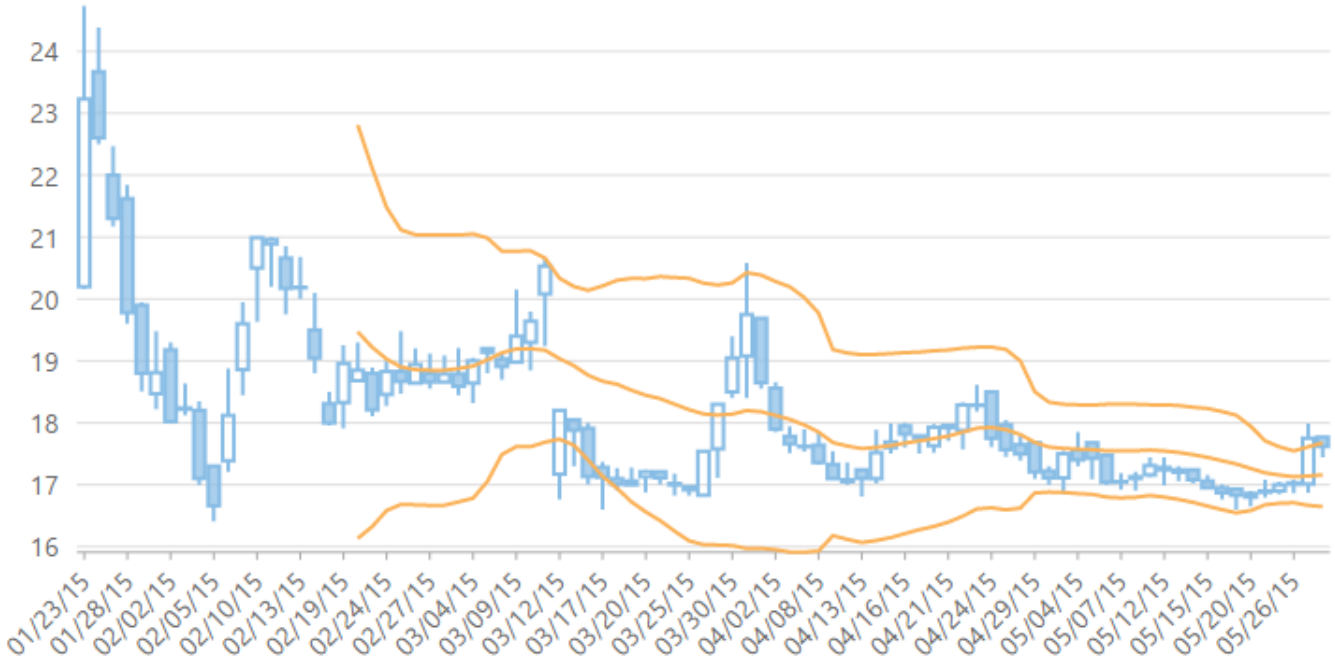
Bollinger Bands Overlay

Bollinger Bands Overlay for the FinancialChart are volatility bands, which display upper and lower limits of normal

price movements. These bands are placed above and below price series. As Volatility is based on Standard Deviation of prices, these bands widen when the Volatility increases and narrow when it decreases.

Bollinger bands are immensely helpful in providing a relative definition of high and low price. Prices plotted near the upper band are considered high, while prices near the lower band are low.

The image below shows how FinancialChart appears when the Fibonacci series Bollinger Bands Overlay are added to the chart, with Period set to 20 and Multiplier set to 2.



The following code example demonstrates how to add Bollinger Bands overlay to the FinancialChart. This example uses data from BoxData.cs model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path,
System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
```

```
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open,
Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean
Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit
organization committed to accelerating the transition to a sustainable world powered
by clean energy, is leveraging the Box platform to power a cloud-based document
submission and review process as part of the Clean Vehicle Rebate Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom
Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate
in the J.P. Morgan Technology, Media and Telecom Conference in Boston on Monday, May
18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™
Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery
and Fewer Flow Interruptions including Silent Occlusions Infusion Set Developed in
Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX)
has declared a quarterly dividend of 60.0 cents per common share payable on June
...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first
quarter which ended April 30, 2015, following the close of the market on Wednesday,
June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
```

```
    "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
    "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on
Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14.
The analysts believe that the company's dividend ...",
    "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File
Sharing</b>"+
    "<br>Box today announced that it is working with the U.S. Department of Justice
to deliver file sharing and information management to the agency's workforce. After
rigorous assessment, Box will receive an agency Authorization to Operate this week,
allowing the DOJ to leverage the platform across all its component agencies.",
    "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
    "<br>Box today announced that the ASPCA® , the first humane society established
in North America, is bringing Box's cloud content sharing and collaboration platform
to its employees."
    };

        return _annotationToolTips;
    }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json

```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23,
"volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6,
"volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3,
"volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78,
"volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8,
"volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81,
"volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02,
"volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24,
"volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1,
"volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66,
"volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12,
"volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6,
"volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99,
"volume": 2804409 },
```

```
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96,
"volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17,
"volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18,
"volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05,
"volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18,
"volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96,
"volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85,
"volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21,
"volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83,
"volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67,
"volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94,
"volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66,
"volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79,
"volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59,
"volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19,
"volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14,
"volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91,
"volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4,
"volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64,
"volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53,
"volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2,
"volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88,
"volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13,
"volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12,
"volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01,
"volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06,
```

```
"volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21,
"volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11,
"volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17,
"volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97,
"volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54,
"volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3,
"volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05,
"volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75,
"volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65,
"volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9,
"volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66,
"volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61,
"volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36,
"volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1,
"volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05,
"volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1,
"volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52,
"volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69,
"volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82,
"volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79,
"volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93,
"volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92,
"volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29,
"volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28,
"volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75,
"volume": 650762 },
```

```
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57,
"volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5,
"volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21,
"volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11,
"volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5,
"volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4,
"volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43,
"volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04,
"volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04,
"volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1,
"volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31,
"volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24,
"volume": 376961 },
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2,
"volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08,
"volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95,
"volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87,
"volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83,
"volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86,
"volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88,
"volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17,
"volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01,
"volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75,
"volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62,
"volume": 338482 }
]
```

BollingerBandsController.cs

C# copyCode

```
// GET: /BollingerBands/
```

```
public ActionResult BBIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.SymbolSize(6)
.ChartType(C1.Web.Mvc.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close");
```

```
        sers.AddBollingerBands().Binding("Close").Period(20).Multiplier(2);
    })
.Tooltip(t => t.Content("tooltip")))
```

Tag Helpers

Example Title copyCode

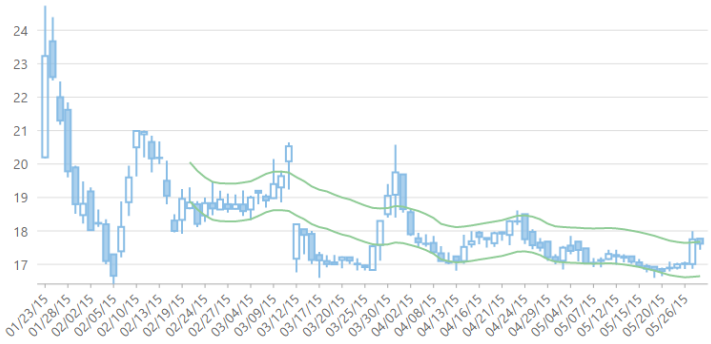
```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

<c1-financial-chart symbol-size="6" binding-x="X" chart-
type="C1.Web.Mvc.Finance.ChartType.Candlestick">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-financial-chart-series binding="High,Low,Open,Close" name="BOX"></c1-
financial-chart-series>
    <c1-flex-chart-bollinger-bands binding="Close" period="20" multiplier="2"
name="Bollinger Bands"></c1-flex-chart-bollinger-bands>
    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
```

```
</c1-financial-chart>
```

Envelopes Overlay

Envelopes overlay represent Moving Average Envelopes overlay series for the FinancialChart. These moving average envelopes are percentage based envelopes that are set above and below a standard moving average. The moving average could be simple or exponential moving average.

The image below shows how FinancialChart appears when Simple Moving Average Envelopes overlay are added to the chart, with Period set to 20 and size set to 3%.



The following code example demonstrates how to add Simple Moving Average Envelopes to the FinancialChart. This example uses data from **BoxData.cs** model, and box.json file.

In Code

Add a BoxData.cs class to the Models folder.

Model

BoxData.cs copyCode

```
public static class BoxData
{
    private static List<FinanceData> _jsonData;
    public static List<FinanceData> GetDataFromJson()
    {
        if (_jsonData != null)
        {
            return _jsonData;
        }

        string path = HttpContext.Current.Server.MapPath("~/Content/box.json");
        string jsonText = new StreamReader(path, System.Text.Encoding.Default).ReadToEnd();
        JArray ja = (JArray)JsonConvert.DeserializeObject(jsonText);
        List<FinanceData> list = new List<FinanceData>();
        foreach (var obj in ja)
        {
            DateTime date = Convert.ToDateTime(obj["date"]);
            double high = Convert.ToDouble(obj["high"].ToString());
            double low = Convert.ToDouble(obj["low"].ToString());
            double open = Convert.ToDouble(obj["open"].ToString());
            double close = Convert.ToDouble(obj["close"].ToString());
            double volume = Convert.ToDouble(obj["volume"].ToString());
            list.Add(new FinanceData { X = date, High = high, Low = low, Open = open, Close = close, Volume = volume });
        }
        _jsonData = list;
        return list;
    }

    private static List<string> _annotationToolTips;
    public static List<string> GetAnnotationTooltips()
    {
        if (_annotationToolTips != null)
        {
            return _annotationToolTips;
        }
        _annotationToolTips = new List<string>{
     "<b>Why the hot IPO market is cooling off?</b>",
     "<b>Tech IPO market healthy?</b>",
     "<b>Center for Sustainable Energy Leverages Box to Power California Clean Vehicle Rebate Project</b>"
     +"<br>Box today announced that the Center for Sustainable Energy, a nonprofit organization committed to accelerating the transition to a sustainable world
powered by clean energy, is leveraging the Box platform to power a cloud-based document submission and review process as part of the Clean Vehicle Rebate
Project.",
     "<b>Box to Present at the J.P. Morgan Technology, Media and Telecom Conference</b>"+
     "<br>Box today announced that Aaron Levie, co-founder and CEO, will participate in the J.P. Morgan Technology, Media and Telecom Conference in Boston on
Monday, May 18, 2015.",
     "<b>BD Receives FDA Clearance for a Novel Infusion Set with BD FlowSmart™ Technology to Enhance the Use of Insulin Pumps</b>"+
     "<br>Unique Side-Ported Catheter Designed to Offer Consistent Insulin Delivery and Fewer Flow Interruptions including Silent Occlusions Infusion Set
Developed in Collaboration with JDRF and Helmsley Charitable ...",
     "<b>Wall Street tech debutant Box dismisses doubters</b>",
     "<b>BD Board Declares Dividend</b>"+
     "<br>The Board of Directors of BD (Becton, Dickinson and Company) (NYSE: BDX) has declared a quarterly dividend of 60.0 cents per common share payable on
June ...",
     "<b>Box Sets Date to Announce First Quarter Fiscal 2016 Financial Results</b>"+
     "<br>Box today announced that it will report financial results for its first quarter which ended April 30, 2015, following the close of the market on
Wednesday, June 10, 2015.",
     "<b>Making Money With Charles Payne: 05/21/15</b>",
     "<b>Best Buy to move sideways: Analyst</b>",
     "<b>Macquarie Upgrades Hercules Technology, Likes Dividend</b>"+
```

```
    "<br>In a report published Tuesday, Macquarie analysts upgraded their rating on Hercules Technology Growth Capital, Inc. (NYSE: HTGX), with a price target of $14. The analysts believe that the company's dividend ...",
    "<b>The U.S. Department of Justice Selects Box to Enable Enterprise File Sharing</b>"+
    "<br>Box today announced that it is working with the U.S. Department of Justice to deliver file sharing and information management to the agency's workforce. After rigorous assessment, Box will receive an agency Authorization to Operate this week, allowing the DOJ to leverage the platform across all its component agencies.",
    "<b>Box Enables Improved Collaboration and Content Sharing at the ASPCA</b>"+
    "<br>Box today announced that the ASPCA® , the first humane society established in North America, is bringing Box's cloud content sharing and collaboration platform to its employees."
    };

        return _annotationToolTips;
    }
}
```

Also add a box.json file to your project, and paste the below content in it.

box.json

box.json


```
[
    { "date": "01/23/15", "open": 20.2, "high": 24.73, "low": 20.16, "close": 23.23, "volume": 42593223 },
    { "date": "01/26/15", "open": 23.67, "high": 24.39, "low": 22.5, "close": 22.6, "volume": 8677164 },
    { "date": "01/27/15", "open": 22, "high": 22.47, "low": 21.17, "close": 21.3, "volume": 3272512 },
    { "date": "01/28/15", "open": 21.62, "high": 21.84, "low": 19.6, "close": 19.78, "volume": 5047364 },
    { "date": "01/29/15", "open": 19.9, "high": 19.95, "low": 18.51, "close": 18.8, "volume": 3419482 },
    { "date": "01/30/15", "open": 18.47, "high": 19.48, "low": 18.22, "close": 18.81, "volume": 2266439 },
    { "date": "02/02/15", "open": 19.18, "high": 19.3, "low": 18.01, "close": 18.02, "volume": 2071168 },
    { "date": "02/03/15", "open": 18.22, "high": 18.64, "low": 18.12, "close": 18.24, "volume": 1587435 },
    { "date": "02/04/15", "open": 18.2, "high": 18.35, "low": 17, "close": 17.1, "volume": 2912224 },
    { "date": "02/05/15", "open": 17.3, "high": 17.31, "low": 16.41, "close": 16.66, "volume": 2682187 },
    { "date": "02/06/15", "open": 17.39, "high": 18.88, "low": 17.21, "close": 18.12, "volume": 3929164 },
    { "date": "02/09/15", "open": 18.86, "high": 19.95, "low": 18.45, "close": 19.6, "volume": 3226650 },
    { "date": "02/10/15", "open": 20.5, "high": 21, "low": 19.63, "close": 20.99, "volume": 2804409 },
    { "date": "02/11/15", "open": 20.89, "high": 21, "low": 20.2, "close": 20.96, "volume": 1698365 },
    { "date": "02/12/15", "open": 20.66, "high": 20.85, "low": 19.75, "close": 20.17, "volume": 1370320 },
    { "date": "02/13/15", "open": 20.19, "high": 20.68, "low": 20, "close": 20.18, "volume": 711951 },
    { "date": "02/17/15", "open": 19.5, "high": 20.1, "low": 18.8, "close": 19.05, "volume": 2093602 },
    { "date": "02/18/15", "open": 18.31, "high": 18.5, "low": 17.96, "close": 18, "volume": 1849490 },
    { "date": "02/19/15", "open": 18.33, "high": 19.25, "low": 17.91, "close": 18.96, "volume": 1311518 },
    { "date": "02/20/15", "open": 18.68, "high": 19.3, "low": 18.65, "close": 18.85, "volume": 1001692 },
    { "date": "02/23/15", "open": 18.8, "high": 18.89, "low": 18.11, "close": 18.21, "volume": 670087 },
    { "date": "02/24/15", "open": 18.46, "high": 19, "low": 18.27, "close": 18.83, "volume": 759263 },
    { "date": "02/25/15", "open": 18.83, "high": 19.48, "low": 18.47, "close": 18.67, "volume": 915580 },
    { "date": "02/26/15", "open": 18.64, "high": 19.2, "low": 18.64, "close": 18.94, "volume": 461283 },
    { "date": "02/27/15", "open": 18.8, "high": 19.12, "low": 18.55, "close": 18.66, "volume": 617199 },
    { "date": "03/02/15", "open": 18.66, "high": 19.09, "low": 18.65, "close": 18.79, "volume": 519605 },
    { "date": "03/03/15", "open": 18.79, "high": 19.21, "low": 18.45, "close": 18.59, "volume": 832415 },
    { "date": "03/04/15", "open": 18.64, "high": 19.05, "low": 18.32, "close": 19, "volume": 539688 },
    { "date": "03/05/15", "open": 19.2, "high": 19.2, "low": 18.8, "close": 19.14, "volume": 486149 },
    { "date": "03/06/15", "open": 19.03, "high": 19.1, "low": 18.7, "close": 18.91, "volume": 685659 },
    { "date": "03/09/15", "open": 18.98, "high": 20.15, "low": 18.96, "close": 19.4, "volume": 1321363 },
    { "date": "03/10/15", "open": 19.3, "high": 19.8, "low": 18.85, "close": 19.64, "volume": 615743 },
    { "date": "03/11/15", "open": 20.08, "high": 20.65, "low": 19.24, "close": 20.53, "volume": 2167167 },
    { "date": "03/12/15", "open": 17.17, "high": 18.2, "low": 16.76, "close": 18.2, "volume": 6837638 },
    { "date": "03/13/15", "open": 18.05, "high": 18.05, "low": 17.3, "close": 17.88, "volume": 1715629 },
    { "date": "03/16/15", "open": 17.91, "high": 18, "low": 17.01, "close": 17.13, "volume": 1321313 },
    { "date": "03/17/15", "open": 17.28, "high": 17.37, "low": 16.6, "close": 17.12, "volume": 1272242 },
    { "date": "03/18/15", "open": 17.1, "high": 17.27, "low": 16.91, "close": 17.01, "volume": 530063 },
    { "date": "03/19/15", "open": 17, "high": 17.28, "low": 17, "close": 17.06, "volume": 536427 },
    { "date": "03/20/15", "open": 17.13, "high": 17.24, "low": 16.88, "close": 17.21, "volume": 1320237 },
    { "date": "03/23/15", "open": 17.21, "high": 17.23, "low": 17.01, "close": 17.11, "volume": 509798 },
    { "date": "03/24/15", "open": 17.02, "high": 17.18, "low": 16.82, "close": 17, "volume": 962149 },
    { "date": "03/25/15", "open": 16.92, "high": 16.99, "low": 16.82, "close": 16.97, "volume": 565673 },
    { "date": "03/26/15", "open": 16.83, "high": 17.56, "low": 16.83, "close": 17.54, "volume": 884523 },
    { "date": "03/27/15", "open": 17.58, "high": 18.3, "low": 17.11, "close": 18.3, "volume": 705626 },
    { "date": "03/30/15", "open": 18.5, "high": 19.4, "low": 18.4, "close": 19.05, "volume": 1151620 },
    { "date": "03/31/15", "open": 19.08, "high": 20.58, "low": 18.4, "close": 19.75, "volume": 2020679 },
    { "date": "04/01/15", "open": 19.69, "high": 19.69, "low": 18.55, "close": 18.65, "volume": 961078 },
    { "date": "04/02/15", "open": 18.56, "high": 18.66, "low": 17.85, "close": 17.9, "volume": 884233 },
    { "date": "04/06/15", "open": 17.78, "high": 17.94, "low": 17.51, "close": 17.66, "volume": 605252 },
    { "date": "04/07/15", "open": 17.62, "high": 17.9, "low": 17.53, "close": 17.61, "volume": 591988 },
    { "date": "04/08/15", "open": 17.64, "high": 17.85, "low": 17.32, "close": 17.36, "volume": 618855 },
    { "date": "04/09/15", "open": 17.33, "high": 17.54, "low": 17.1, "close": 17.1, "volume": 761855 },
    { "date": "04/10/15", "open": 17.08, "high": 17.36, "low": 17, "close": 17.05, "volume": 568373 },
    { "date": "04/13/15", "open": 17.24, "high": 17.26, "low": 16.81, "close": 17.1, "volume": 667142 },
    { "date": "04/14/15", "open": 17.1, "high": 17.89, "low": 17.02, "close": 17.52, "volume": 870138 },
    { "date": "04/15/15", "open": 17.6, "high": 17.99, "low": 17.5, "close": 17.69, "volume": 530456 },
    { "date": "04/16/15", "open": 17.95, "high": 18, "low": 17.6, "close": 17.82, "volume": 548730 },
    { "date": "04/17/15", "open": 17.75, "high": 17.79, "low": 17.5, "close": 17.79, "volume": 446373 },
    { "date": "04/20/15", "open": 17.63, "high": 17.98, "low": 17.52, "close": 17.93, "volume": 487017 },
    { "date": "04/21/15", "open": 17.96, "high": 17.98, "low": 17.71, "close": 17.92, "volume": 320302 },
    { "date": "04/22/15", "open": 17.88, "high": 18.33, "low": 17.57, "close": 18.29, "volume": 644812 },
    { "date": "04/23/15", "open": 18.29, "high": 18.61, "low": 18.18, "close": 18.28, "volume": 563879 },
    { "date": "04/24/15", "open": 18.5, "high": 18.5, "low": 17.61, "close": 17.75, "volume": 650762 },
    { "date": "04/27/15", "open": 17.97, "high": 18.05, "low": 17.45, "close": 17.57, "volume": 437294 },
    { "date": "04/28/15", "open": 17.65, "high": 17.79, "low": 17.39, "close": 17.5, "volume": 224519 },
    { "date": "04/29/15", "open": 17.68, "high": 17.68, "low": 17.1, "close": 17.21, "volume": 495706 },
    { "date": "04/30/15", "open": 17.22, "high": 17.3, "low": 17, "close": 17.11, "volume": 391040 },
    { "date": "05/01/15", "open": 17.11, "high": 17.55, "low": 16.85, "close": 17.5, "volume": 563075 },
    { "date": "05/04/15", "open": 17.56, "high": 17.85, "low": 17.3, "close": 17.4, "volume": 253138 },
    { "date": "05/05/15", "open": 17.68, "high": 17.68, "low": 17.09, "close": 17.43, "volume": 290935 },
    { "date": "05/06/15", "open": 17.48, "high": 17.48, "low": 17, "close": 17.04, "volume": 313662 },
    { "date": "05/07/15", "open": 17.05, "high": 17.19, "low": 16.92, "close": 17.04, "volume": 360284 },
    { "date": "05/08/15", "open": 17.13, "high": 17.21, "low": 16.91, "close": 17.1, "volume": 297653 },
    { "date": "05/11/15", "open": 17.16, "high": 17.44, "low": 17.13, "close": 17.31, "volume": 268504 },
    { "date": "05/12/15", "open": 17.28, "high": 17.44, "low": 16.99, "close": 17.24, "volume": 376961 },
```

```
    { "date": "05/13/15", "open": 17.24, "high": 17.3, "low": 17.06, "close": 17.2, "volume": 244617 },
    { "date": "05/14/15", "open": 17.24, "high": 17.25, "low": 17.02, "close": 17.08, "volume": 252526 },
    { "date": "05/15/15", "open": 17.06, "high": 17.16, "low": 16.95, "close": 16.95, "volume": 274783 },
    { "date": "05/18/15", "open": 16.95, "high": 17.01, "low": 16.76, "close": 16.87, "volume": 418513 },
    { "date": "05/19/15", "open": 16.93, "high": 16.94, "low": 16.6, "close": 16.83, "volume": 367660 },
    { "date": "05/20/15", "open": 16.8, "high": 16.9, "low": 16.65, "close": 16.86, "volume": 297914 },
    { "date": "05/21/15", "open": 16.9, "high": 17.08, "low": 16.79, "close": 16.88, "volume": 229346 },
    { "date": "05/22/15", "open": 16.9, "high": 17.05, "low": 16.85, "close": 17, "volume": 253279 },
    { "date": "05/26/15", "open": 17.03, "high": 17.08, "low": 16.86, "close": 17.01, "volume": 212640 },
    { "date": "05/27/15", "open": 17.01, "high": 17.99, "low": 16.87, "close": 17.75, "volume": 857109 },
    { "date": "05/28/15", "open": 17.77, "high": 17.77, "low": 17.44, "close": 17.62, "volume": 338482 }
]
```

EnvelopesController.cs

C# copyCode

```
// GET: /Envelopes/

public ActionResult EnvelopesIndex()
{
    var model = BoxData.GetDataFromJson();
    return View(model);
}
```

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

@(Html.C1().FinancialChart()
.Bind(Model)
.BindingX("X")
.SymbolSize(6)
.ChartType(C1.Web.Mvc.Finance.ChartType.Candlestick)
.Series(sers =>
    {
        sers.Add().Binding("High,Low,Open,Close");

sers.AddEnvelopes().Binding("Close").Period(20).Size(0.03).Type(C1.Web.Mvc.Finance.MovingAverageType.Simple).Visibility(C1.Web.Mvc.Chart.SeriesVisibility.Hidden);
    })
.Tooltip(t => t.Content("tooltip")))
```

Tag Helpers

Example Title copyCode

```
@using MVCFinancialChart.Models

@model List<FinanceData>

<script>
    // function used for displaying XYV data in tooltips
    function tooltip(ht) {
        var date, content;

        if (!ht || !ht.item) {
            return '';
        }

        date = ht.item.X ? ht.item.X : null;

        if (wijmo.isDate(date)) {
            date = wijmo.Globalize.formatDate(date, 'MM/dd/yy');
```

```
        }

        content =
                '<b>' + ht.name + '</b><br/>' +
                'Date: ' + date + '<br/>' +
                'Y: ' + wijmo.Globalize.format(ht.y, 'n2');
        if (ht.item.Volume) {
            content +=
            '<br/>' +
            'Volume: ' + wijmo.Globalize.format(ht.item.Volume, 'n0');
        }

        return content;
    }
</script>

<c1-financial-chart symbol-size="6" binding-x="X" chart-type="C1.Web.Mvc.Finance.ChartType.Candlestick">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-financial-chart-series binding="High,Low,Open,Close" name="BOX"></c1-financial-chart-series>
    <c1-flex-chart-envelopes binding="Close" period="20" size="0.03" type="C1.Web.Mvc.Finance.MovingAverageType.Simple" name="Envelopes"></c1-flex-chart-
envelopes>
    <c1-flex-chart-tooltip content="tooltip"></c1-flex-chart-tooltip>
</c1-financial-chart>
```

FinancialChart ASP.NET Core Tags

FlexSheet control supports the following ASP.NET Core Tags:

<c1-financial-chart/>

- AxisX
- AxisY
- binding
- binding-x
- chart-type
- class
- style
- DataLabel
- disabled
- footer
- FooterStyle
- got-focus
- header
- HeaderStyle
- height
- id
- interpolate-nulls
- item-formatter
- kagi-fields
- kagi-range-mode
- kagi-reversal-amount
- ItemsSource
- legend-position
- legend-toggle
- line-break-new-line-breaks
- lost-focus
- palette
- plot-margin
- rendered
- rendering
- renko-box-size
- renko-fields
- renko-range-mode

- selection-changed
- selection-index
- selection-mode
- Series
- series-visibility-changed
- symbol-size
- Tooltip
- width

<c1-financial-chart-series/>

- <c1-flex-chart-fibonacci/>
- <c1-flex-chart-fibonacci-arcs/>
- <c1-flex-chart-fibonacci-fans/>
- <c1-flex-chart-fibonacci-time-zones/>
- <c1-flex-chart-atr/>
- <c1-flex-chart-rsi/>
- <c1-flex-chart-cci/>
- <c1-flex-chart-williams-r/>
- <c1-flex-chart-bollinger-bands/>
- <c1-flex-chart-envelopes/>
- <c1-flex-chart-macd/>
- <c1-flex-chart-macd-histogram/>
- <c1-flex-chart-stochastic/>

c1-flex-chart-line-style

<c1-flex-chart-fibonacci/>

- stroke
- stroke-width
- c1-property

c1-flex-chart-axis

- c1-property
- position
- major-grid
- axis-line
- binding
- format
- item-formatter
- label-align
- label-angle
- labels
- log-base
- major-unit
- max
- min
- minor-grid
- minor-unit
- name
- origin
- range-changed
- reversed

- title

c1-flex-chart-datalabel

- content
- offset
- position
- rendering

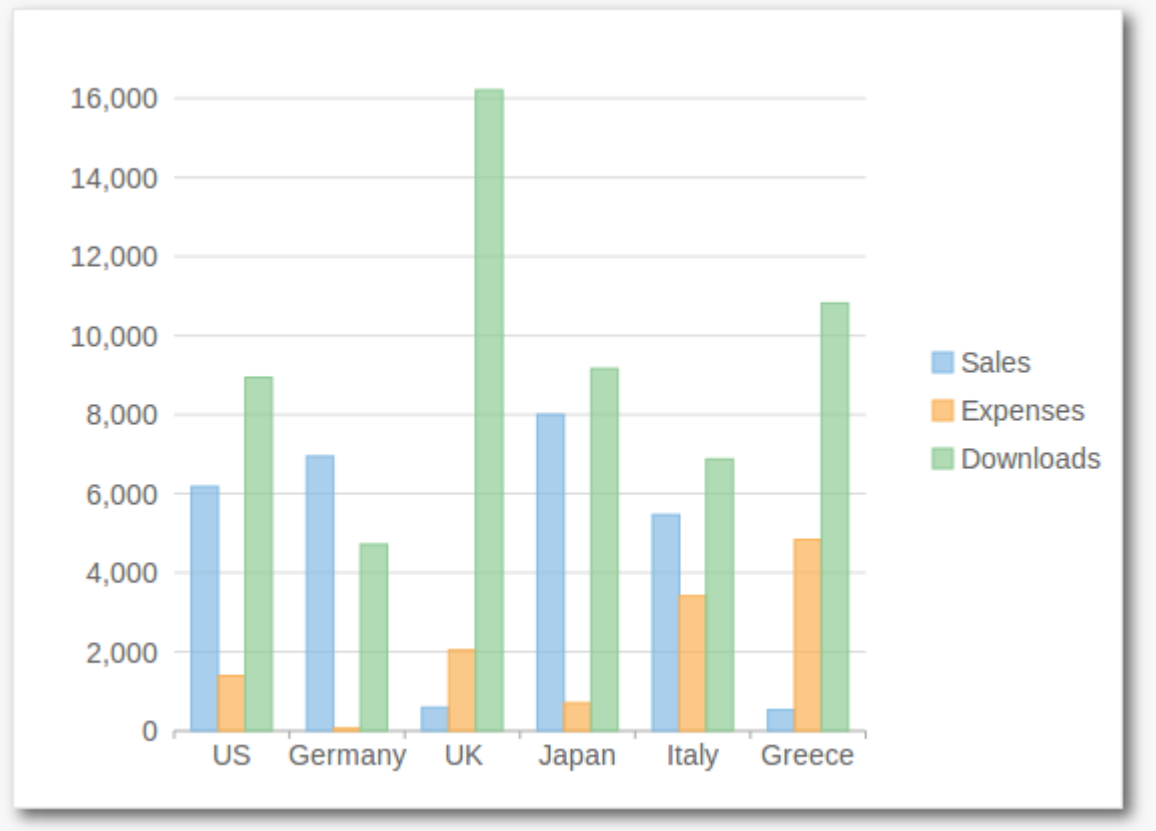
c1-flex-chart-title-style

- fill
- font-family
- font-size
- foreground
- halign
- c1-property

FlexChart

The FlexChart control allows you to represent data visually in web applications. Depending on the type of data you need to display, you can represent your data as bars, columns, bubbles, candlesticks, lines, scattered points, or even display them in multiple chart types.

FlexChart manages the underlying complexities inherent in a chart control completely, allowing developers to concentrate on important application specific tasks.



Key Features

- **Chart Type**: Change a line chart to a bar chart or any other chart type by setting a single property. FlexChart supports twelve different chart types.
- **Tooltips**: Display chart values using tooltips.
- **Multiple Series**: Add multiple series on a single chart.
- **Header and Footer**: Use simple properties to set a title and footer text.
- **Legend**: Change position of the legend as needed.

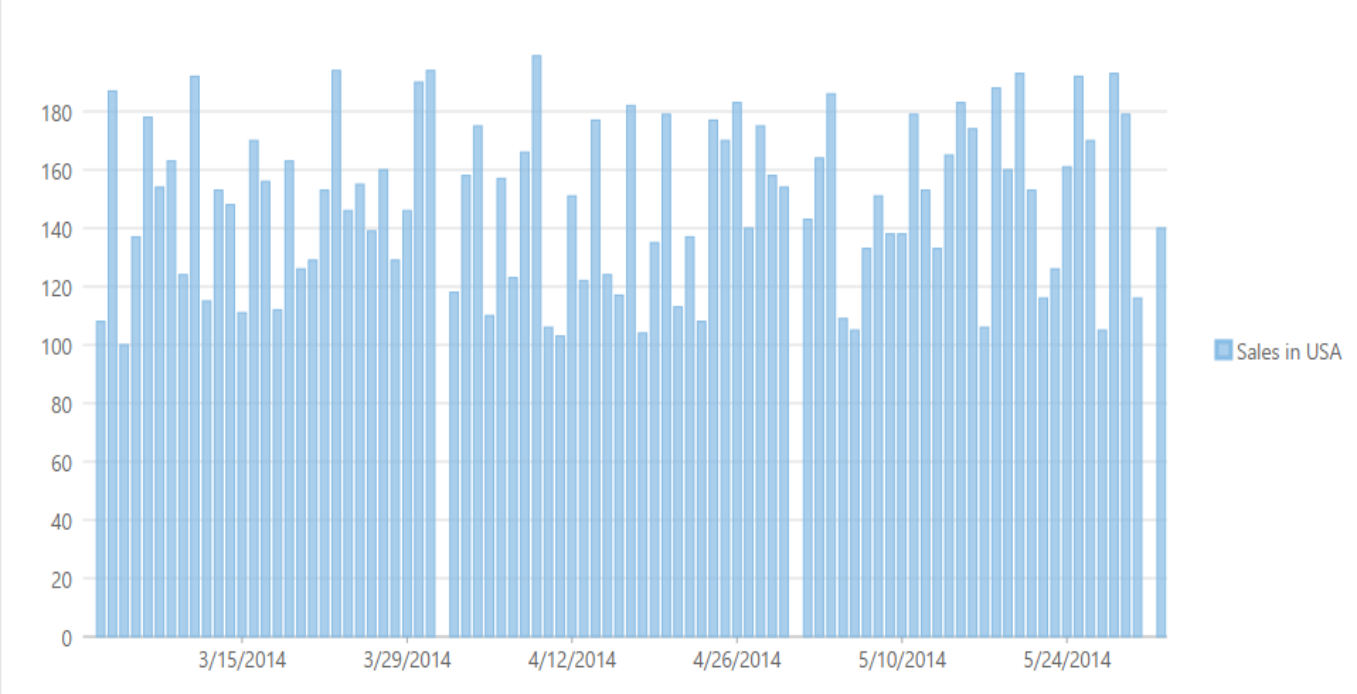
Quick Start: Add Data to FlexChart

This section describes how to add a FlexChart control to your MVC web application and add data to it.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for FlexChart**
- **Step 3: Add a FlexChart control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for FlexChart

1. Add a new class to the folder **Models** (for example: `FlexChartDataSource.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes that serve as a datasource for the FlexChart control.

C#

Index.cshtml copyCode

```
public class FlexChartDataSource
{
    public string Name { get; set; }

    public int MarPrice { get; set; }
    public int AprPrice { get; set; }
    public int MayPrice { get; set; }

    private IEnumerable<FruitSale> _sales = null;
    public IEnumerable<FruitSale> Sales
    {
        get
        {
            if (_sales == null)
            {
                _sales = GetSales();
            }
            return _sales;
        }
    }

    public static IEnumerable<FlexChartDataSource> GetFruitsSales()
    {
        var rand = new Random(0);
        var fruits = new[] { "Oranges", "Apples", "Pears", "Bananas",
"Pineapples" };
        var list = fruits.Select((f, i) =>
        {
            int mar = rand.Next(1, 6);
            int apr = rand.Next(1, 9);
            int may = rand.Next(1, 6);
            return new FlexChartDataSource { Name = f, MarPrice = mar, AprPrice
= apr, MayPrice = may };
        });

        return list;
    }

    private IEnumerable<FruitSale> GetSales()
    {
        var rand = new Random(0);
        var today = DateTime.Now.Date;
        var firstDay = new DateTime(today.Year - 1, 3, 1);
        var dataTimes = new List<DateTime>();
        for (int i = 0; i < 92; i++)
        {
            dataTimes.Add(firstDay.AddDays(i + 1));
        }
        var list = dataTimes.Select((date, i) =>
```

```
        {
            FruitSale sale = new FruitSale { Date = date };
            sale.SalesInChina = rand.Next(150, 250);
            if (i % 30 != 0)
            {
                sale.SalesInUSA = rand.Next(100, 200);
                sale.SalesInJapan = rand.Next(0, 100);
            }
            else
            {
                sale.SalesInUSA = null;
                sale.SalesInJapan = null;
            }

            return sale;
        });

        return list;
    }
}

public class FruitSale
{
    public DateTime Date { get; set; }
    public int? SalesInUSA { get; set; }
    public int? SalesInChina { get; set; }
    public int? SalesInJapan { get; set; }
}
```

VB

Index.vbhtml

```
Public Class FlexChartDataSource
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String

    Public Property MarPrice() As Integer
        Get
            Return m_MarPrice
        End Get
        Set
            m_MarPrice = Value
        End Set
```

```
    End Property
    Private m_MarPrice As Integer
    Public Property AprPrice() As Integer
        Get
            Return m_AprPrice
        End Get
        Set
            m_AprPrice = Value
        End Set
    End Property
    Private m_AprPrice As Integer
    Public Property MayPrice() As Integer
        Get
            Return m_MayPrice
        End Get
        Set
            m_MayPrice = Value
        End Set
    End Property
    Private m_MayPrice As Integer

    Private _sales As IEnumerable(Of FruitSale) = Nothing
    Public ReadOnly Property Sales() As IEnumerable(Of FruitSale)
        Get
            If _sales Is Nothing Then
                _sales = GetSales()
            End If
            Return _sales
        End Get
    End Property

    Public Shared Function GetFruitsSales() As IEnumerable(Of
FlexChartDataSource)
        Dim rand = New Random(0)
        Dim fruits = New String() {"Oranges", "Apples", "Pears", "Bananas",
"Pineapples"}
        Dim list = fruits.[Select](Function(f, i)
        Dim mar As Integer = rand.[Next](1, 6)
        Dim apr As Integer = rand.[Next](1, 9)
        Dim may As Integer = rand.[Next](1, 6)
        Return New FlexChartDataSource() With {
                Key.Name = f,
            Key.MarPrice = mar,
            Key.AprPrice = apr,
            Key.MayPrice = may
            }
End Function)

        Return list
    End Function
```

```
    Private Function GetSales() As IEnumerable(Of FruitSale)
        Dim rand = New Random(0)
        Dim today = DateTime.Now.[Date]
        Dim firstDay = New DateTime(today.Year - 1, 3, 1)
        Dim dataTimes = New List(Of DateTime)()
        For i As Integer = 0 To 91
            dataTimes.Add(firstDay.AddDays(i + 1))
        Next
        Dim list = dataTimes.[Select](Function([date], i)
        Dim sale As New FruitSale() With {
        Key.[Date] = [date]
        }
        sale.SalesInChina = rand.[Next](150, 250)
                If i Mod 30 <> 0 Then
        sale.SalesInUSA = rand.[Next](100, 200)
        sale.SalesInJapan = rand.[Next](0, 100)
                Else
        sale.SalesInUSA = Nothing
        sale.SalesInJapan = Nothing
                End If
        Return sale
        End Function)
        Return list
    End Function
End Class

Public Class FruitSale
    Public Property [Date]() As DateTime
        Get
            Return m_Date
        End Get
        Set
            m_Date = Value
        End Set
    End Property
    Private m_Date As DateTime
    Public Property SalesInUSA() As System.Nullable(Of Integer)
        Get
            Return m_SalesInUSA
        End Get
        Set
            m_SalesInUSA = Value
        End Set
    End Property
    Private m_SalesInUSA As System.Nullable(Of Integer)
    Public Property SalesInChina() As System.Nullable(Of Integer)
        Get
            Return m_SalesInChina
        End Get
        Set
            m_SalesInChina = Value
```

```
        End Set
    End Property
    Private m_SalesInChina As System.Nullable(Of Integer)
    Public Property SalesInJapan() As System.Nullable(Of Integer)
        Get
            Return m_SalesInJapan
        End Get
        Set
            m_SalesInJapan = Value
        End Set
    End Property
    Private m_SalesInJapan As System.Nullable(Of Integer)
End Class
```

Back to Top

Step 3: Add a FlexChart control

Complete the following steps to initialize a FlexChart control.

Add a new Controller

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: `QuickStartController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method `Index()` with the following method.

C#

C# copyCode

```
public ActionResult QuickStart()
{
  // Set DataSource
  FlexChartDataSource ds = new FlexChartDataSource();
  return View(ds.Sales);
}
```

VB

VB

```
Public Function QuickStart() As ActionResult
        ' Set DataSource
```

```
        Dim ds As New FlexChartDataSource()
        Return View(ds.Sales)
End Function
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller `QuickStartController` to open it.
2. Place the cursor inside the method `QuickStart()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **QuickStart** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.
6. Instantiate a FlexChart control in the view QuickStart as shown below.

HTML Helpers

Razor copyCode

```
@using MvcApplication1.Models
@model IEnumerable<FruitSale>
@(Html.C1().FlexChart()
.Bind("Date", Model)
.ChartType(C1.Web.Mvc.Chart.ChartType.Column)
.Series(sers =>
  {
      sers.Add()
     .Binding("SalesInUSA")
     .Name("Sales in USA");
  })
)
```

Index.vbhtml

```
@Imports C1.Web.Mvc
@Imports C1.Web.Mvc.Chart
@Imports System.Drawing

@ModelType IEnumerable(Of FruitSale)

@(Html.C1().FlexChart() _
    .Bind("Date", Model) _
    .ChartType(C1.Web.Mvc.Chart.ChartType.Column) _
    .Series(Sub(ser)
                ser.Add() _
                .Binding("SalesInUSA") _
                .Name("Sales in USA")
        End Sub) _)
```

Tag Helpers

HTML copyCode

```
<c1-flex-chart binding-x="Name"chart-type="ChartType.Column" >
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA">
    </c1-flex-chart-series>
</c1-flex-chart>
```

Back to Top

Step 4: Build and Run the Project

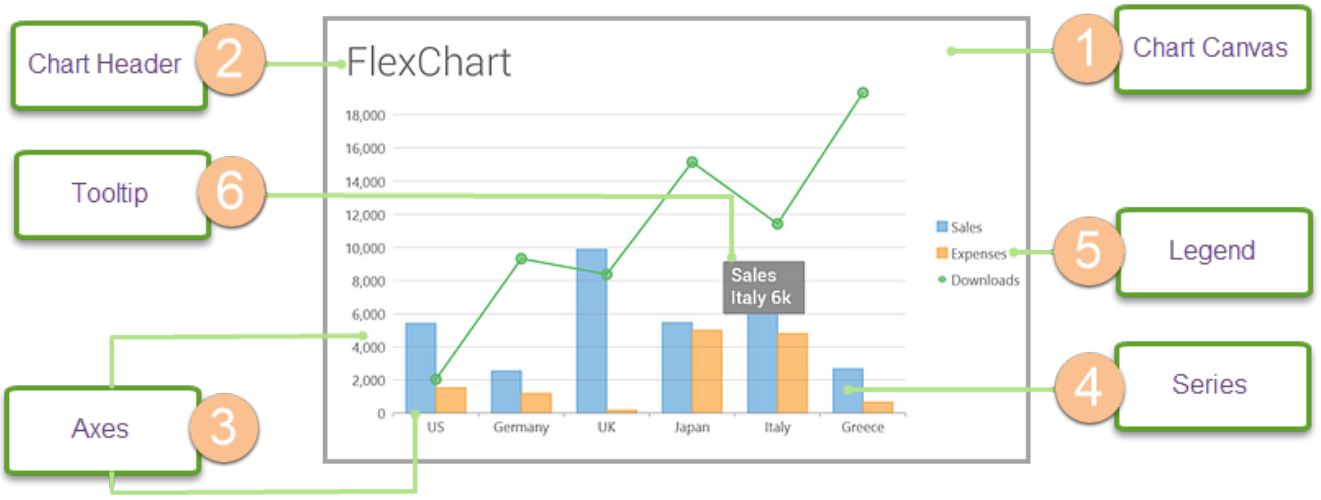
1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

> Append the folder name and view name to the generated URL (for example: http://localhost:1234/**FlexChart/Index**) in the address bar of the browser to see the view.

Back to Top

Chart Elements

FlexChart is composed of several elements as shown below:



1	**Chart Canvas**	Area on which all chart elements including the series, axes title and legend are placed.
2	**Chart Header**	Text that you want to display at the top of your chart, basically a title that serves as a heading for your chart.
3	**Axes**	Two primary axes, X and Y. Although in some cases you may add secondary axes as well.
4	**Series**	Collection of data that is plotted on the chart.
5	**Legend**	Name of the series added in the chart along with predefined symbols and colors used to plot data for that series.

6	**Tooltip**	Tooltips or labels that appear when you hover on a series.

Chart Types

You can change the type of the FlexChart control depending on your requirement. Chart type can be changed by setting the ChartType property of the FlexChart control. In this case, if multiple series are added to the FlexChart, all of them are in of the same chart type. To know how to add multiple series and to set a different ChartType of each series, see Mixed Charts.

FlexChart supports the following chart types.

Area Chart	**Bar and Column Chart**	**Bubble Chart**	**Candlestick Chart**
Funnel Chart	**HighLowOpenClose Chart**	**Line and LineSymbols chart**	**Scatter Chart**
Spline and SplineSymbols chart	**SplineArea chart**		

In Code

HTML Helpers

Razor copyCode

```
.ChartType(C1.Web.Mvc.Chart.ChartType.Bar)
```

Tag Helpers

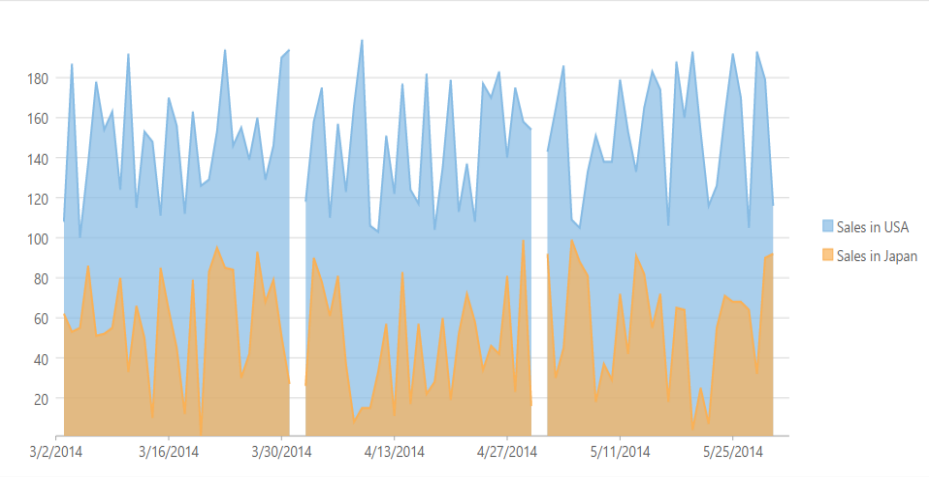
HTML copyCode

```
<c1-flex-chart chart-type="ChartType.Bar"></c1-flex-chart>
```

Area chart

An Area chart draws each series as connected points of data and the area below the connected points is filled with color to denote volume. Each new series is drawn on top of the preceding series. The series can either be drawn independently or stacked.

These charts are commonly used to show trends between associated attributes over time.

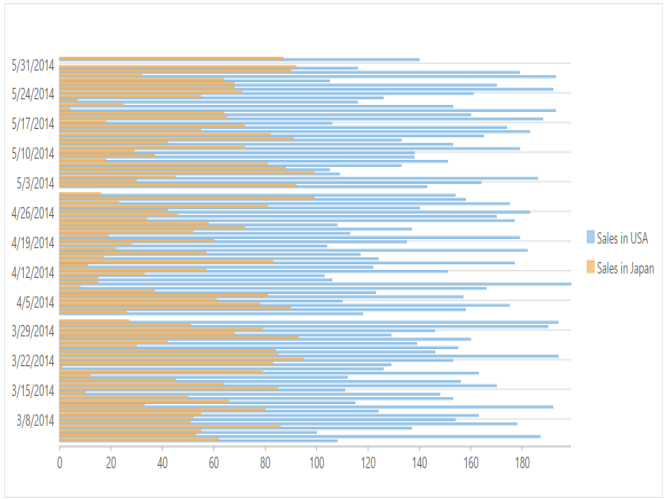


Back to Top

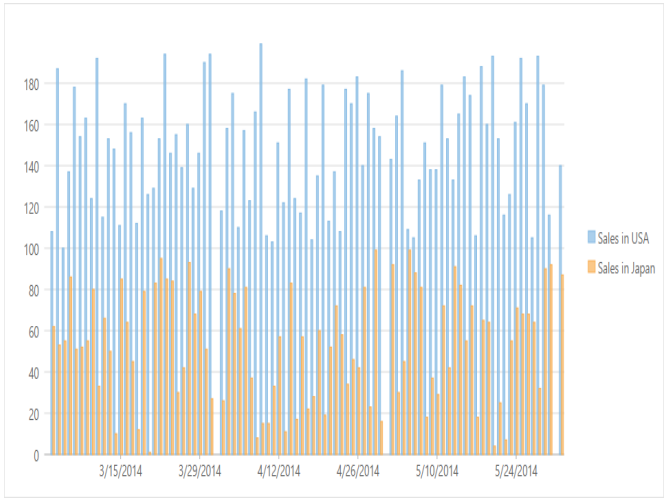
Bar and Column chart

A Bar chart or a Column chart represents each series in the form of bars of the same color and width, whose length is determined by its value. Each new series is plotted in the form of bars next to the bars of the preceding series. When the bars are arranged horizontally, the chart is called a bar chart and when the bars are arranged vertically, the chart is called column chart. Bar charts and Column charts can be either grouped or stacked.

These charts are commonly used to visually represent data that is grouped into discrete categories, for example: age groups, months, etc.



Bar Chart



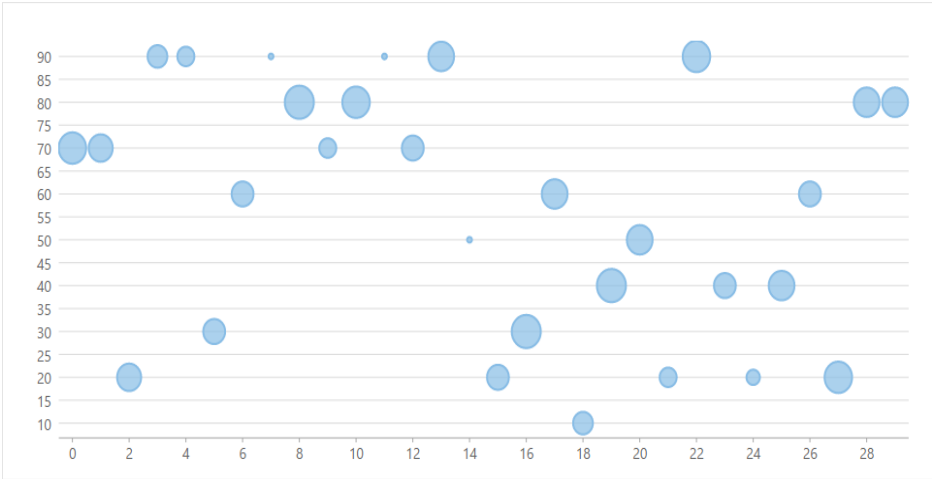
Column Chart

Back to Top

Bubble chart

A Bubble chart represents three dimensions of data. The X and Y values denote two of the data dimensions. The third dimension is denoted by the size of the bubble.

These charts are used to compare entities based on their relative positions on the axis as well as their size.



Back to Top

Candlestick chart

A Candlestick chart is a financial chart that shows the opening, closing, high and low prices of a given stock. It is a special type of HiLoOpenClose chart that is used to show the relationship between open and close as well as high and low. Candle chart uses price data (high, low, open, and close values) and it includes a thick candle-like body that uses the color and size of the body to reveal additional information about the relationship between the open and close values. for example:, long transparent candles show buying pressure and long filled candles show selling pressure.

Elements of a Candlestick chart

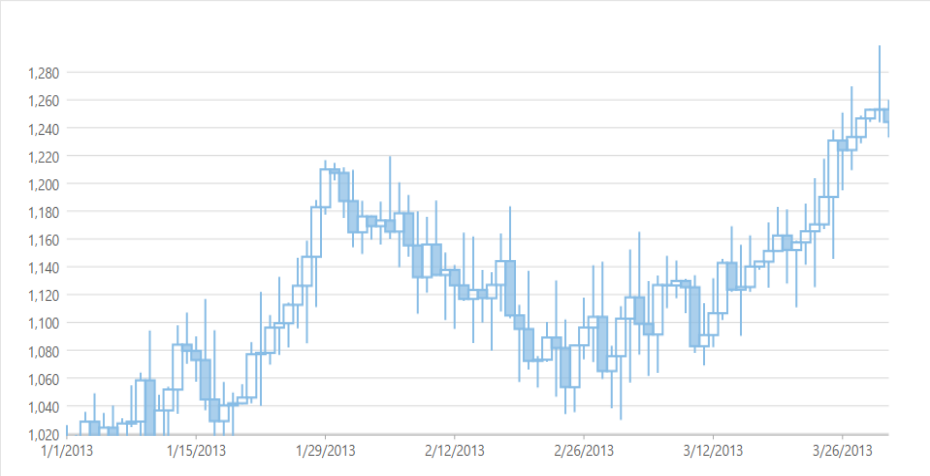
The Candlestick chart is made up of the following elements: **candle**, **wick**, and **tail**.

- **Candle**: The candle or the body (the solid bar between the opening and closing values) represents the change in stock price from opening to closing.
- **Wick and Tail**: The thin lines, wick and tail, above and below the candle depict the high/low range.
- **Hollow Body:** A hollow candle or transparent candle indicates a rising stock price (close was higher than open). In a hollow candle, the bottom of the body represents the opening price and the top of the body represents the closing price.
- **Filled Body:** A filled candle indicates a falling stock price (open was higher than close). In a filled candle the top of the body represents the opening price and the bottom of the body represents the closing price.

In a Candlestick there are five values for each data point in the series.

- **x:** Determines the date position along the x axis.
- **high:** Determines the highest price for the day, and plots it as the top of the candle along the y axis.
- **low:** Determines the lowest price for the day, and plots it as the bottom of the candle along the y axis.
- **open:** Determines the opening price for the day.
- **close:** Determines the closing price for the day.

The following image shows a candlestick chart displaying stock prices.



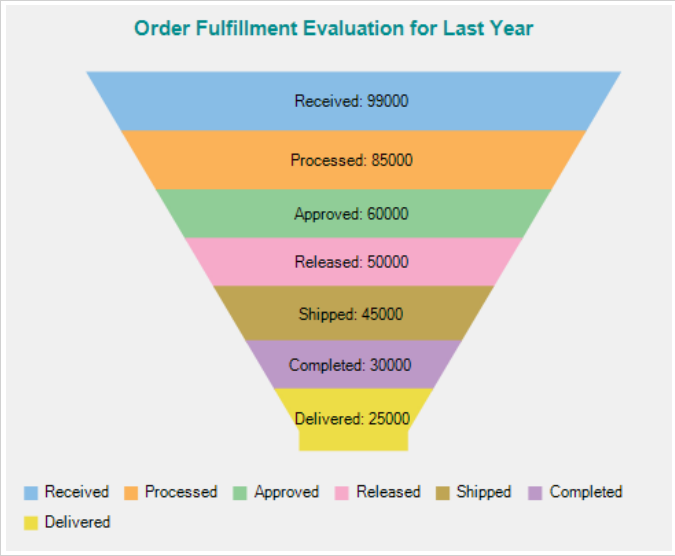
Back to Top

Funnel chart

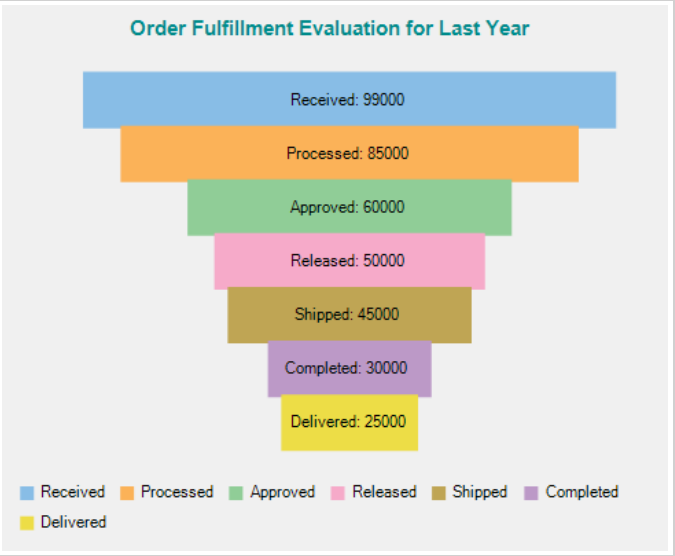
A funnel chart allows you to represent sequential stages in a linear process. For instance, a sales process that tracks prospects across the stages, such as Sales Prospects, Qualified Prospects, Price Quotes, Negotiations, and Closed Sales. In the process, each stage represents a proportion (percentage) of the total. Therefore, the chart takes the funnel shape with the first stage being the largest and each following stage smaller than the predecessor. Funnel charts are useful in identifying potential problem areas in processes where it is noticeable at what stages and rate the values decrease.

- **Trapezoid chart:** Contains a pair of parallel sides.
- **Stacked Bar chart**: Places related values on top of one another in the form of horizontal bars.

The following images show both Trapezoid and Stacked Bar charts displaying the number of orders across seven stages of an order fulfillment evaluation process.



Trapezoid Chart

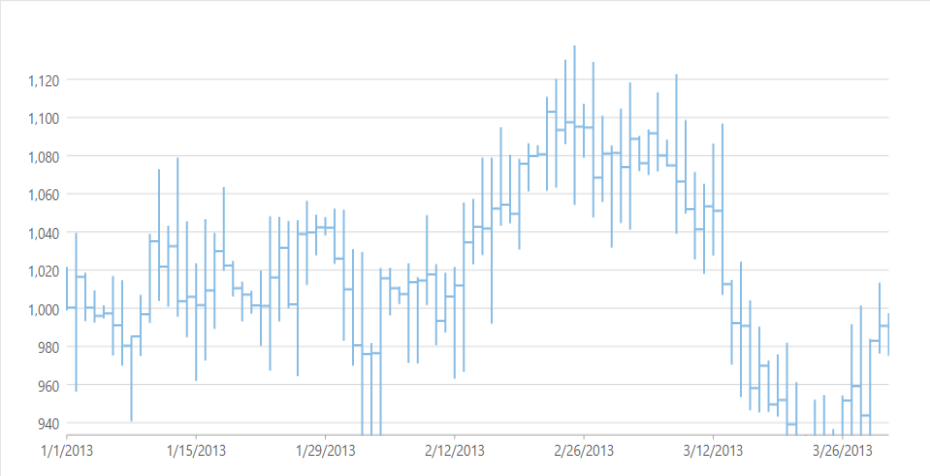


Stacked Bar Chart

Back to Top

HighLowOpenClose chart

HiLoOpenClose are financial charts that combine four independent values to supply high, low, open, and close data for a point in a series. In addition to showing the high and low value of a stock, the Y2 and Y3 array elements represent the stock's opening and closing price, respectively.

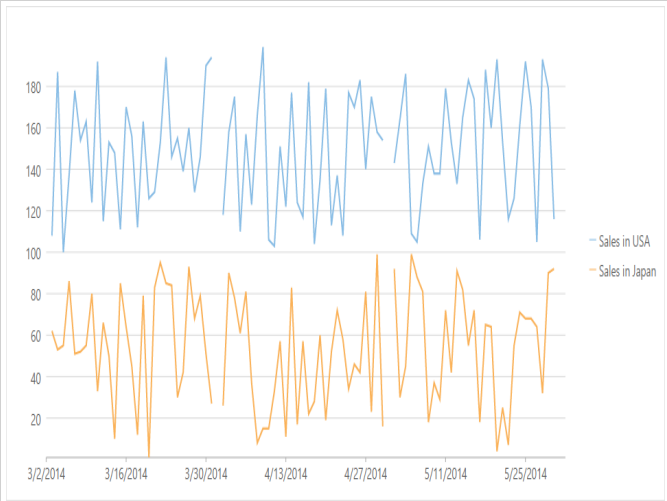


Back to Top

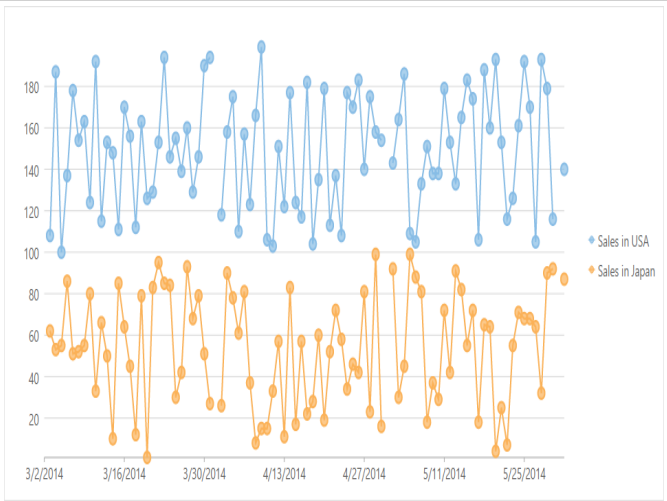
Line and LineSymbols chart

A Line chart draws each series as connected points of data, similar to area chart except that the area below the connected points is not filled. The series can be drawn independently or stacked. LineSymbol chart is similar to line chart except that it represents data points using symbols. It is the most effective way of denoting changes in value between different groups of data.

These charts are commonly used to show trends and performance over time.



Line Chart



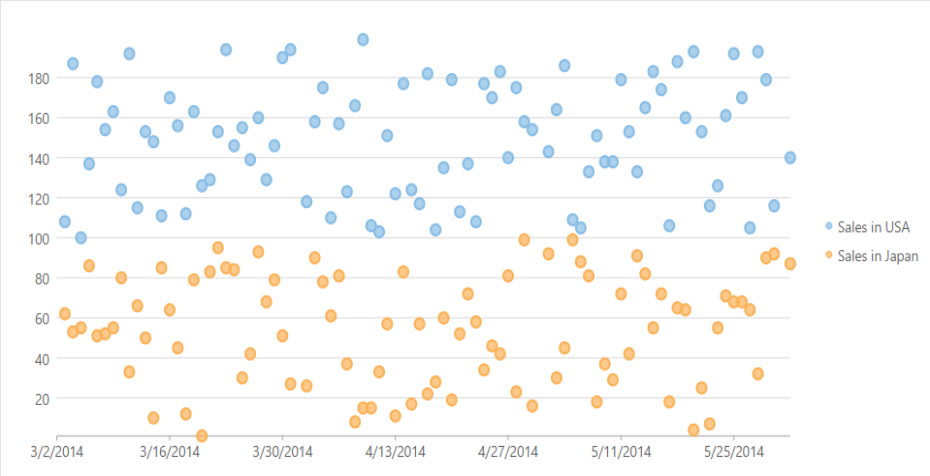
LineSymbols Chart

Back to Top

Scatter chart

A Scatter chart represents a series in the form of points plotted using their X and Y axis coordinates. The X and Y axis coordinates are combined into single data points and displayed in uneven intervals or clusters.

These charts are commonly used to determine the variation in data point density with varying x and y coordinates.

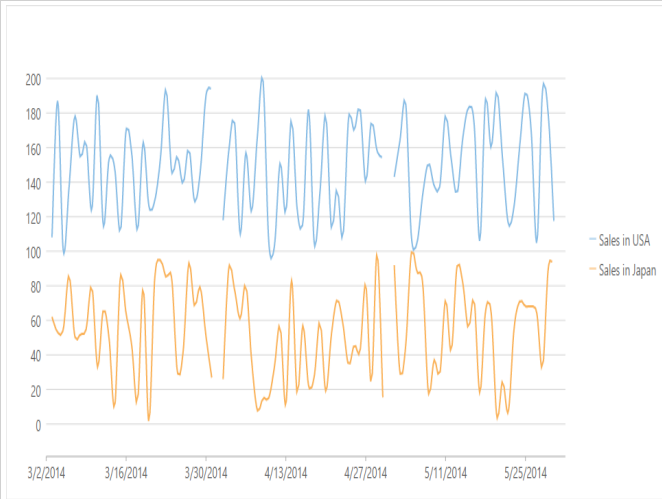


Back to Top

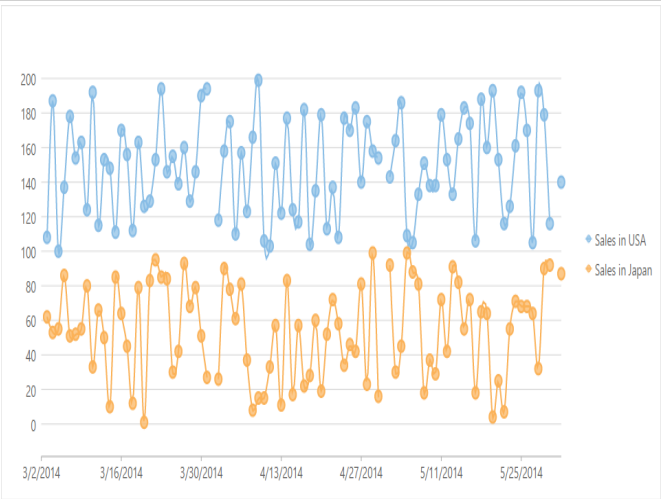
Spline and SplineSymbols chart

A Spline chart draws each series as connected points of data, similar to line chart except that the points are connected by a smooth curved line. There are no sharp corners or abrupt changes in the tightness of the curve. SplineSymbol chart is similar to spline chart except that it represents data points using symbols. It is the most effective way of denoting changes in value between different groups of data.

These charts are commonly used to plot data such as product life cycle, that requires the use of curve-fittings.



Spline Chart

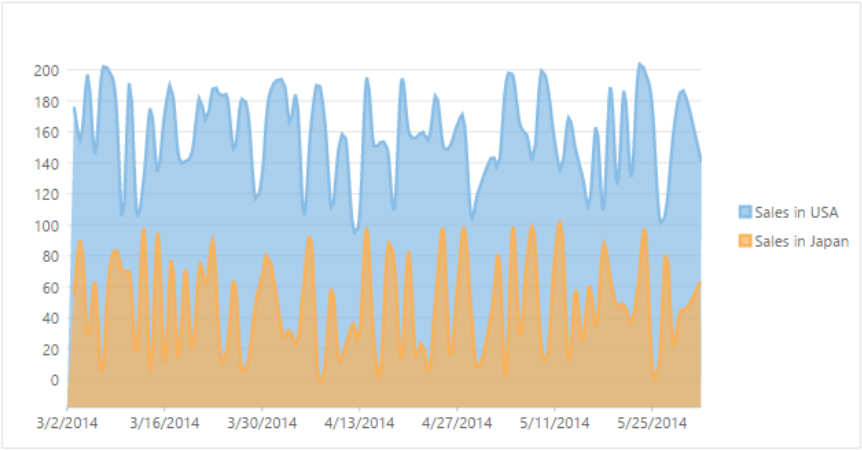


SplineSymbols Chart

Back to Top

SplineArea chart

SplineArea charts are spline charts that display the area below the spline filled with color. SplineArea chart is similar to Area chart as both the charts show area, except that SplineArea chart uses splines and Area chart uses lines to connect data points.



Back to Top

Comparing MVC Charts

Explore all of the features offered by FlexChart in ASP.NET MVC. You can download the comparison matrix in the PDF format.

Chart Types

Chart Types	FlexChart	MS Chart
Area	✓	✓
StackedArea	✓	✓
StackedArea100	✓	✓
SplineArea	✓	✓
StackedSplineArea	✓	
StackedSplineArea100	✓	
Bar	✓	✓
StackedBar	✓	✓
StackedBar100	✓	✓
Bubble	✓	✓
CandleStick	✓	✓
Column	✓	✓
StackedColumn	✓	✓
StackedColumn100	✓	✓
Stock/ HighLowOpenClose	✓	✓
Sunburst	✓	✓
Line	✓	✓
LineStacked	✓	
LineStacked100	✓	
Spline	✓	✓
SplineStacked	✓	
SplineStacked100	✓	
LineSymbols	✓	
StackedLineSymbols	✓	
StackedLineSymbols100	✓	
SplineSymbols	✓	
SplineSymbolsStacked	✓	

StackedSplineSymbols100	✓	
Pie	✓	✓
Doughnut	✓	✓
PieExploded	✓	✓
DoughnutExploded	✓	✓
Point/ Scatter	✓	✓
2D	✓	✓
Heikin-Ashi	*	
LineBreak/ThreeLineBreak	*	✓
Renko	*	✓
Kagi	*	✓
ColumnVolume	*	
EquiVolume	*	
CandleVolume	*	
ArmsCandleVolume	*	
FastLine		✓
FastPoint		✓
StepLine		✓
RangeBar		✓
RangeColumn		✓
SplineRange		✓
Range		✓
PointAndFigure		✓
Polar	✓	✓
Radar	✓	✓
BoxPlot	✓	✓
ErrorBar	✓	✓
Funnel		✓
Pyramid		✓
3D		✓

* Available in FinancialChart

Data Binding

Data Binding	FlexChart	MS Chart
MS SQL Server	✓	✓

OLEDB	✓	✓
ODBC	✓	✓
Object Data Source	✓	✓
Oracle	✓	✓
SharePoint Objects	✓	✓
Unbound Data		✓

Core Features

Core Features	FlexChart	MS Chart
Handle Empty/ Null Data Points	✓	✓
HitTest	✓	✓
DirectX Rendering	✓	
Trendlines	✓	
Coordinate Conversion Methods	✓	✓
Serialization Support		✓
DataPoint Filtering		✓
DataPoint Sorting		✓
Grouping		✓
DataPoints Searching		✓
Copy/ Merge/ Split Series Data		✓
Export to Dataset		✓
Statistical Analysis using predefined formulas		✓

Look & Feel

Look & Feel	FlexChart	MS Chart
Themes	23	12
Custom Palette	✓	✓
Background Color	✓	✓
Background Image	✓	✓
Background Gradient/ HatchStyles		✓
Border and Border Styles		✓

Chart Area

Chart Area	FlexChart	MS Chart

Header	✓	✓
Footer	✓	
Multiple Headers		✓
Header/ Footer Borders	✓	✓
Header/ Footer Alignment	✓	✓
Rotate ChartArea	✓	
Multiple Chart Areas		✓
Toggle Visibility		✓

Plot Area

Plot Area	FlexChart	MS Chart
Plot position configuration		✓
Plot margins	✓	
Zones	✓	
Conditional Item Formatting	✓	
LineMarkers/CrossHair	✓	
Markers for PlotElements	Supported on FlexChart with LineSymbols/ SplineSymbols and Scatter chart types	✓
Markers Size	✓	✓
Markers as Images		✓
Markers: Border and Border styling		✓
Background Image/ Gradient/ HatchStyle for PlotElements		✓
Border and Border styling for Plot Elements		✓
Empty Points Styling		✓

Data Labels

Data Labels	FlexChart	MS Chart
Offset	✓	
ConnectingLines	✓	✓
Borders and Border styling	✓	✓
Styling	✓	✓

Format String	✓	✓
Custom Content	✓	
Positions for Cartesian charts	Bottom/ Center/ Left/ None/ Right/ Top	Only applies to bar charts. Outside/ Left/ Right/ Center
Positions for Pie charts	Center/ Inside/ Outside/ None	Inside/ Outside/ None

Axes

Axes	FlexChart	MS Chart
Axes: Primary X/Y	✓	✓
Axes: Secondary X/Y	✓	✓
Axes: Multiple Secondary X/Y	✓	
Axis Label: Format strings	✓	
Axis Label: Hide	✓	✓
Axis Label: styling	✓	✓
Axis Range (Min/Max) values	✓	
Axis: Hide	✓	✓
Axis: Logarithmic	✓	✓
Axis: Reverse	✓	✓
AxisLine Styling	✓	✓
Labels: Alignment	✓	
Labels: Angle	✓	✓
Labels: Hide overlapping	✓	
Major/ Minor GridLines	✓	✓
Major/ Minor TickMarks	✓	✓
Major/ Minor Units	✓	✓
TickMarks Length	✓	
TickMarks Styling	✓	✓
Title and Title Styling	✓	✓
Configure Origin	Any value	Auto/ Min/ Max
TickMarks Position	Cross/ Inside/ Outside/ None	Cross/ Inside/ Outside/ None
Position	Top/ Bottom/ Left/ Right/ Auto/ None	
ArrowHead styling		✓

Axis Label: Custom		✓
Axis Label: Staggered		✓
ScaleBreaks		✓

Series

Series	FlexChart	MS Chart
Multiple Series	✓	✓
Data binding	✓	✓
Chart types at series level	✓	✓
Styling	✓	✓
X and Y value Functions	✓	
Visibility	X and Y value Functions	Plot/ Plot and Legend/ Hidden

Legends

Legends	FlexChart	MS Chart
Title		✓
Title Style		✓
Toggle Series Visibility from legend	✓	
Orientation	Auto/ Vertical/ Horizontal	Vertical/ Horizontal/ Table
Position	Left/ Top/ Right/ Bottom/Auto/None	Left/ Top/ Right/ Bottom
Custom Legend Items		✓
Items Ordering		✓
Multiple Legends		✓

Marker Symbols

Marker Symbols	FlexChart	MS Chart
Box	✓	✓
Dot	✓	✓
Diamond		✓
Triangle		✓
Cross		✓
Star4		✓
Star5		✓
Star6		✓

Star10		✓

User Interactions

User Interactions	FlexChart	MS Chart
Tooltips	✓	
Series selection	✓	
Point selection	✓	
Zooming	✓	
Scrolling	✓	
Draggable Annotations		

Tooltips

Tooltips	FlexChart	MS Chart
Auto tooltips	✓	
Custom content	✓	String type only
Show Delay	✓	
Styling	✓	
Tooltips for different chart elements		✓

Pie Charts

Pie Charts	FlexChart	MS Chart
Exploded slices	✓	✓
Inner Radius	✓	✓
Starting Angle of first slice	✓	✓
Selected Item Position	✓	
Selected Item Offset	✓	

Exporting & Importing

Exporting & Importing	FlexChart	MS Chart
Export to JPEG/ JPG	✓	✓
Export to PNG	✓	✓
Export to SVG		
Export to BMP		✓
Export to EMF		✓
Export to EMfDual		✓
Export to EMfPlus		✓

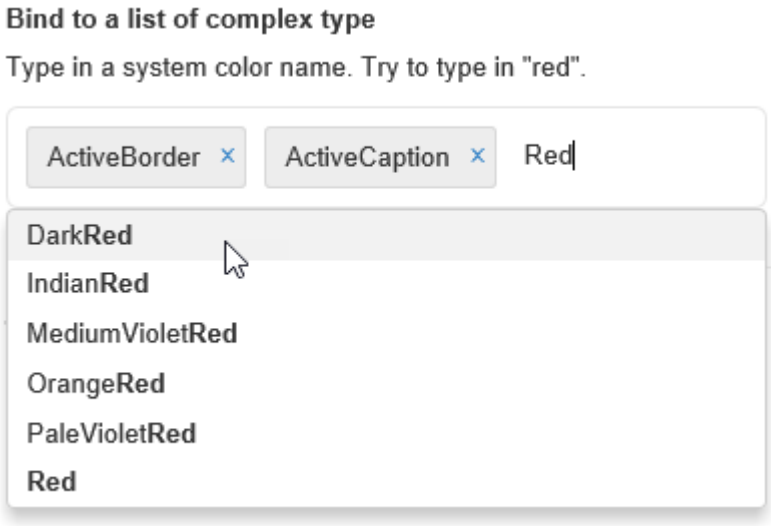
Export to GIF		✓
Export to TIFF		✓
Save/ Load to Binary files		✓
Save/ Load to memory streams		✓
Save/ Load to XML		✓
Printing	✓	✓

Complex Type

A complex data type or complex type is a composite of other existing data types. The main advantage that complex data types have over user-defined types is that users can access and manipulate the individual components of a complex data type. MultiAutoComplete can bind a list of ComplexType data. This can be achieved with the help of DisplayMemberPath and SelectedValuePath properties.

In the example below, MultiAutoComplete uses the **KnownColor Enumeration**, which is a part of **System.Drawing** namespace. KnownColor Enum includes a list of all the system colors that are recognized by the system. To access the Enum values, we have created a **NamedColor** model and defined two properties **Name** and **Value**. These properties are then used in the **View** code to access the Enum values.

The following image shows MultiAutoComplete control after setting the **SelectedValuePath** and **DisplayMemberPath** properties.



The following code example demonstrates how to enable complex type data binding in AutoComplete:

Create a Custom Datasource for MultiAutoComplete

1. Add a new class to the folder **Models** (for example: `NamedColor.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `NamedColor.cs` model. We are defining two string variables **Name** and **Value** to access the KnownColor Enum values.

C#

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

```
using System.Web;

namespace MultiAutoComplete.Models
{
    public class NamedColor
    {
        public string Name { get; set; }
        public string Value { get; set; }
    }
}
```

Back to Top

Create a new Controller

AutoCompleteController.cs

Razor

```
using System;
using System.Drawing;
using System.Linq;
using System.Web.Mvc;
using <ApplicationName>.Models;

namespace MultiAutoComplete.Controllers
{
    partial class MultiAutoCompleteController
    {
        public ActionResult ComplexType()
        {
            var list = GetSystemColors();
            return View(list);
        }

        private static NamedColor[] GetSystemColors()
        {
            return Enum.GetValues(typeof(KnownColor))
                .Cast<KnownColor>()
                .Select(c => new NamedColor
                {
                    Name = c.ToString(),
                    Value = "#" +
Color.FromKnownColor(c).ToArgb().ToString("X8").Substring(2)
                })
                .ToArray();
        }
    }
}
```

View- AutoComplete.cshtml

HTML Helpers

Razor

```
@model IEnumerable<MvcExplorer.Models.NamedColor>

<div>
    <label>Bind to a list of complex type</label>
    <p>Type in a system color name. Try to type in "red".</p>
    @(Html.C1().MultiAutoComplete()
        .Bind(Model)
        .DisplayMemberPath("Name")
        .SelectedIndexes(0,1)
    )
</div>
```

Tag Helpers

Razor

```
@model IEnumerable<MvcExplorer.Models.NamedColor>

<div>
    <label>Bind to a list of complex type</label>
    <p>Type in a system color name. Try to type in "red".</p>
    <c1-multi-auto-complete display-member-path="Name" selected-indexes="new
List<int> { 0, 1 }">
        <c1-items-source source-collection="@Model">
        </c1-items-source>
    </c1-multi-auto-complete>
</div>
```

Back to Top

Features

Series

This section contains information about:

WaterFall Series
: Learn how to add WaterFall Series in FlexChart.

ErrorBar Series
: Learn how to add ErrorBar Series in FlexChart.

BoxWhisker Series
: Learn how to add BoxWhisker Series in FlexChart.

Waterfall

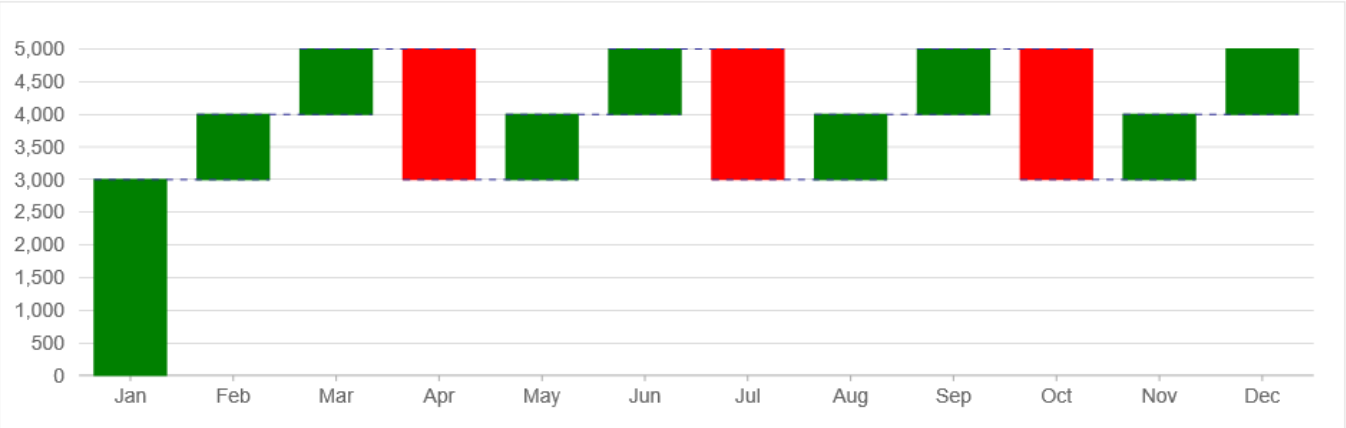
Waterfall series is a form of data visualization that helps in understanding the progressive effect of positive and

negative values in FlexChart. Waterfall series can be used for analytical purposes, mainly for understanding or explaining the continuous transition in the value, which is subjected to increase or decrease.

This topic describes how to use Waterfall series in your FlexChart to represent the positives and negative values of an entity.

- **Step 1: Create a Datasource for FlexChart**
- **Step 2: Add a FlexChart**
- **Step 3: Build and Run the Project**

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



Step 1: Create a Datasource for FlexChart

1. Add a new class to the folder **Models** (for example: `Waterfall.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define a class which will serve as a datasource for the FlexChart.

C#

```
public class Waterfall
    {
        public int WaterfallId { get; set; }
        public string Name { get; set; }
        public int Value { get; set; }
        public Waterfall()
        {
        }
        public Waterfall(string name, int value)
        {
            Name = name;
            Value = value;
        }
        public static List<Waterfall> GetData()
        {
            var names = new string[] { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
            var data = new List<Waterfall>();
            for (int i = 0, len = names.Length; i < len; i++)
            {
                data.Add(new Waterfall(names[i], (((i % 3) + 3) * 1000)));
            };
            return data;
```

```
        }
}
```

Back to Top

Step 2: Add a FlexChart

Complete the following steps to initialize a FlexChart.

Add a new Controller

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: `FlexChartController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

```
C#
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method Index() with the following method.

```
C#
public ActionResult Index()
        {
            return View(Waterfall.GetData());
        }
```

Back to Top

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `FlexChartController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

HTML Helpers

```
Razor
@(Html.C1().FlexChart()
.Bind("Name", "Value", Model)
.Series(ser =>
{
    ser.AddWaterfall()
    .RelativeData(false)
    .ConnectorLines(true)
    .Styles(s =>
    s.ConnectorLines(cnt => cnt.StrokeDasharray("5 5").Stroke("#339")));
```

```
})
.Height("300px")
)
```

Tag Helpers

Razor

```
@{
    var waterfallStyle = new WaterfallStyles();
    waterfallStyle.ConnectorLines = new SVGStyle { Stroke = "#339",
StrokeDasharray = "5 5" };
}
<c1-flex-chart id="flexchart" binding="Value" binding-x="Name" height="300px">
    <c1-items-source source-collection="Model" />
    <c1-flex-chart-waterfall styles="waterfallStyle" relative-data="false"
connector-lines="true" />
</c1-flex-chart>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**FlexChart/Index**) in the address bar of the browser to see the view.

Back to Top

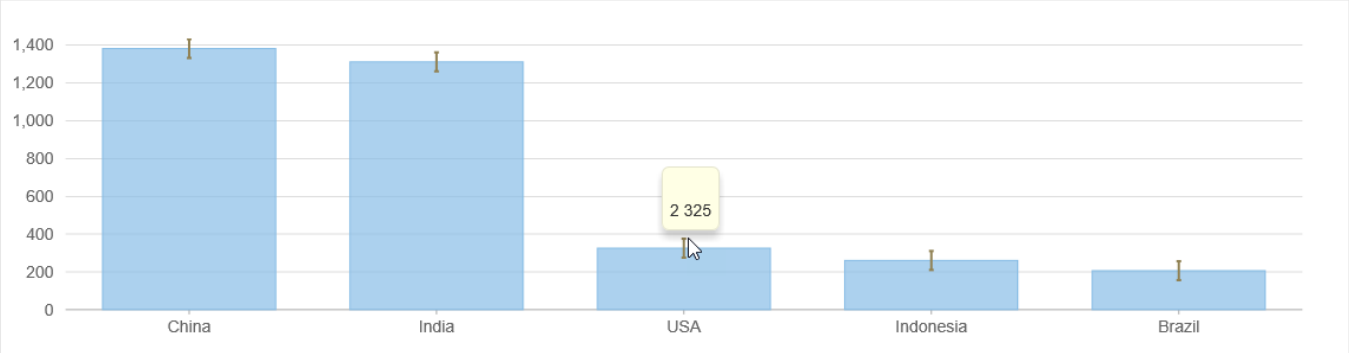
ErrorBar

ErrorBar chart enables user to spot margins of error and standard deviations at a glance. It can be shown as a standard error amount, a percentage, or a standard deviation. User can also set its own custom values to display the exact error amounts in the Chart. Generally, results of scientific studies or experimental sciences use error bars in charts to depict variations in data from original values.

This topic describes how to use ErrorBar series in your FlexChart to represent error and standard deviations. To work with ErrorBar Series, you need to create an instance of **FlexChart** class, which is a part of **C1.Web.Mvc** namespace.

- **Step 1: Create a Datasource for FlexChart**
- **Step 2: Add ErrorBar Series to FlexChart**
- **Step 3: Build and Run the Project**

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



Step 1: Create a Datasource for FlexChart

1. Add a new class to the **Models** folder (for example: `PopulationByCountry.cs`).
2. Add the following code to the new model to define a class which will serve as a datasource for the FlexChart.

PopulationByCountry.cs copyCode

```
using System.Collections.Generic;
namespace ErrorBarSeries.Models
{
    public class PopulationByCountry
    {
        public string Country { get; set; }
        public int Population { get; set; }
        public static List<PopulationByCountry> GetData()
        {
            string[] countries = "China,India,USA,Indonesia,Brazil".Split(new
char[] { ',' });
            int[] population = new int[] { 1380, 1310, 325, 260, 206 };
            var data = new List<PopulationByCountry>();

            for (var i = 0; i < countries.Length; i++)
            {
                data.Add(new PopulationByCountry()
                {
                    Country = countries[i],
                    Population = population[i]
                });
            }
            return data;
        }
    }
}
```

Back to Top

Step 2: Add ErrorBar Series to FlexChart

To add a FlexChart to the application, follow these steps:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.

3. In the **Add Scaffold** dialog, follow these steps:
    1. Select **Empty MVC Controller** template.
    2. Set name of the controller (for example: `ErrorBarController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

```
C#
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method Index() with the following method.

```
C#
public ActionResult Index()
        {
            return View(PopulationByCountry.GetData());
        }
```

Back to Top

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `ErrorBarController`.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

HTML Helpers

```
Index.cshtml copyCode
@model IEnumerable<PopulationByCountry>
@using ErrorBarSeries.Models;
<br />
<br />
@(Html.C1().FlexChart()
.Bind("Country", "Population", Model)
.Series(ser =>
{
    ser.AddErrorBar().Value(50)
    .ErrorBarStyle(errorBarStyle =>
errorBarStyle.Fill("#e6e6e6").Stroke("#918254").StrokeWidth(2));
})
.Height("300px")
)
```

Tag Helpers

```
Index.cshtml copyCode
```

```
@model IEnumerable<PopulationByCountry>
@using ErrorBarSeries.Models;

@{
    var errorBarStyle = new SVGStyle { Fill = "#e6e6e6", Stroke = "#918254",
StrokeWidth = 2 };
}

<c1-flex-chart binding="Population" binding-x="Country" height="300px">
    <c1-items-source source-collection="Model" />
    <c1-flex-chart-error-bar name="Population" value="50" error-bar-
style="@errorBarStyle" />
</c1-flex-chart>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**ErrorBar/Index**) in the address bar of the browser to see the view.

Back to Top

Box-and-Whisker

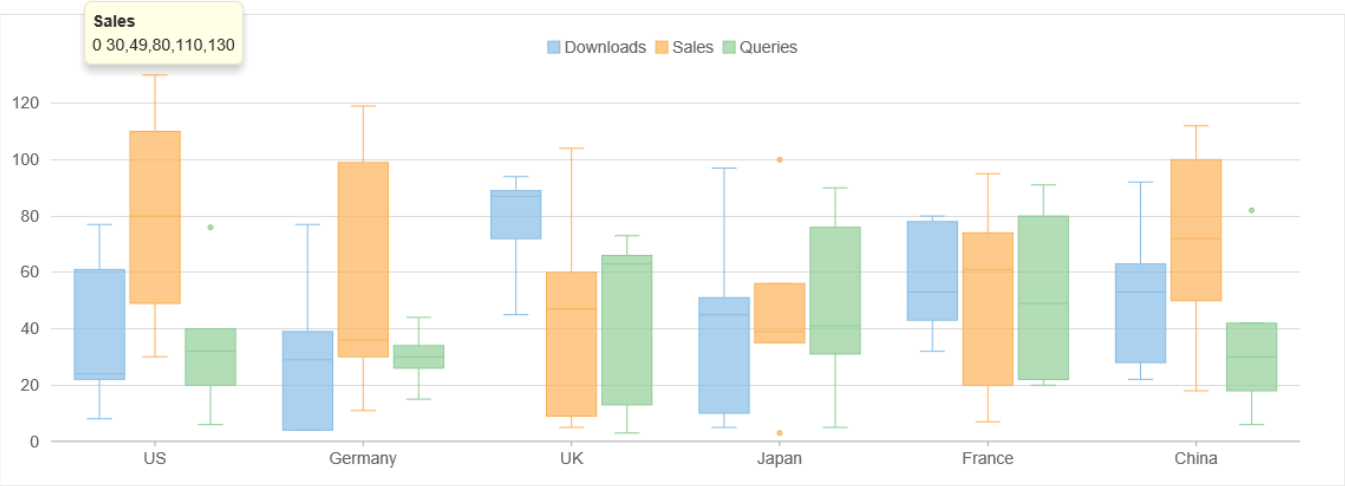
BoxWhisker chart series shows distribution of data into quartiles, highlighting the mean and the outliers. The boxes may have lines extending vertically called "whiskers". These lines indicate variability outside the upper and lower quartiles, and any point outside those lines or whiskers is considered an outlier.

They are most commonly used in statistical analysis. For example, you could use a box and whisker chart to compare medical trial results or teachers' test scores.

This topic describes how to use BoxWhisker series in your FlexChart to show distribution of data into quartiles, highlighting the mean and the outliers. To work with BoxWhisker Series, you need to create an instance of **FlexChart** class, which is a part of **C1.Web.Mvc** namespace.

- **Step 1: Create a Datasource for FlexChart**
- **Step 2: Add BoxWhisker Series to FlexChart**
- **Step 3: Build and Run the Project**

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



Step 1: Create a Datasource for FlexChart

1. Add a new class to the **Models** folder (for example: `ProductSales.cs`).
2. Add the following code to the new model to define a class which will serve as a datasource for the FlexChart.

ProductSale.cs copyCode

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

namespace BoxWhiskerSeries.Models
{
    public class ProductSales
    {
        public int Id { get; set; }
        public string Country { get; set; }
        public List<int> Sales { get; set; }
        public List<int> Downloads { get; set; }
        public List<int> Queries { get; set; }
        public ProductSales(string country, int[] downloads, int[] sales, int[]
queries)
        {
            Country = country;
            Sales = sales.ToList();
            Downloads = downloads.ToList();
            Queries = queries.ToList();
        }
        public static List<ProductSales> GetData()
        {
            int[][][] stats = {

new int[][]{ new int[] { 8, 22, 24, 61, 77 }, new int[] { 30, 49, 80, 110, 130
}, new int[] { 6, 20, 32, 40, 76 } },

new int[][]{ new int[] {4, 4, 29, 39, 77}, new int[] {11, 30, 36, 99, 119}, new
int[] {15, 26, 30, 34, 44} },

new int[][]{ new int[] {45, 72, 87, 89, 94}, new int[] {5, 9, 47, 60, 104}, new
```

```
int[] {3, 13, 63, 66, 73} },

new int[][]{ new int[] {5, 10, 45, 51, 97}, new int[] {3, 35, 39, 56, 100}, new
int[] {5, 31, 41, 76, 90} },

new int[][]{ new int[] {32, 43, 53, 78, 80}, new int[] {7, 20, 61, 74, 95}, new
int[] {20, 22, 49, 80, 91} },

new int[][]{ new int[] {22, 28, 53, 63, 92}, new int[] {18, 50, 72, 100, 112},
new int[] {6, 18, 30, 42, 82} }

};
            var countries = new string[] { "US", "Germany", "UK", "Japan",
"France", "China" };
            var data = new List<ProductSales>();
            for (var i = 0; i < countries.Length; i++)
            {
                data.Add(new ProductSales(countries[i], stats[i][0], stats[i]
[1], stats[i][2]));
            }
            return data;
        }
    }
}
```

Back to Top

Step 2: Add BoxWhisker Series to FlexChart

To add a FlexChart to the application, follow these steps:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select **Empty MVC Controller** template.
    2. Set name of the controller (for example: `BoxWhiskerController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
public ActionResult Index()
        {
            return View(ProductSales.GetData());

        }
```

Back to Top

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `BoxWhiskerController.`

2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

HTML Helpers

Index.cshtml copyCode

```
@using C1.Web.Mvc.Chart;
@using BoxWhiskerSeries.Models;
@model IEnumerable<ProductSales>
<br />
<br />
@(Html.C1().FlexChart()
.Bind("Country", "Downloads", Model)
.Series(ser =>
{
    ser.AddBoxWhisker().Name("Downloads");
    ser.AddBoxWhisker().Binding("Sales").Name("Sales").ShowOutliers(true);
    ser.AddBoxWhisker().Binding("Queries").Name("Queries").ShowOutliers(true);
})
.Legend(Position.Top))
```

Tag Helpers

Index.cshtml copyCode

```
@using C1.Web.Mvc.Chart;
@using BoxWhiskerSeries.Models;
@model IEnumerable<ProductSales>

<c1-flex-chart binding="Downloads" binding-x="Country" legend-position="Top">
<c1-items-source source-collection="Model" />
<c1-flex-chart-boxwhisker name="Downloads" />
<c1-flex-chart-boxwhisker name="Sales" binding="Sales" show-outliers="true" />
<c1-flex-chart-boxwhisker name="Queries" binding="Queries" show-outliers="true"
/>
</c1-flex-chart>
```

Step 3: Build and Run the Project

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

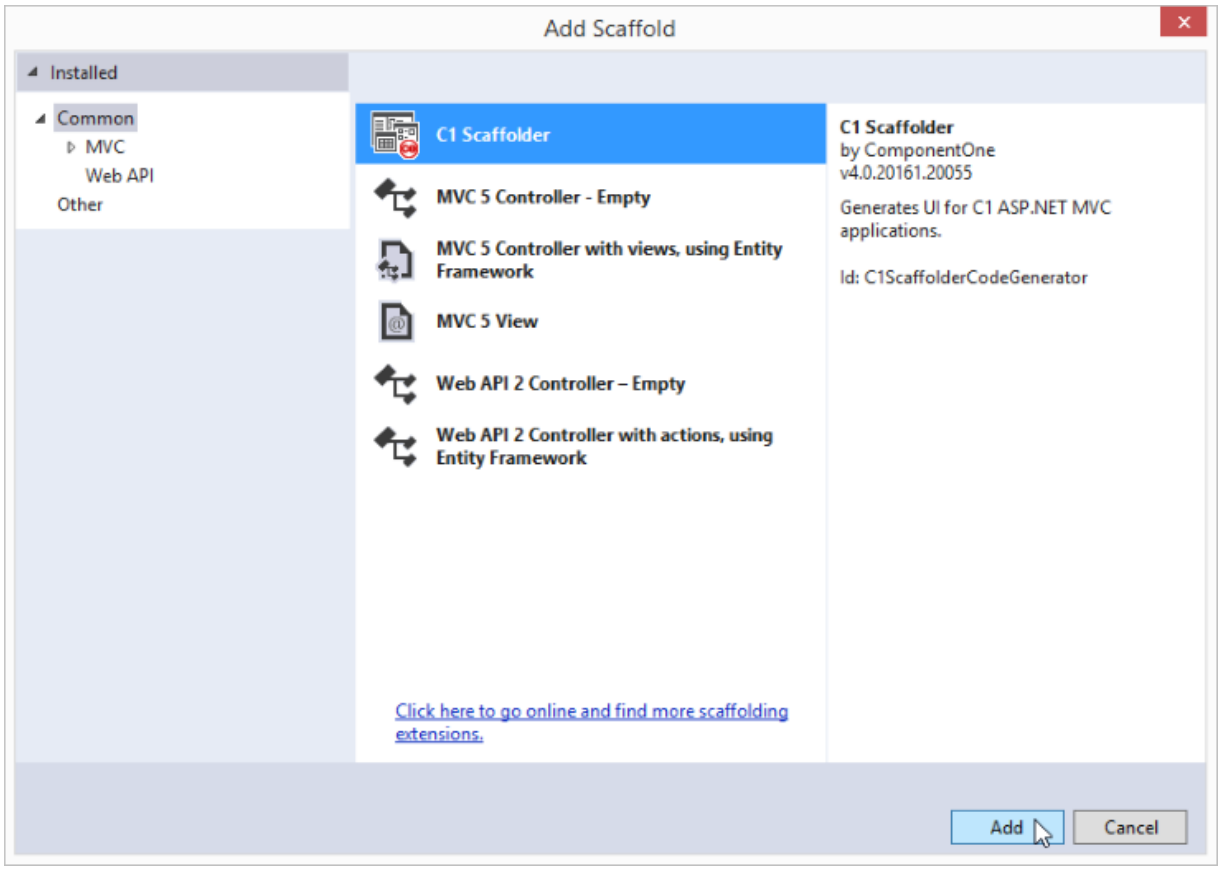
> Append the folder name and view name to the generated URL (for example: http://localhost:1234/**ErrorBar/Index**) in the address bar of the browser to see the view.

Back to Top

Scaffolding

The steps to scaffold **ComponentOne FlexChart** control for ASP.NET MVC are as follows:

1. Configure the datasource. Refer the topic Data Source Configuration to see configuring a datasource in an application.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane.



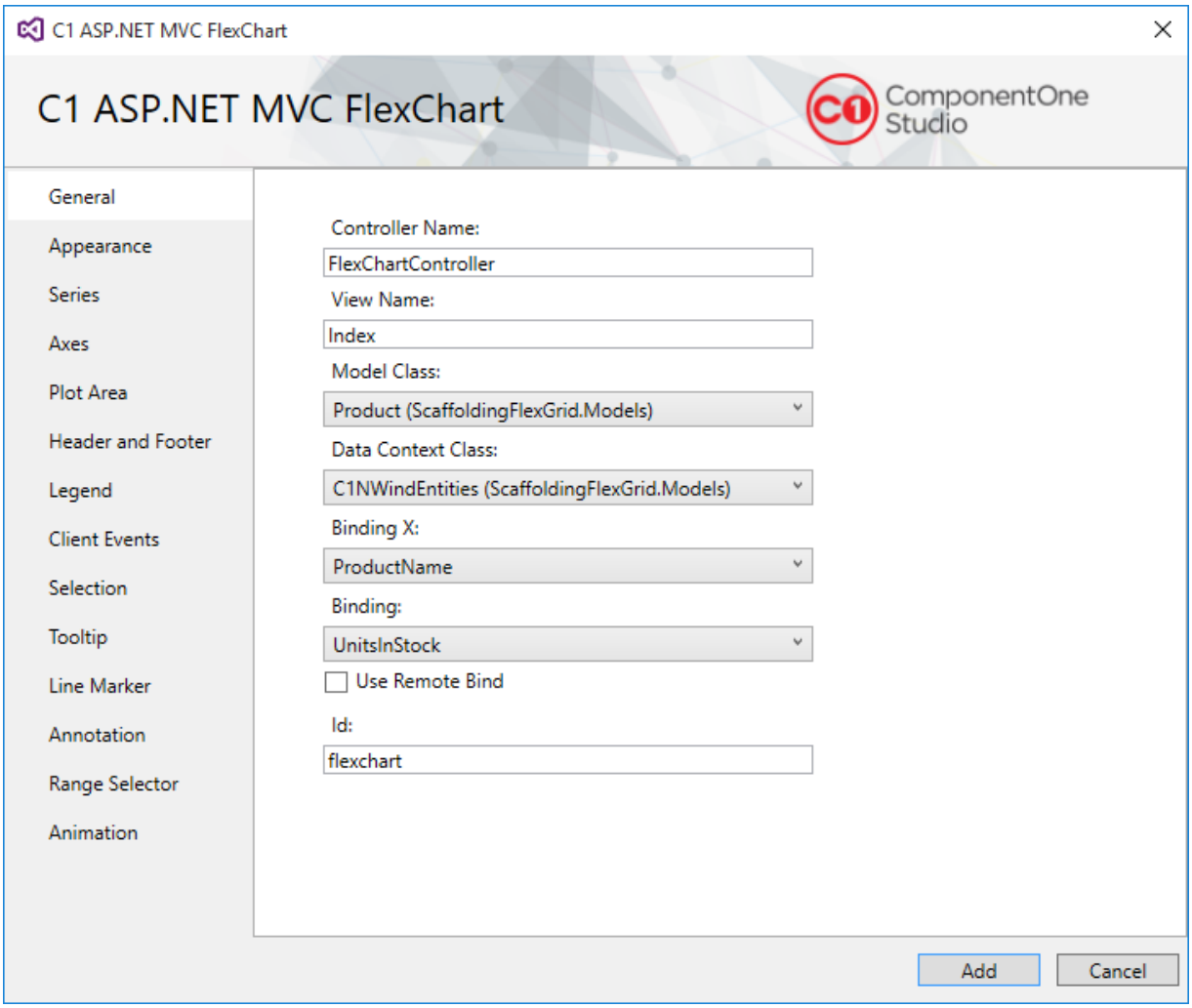
You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

4. Click **Add**.
5. In the **Add Scaffold** dialog, select FlexChart control, and the click **Next**.

The **C1 ASP.NET MVC FlexChart** wizard appears with the General tab selected by default.

6. In the General tab, specify the model details as follows:

    1. Enter the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in **Step 1**. In our case, we select Product to populate data for the FlexChart.
    3. Select **Data Context Class** from the drop-down list. In our case, we select C1NWindEntities.
    4. In the **Binding X** drop-down, select a column to bind it with the x-axis. In our case, we have selected **ProductName**.
    5. In the **Binding Y** drop-down, select a column to bind it with the y-axis. In our case, we have selected **UnitInStock**.



7. Go to **Appearance** tab, and then select **Bar** from the **Chart Type** drop-down option to create a Bar chart.
8. Click **Add**. You will notice that the Controller and View for the selected model are added to your project.
9. Press **F5** to run the project.

Chart Gradient

A gradient is a graphical effect that produces a three dimensional color look by blending one color into another. Multiple colors can be used, where one color gradually fades and changes to the other color. Elements or controls using gradient color look better, because the gradient is generated with the help of browser. FlexChart allows you to apply gradient colors to improve the appearance of your chart, with respect to styling.

The gradient descriptor is an expression formatted as follows:

HTML

```
<type>(<coords>)<colors>[:<offset>[:<opacity>]]
[-<colors>[:<offset>[:<opacity>]]]-<colors>[:<offset>[:<opacity>]]
```

The gradient descriptor includes the following elements.

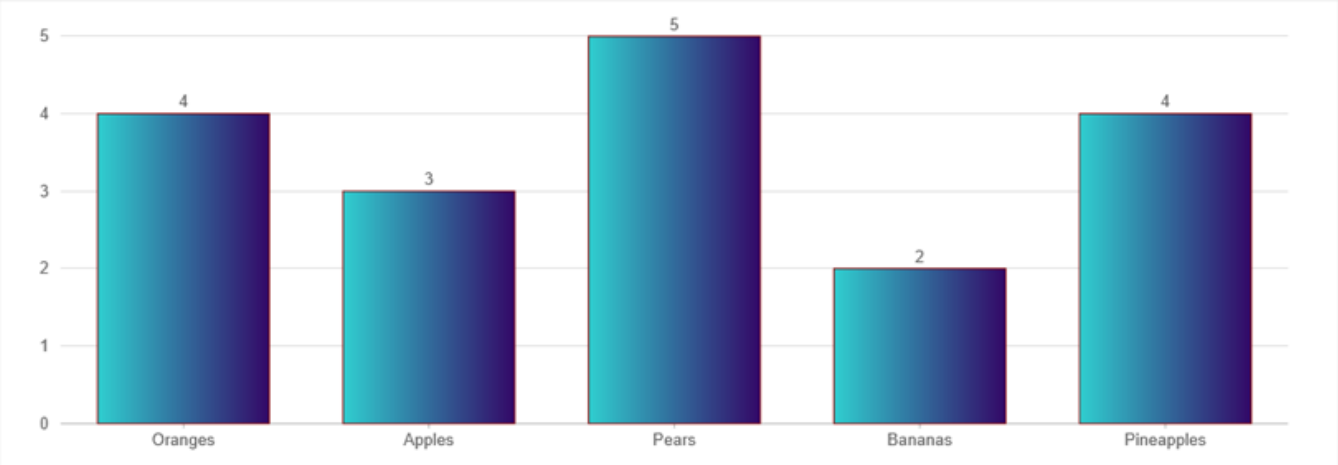
- **Type** - The <type> can be either linear or radial.
  - The uppercase L or R letters indicate absolute coordinates offset from the SVG surface.
  - The lowercase l or r letters indicate coordinates calculated relative to the element to which the gradient is applied.

- **Coordinates**
  - Specify a linear gradient vector as x1, y1, x2, y2.
  - Specify a radial gradient vector as cx, cy, r, and optional fx, fy, fr specifying a focal point away from the center of the circle.

- **Colors -** The <colors> is specified as a list of dash-separated CSS color values. Each color may be followed by a custom offset and opacity value, separated with a colon character.

Linear and radial gradient type examples are as follows.

Linear Gradient Format
`'l(0,0,1,0)#ff0000-#00ff00-#0000ff', 'L(0,0,300,300)#ff0000:0.2:0.2-00ff00:0.8'`
Radial Gradient Format
`'r(0.5,0.5,1)#ff0000-#0000ff', 'R(100,100,100,200,200,200)#ff0000-#0000ff'`



The following code example demonstrates how to add gradient colors in FlexChart. This example uses the **Fruit.cs** model created in the FlexChart: Quick Start section.

ASP.NET

Razor

```
@using ChartGradient.Models
@model IEnumerable<Fruit>
@using C1.Web.Mvc.Chart

@(Html.C1().FlexChart().Id("chartGradients")
    .ChartType(ChartType.Column)
    .Bind(Model)
    .BindingX("Name").Legend(Position.None)
    .DataLabel(label => { label.Content("{y}"); })
    .Series(ser =>
    {
        ser.Add().Name("March").Binding("MarPrice").Style(s => s.Stroke("darkred")
        .StrokeWidth(1).Fill("l(0, 0, 1, 0)#30cfd0-#330867"));
    }))
```

ASP.NET Core

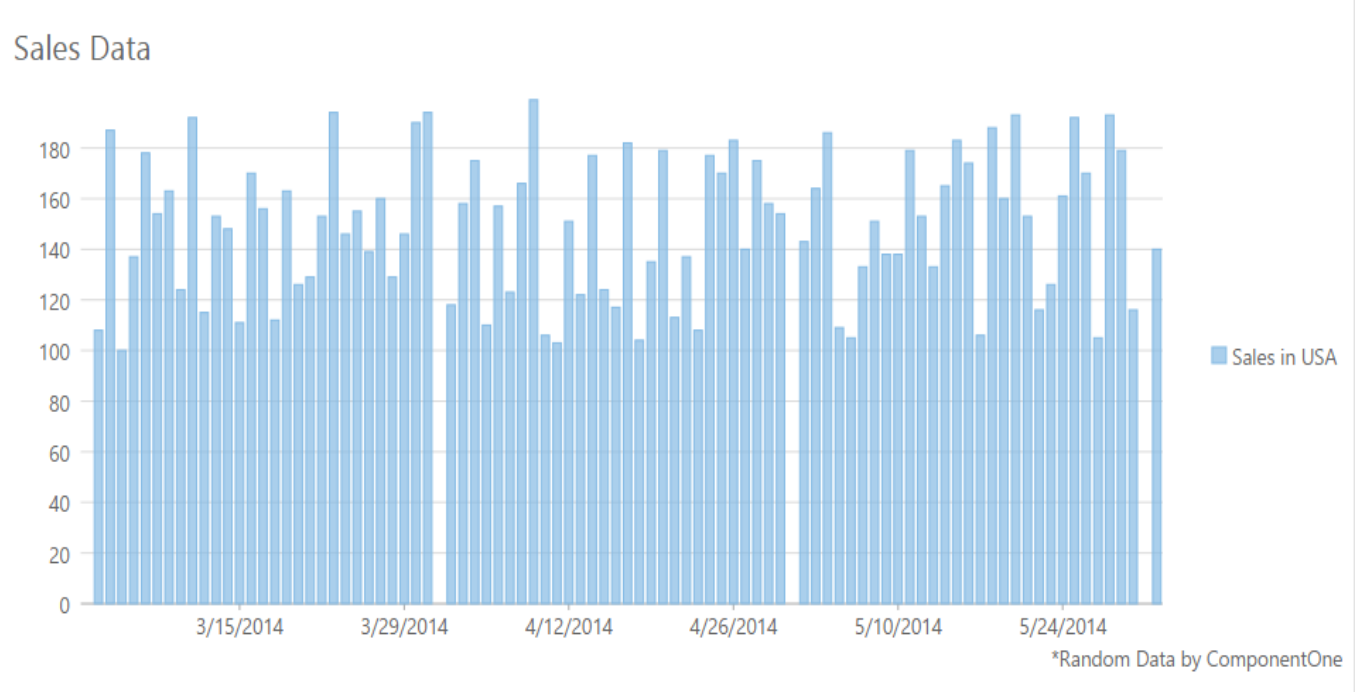
Razor

```
@using ChartGradient.Models
@model IEnumerable<Fruit>
@using C1.Web.Mvc.Chart

@{

var marchStyle = new SVGStyle
    {
        Stroke = "darkred",
        StrokeWidth = 1,
        Fill = "l(0,0,1,0)#30cfd0-#330867"
    };

}
<c1-flex-chart id="chartGradients" binding-x="Name" chart-type="Column" legend-
position="None">
    <c1-flex-chart-datalabel content="{y}" />
    <c1-items-source source-collection="Model" />
    <c1-flex-chart-series name="March" binding="MarPrice" style="@marchStyle"></c1-
flex-chart-series>
</c1-flex-chart>
```

Header and Footer

You can add a title to the FlexChart control by setting its `Header` property. Besides a title, you may also set a footer for the chart by setting the `Footer` property. You can also style the header and footer text with the help of `HeaderStyle` and `FooterStyle` properties.

The image below shows how the FlexChart appears, after these properties have been set.



The following code examples demonstrate how to set these. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.Header("Sales Data")
.HeaderStyle(hs => hs.Halign("left"))
.FooterStyle(fs => fs.Halign("right"))
.Footer("*Random Data by ComponentOne")
```

Tag Helpers

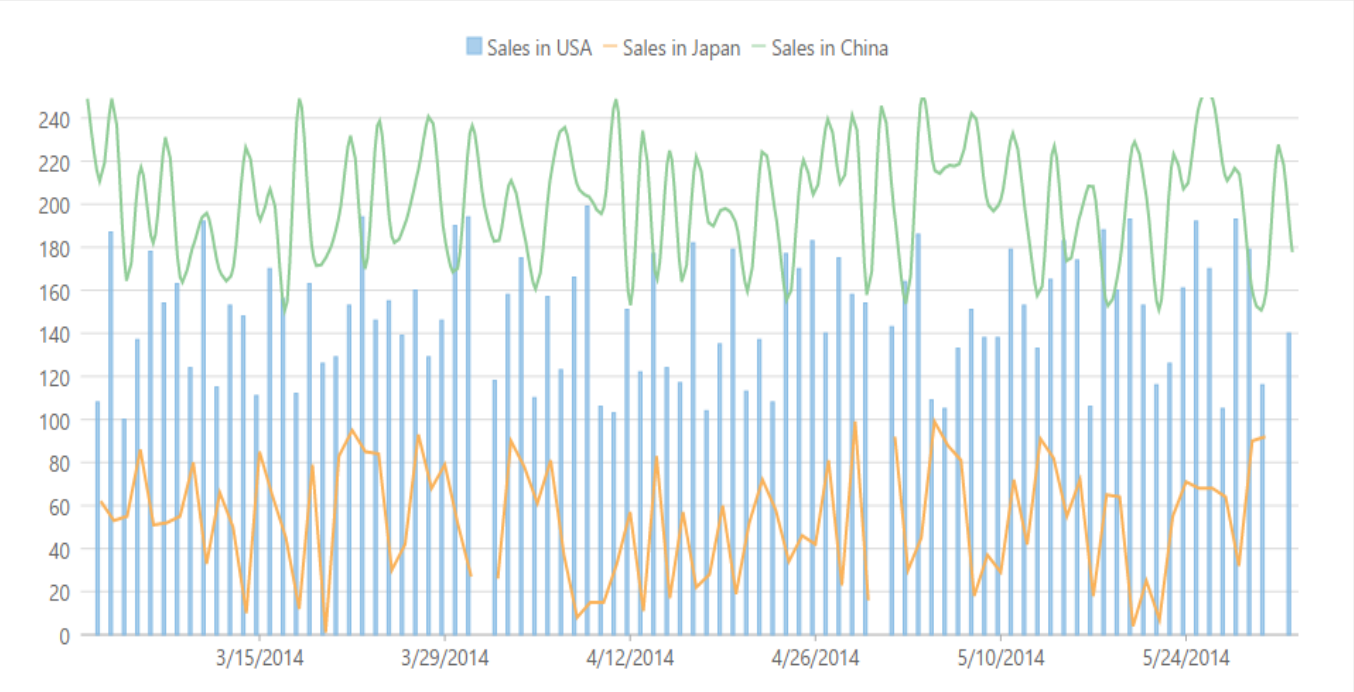
HTML copyCode

```
<c1-flex-chart binding-x="Name" chart-type="ChartType.Column" header="Sales Data"
footer="*Random Data by ComponentOne">
    <c1-flex-chart-title-style c1-property="FooterStyle" halign="right"></c1-flex-
chart-title-style>
    <c1-flex-chart-title-style c1-property="HeaderStyle" halign="left"></c1-flex-
chart-title-style>
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA">
    </c1-flex-chart-series>
</c1-flex-chart>
```

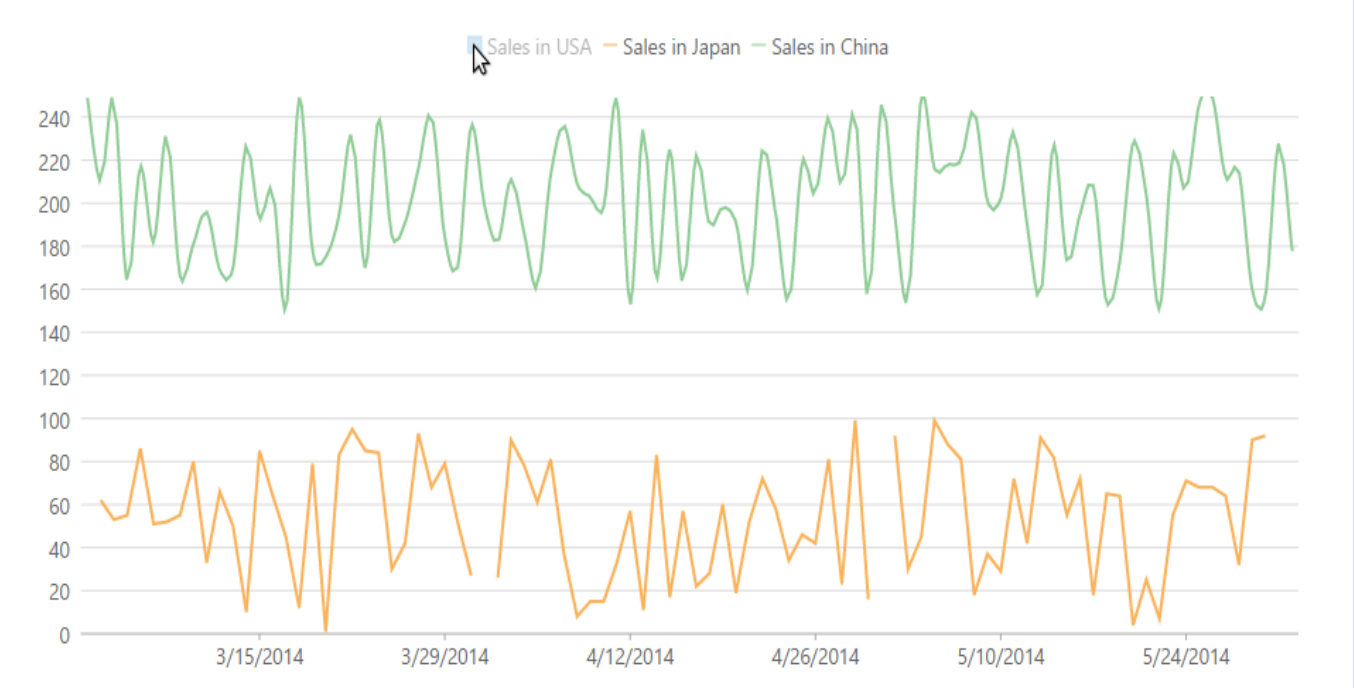
Legend

You can specify the position where you want to display the legend in the `Legend` property of the FlexChart. Setting the `LegendToggle` property to true lets you toggle the visibility of any series by clicking its corresponding legend item.

The image below shows how the FlexChart appears after these properties have been set.



Clicking the legend item makes the corresponding series invisible as shown below.



The following code example demonstrates how to set these properties. This example uses the sample created in the Mixed Charts section.

In Code

HTML Helpers

Razor	copyCode

```
.Legend(C1.Web.Mvc.Chart.Position.Top)
.LegendToggle(true)
```

Tag Helpers

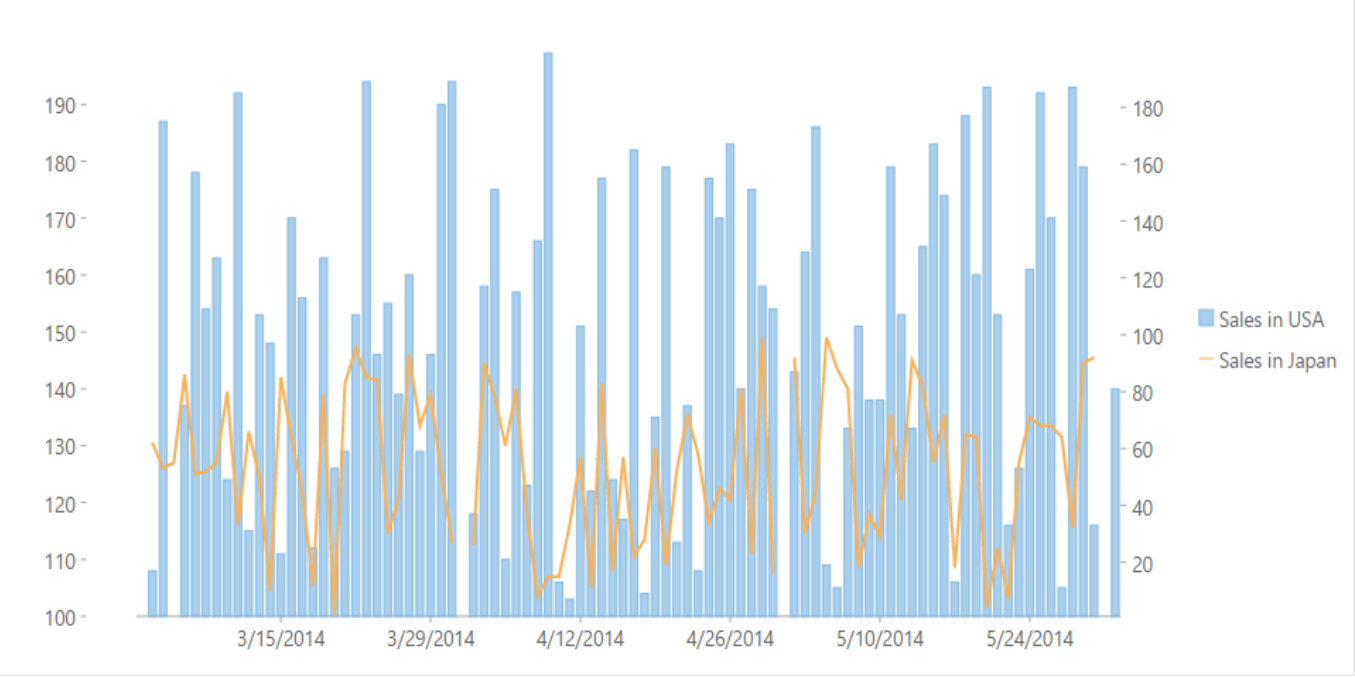
HTML	copyCode

```
<c1-flex-chart legend-position="Position.Top" legend-toggle="true">
```

Multiple Axes

FlexChart allows you to add secondary axes when you add multiple series to the chart. The scale of a secondary axis represents the values of its associated series. You can even change the `ChartType` of the associated series to easily identify the values plotted along the secondary axis.

The image below shows how multiple Y axes appear on the FlexChart.



The following code example demonstrates how to add multiple Y axes to the FlexChart. This example uses the sample created in the Mixed Charts section.

In Code

HTML Helpers

Razor	copyCode

```
// Initialize FlexChart
@(Html.C1().FlexChart()
```

```
.Bind("Date", Model)

//Add Series to the chart
.Series(sers =>
  {
      //Add the first series
      sers.Add()
     .Binding("SalesInUSA")
     .Name("Sales in USA")
     .ChartType(C1.Web.Mvc.Chart.ChartType.Column)

     //Set Y axis position for the first series
     .AxisY(axis => axis.Position(C1.Web.Mvc.Chart.Position.Left));

      //Add the second series
      sers.Add()
     .Binding("SalesInJapan")
     .Name("Sales in Japan")
     .ChartType(C1.Web.Mvc.Chart.ChartType.Line)

     //Set Y axis position for the second series
     .AxisY(axis=>axis.Position(C1.Web.Mvc.Chart.Position.Right));

  })
)
```

Tag Helpers

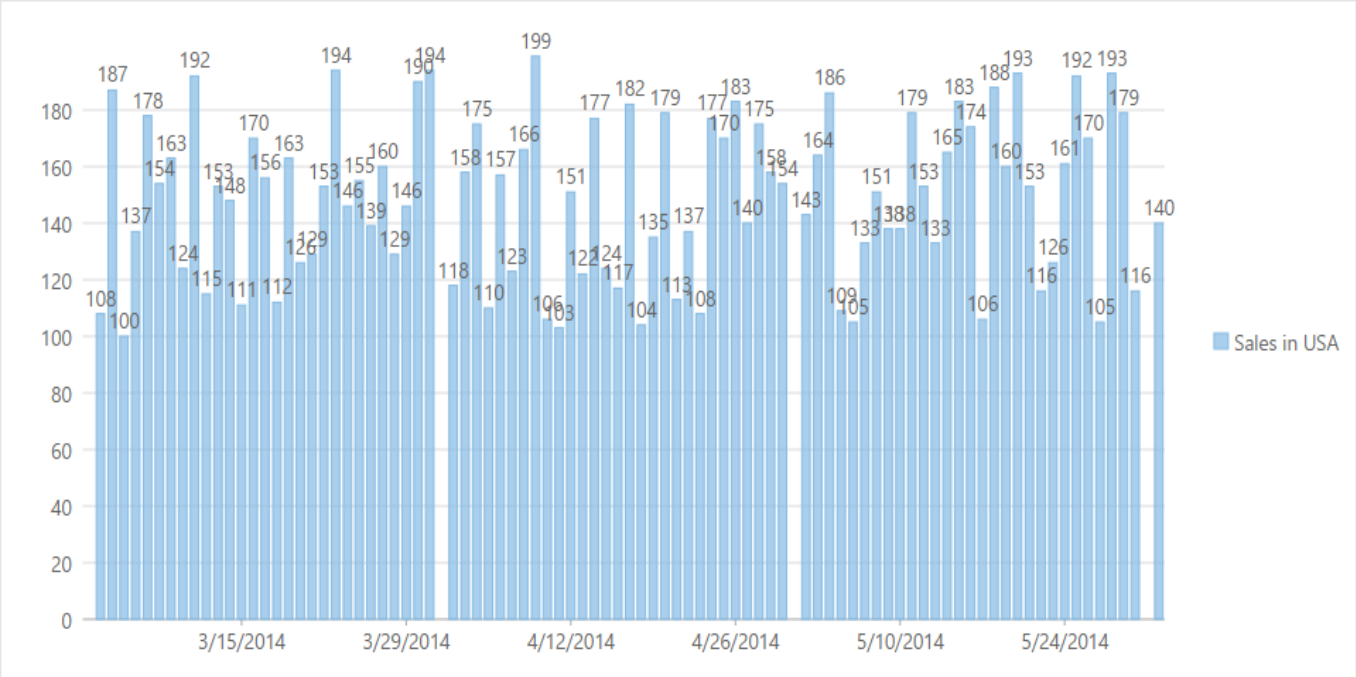
HTML copyCode

```
<c1-flex-chart binding-x="Name" chart-type="ChartType.Column">
    <c1-items-source source-collection="Model"></c1-items-source>

    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA" chart-
type="ChartType.Column"
                          >
        <c1-flex-chart-axis c1-property="AxisX" format="dd-MMM"></c1-flex-chart-axis>
    </c1-flex-chart-series>
    <c1-flex-chart-series binding="SalesInJapan" name="Sales in Japan" chart-
type="ChartType.Line">

    </c1-flex-chart-series>

</c1-flex-chart>
```

Labels

You can add labels to display the value of each data point by using the `DataLabel` property. You can even specify the position where you want to display your labels by setting the `LabelPosition` property.

The image below shows how labels appear on each data point on the FlexChart.



The following code example demonstrates how to add labels to the FlexChart. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.DataLabel(dl => dl.Position(C1.Web.Mvc.Chart.LabelPosition.Top).Content("{y}"))
```

Tag Helpers

HTML copyCode

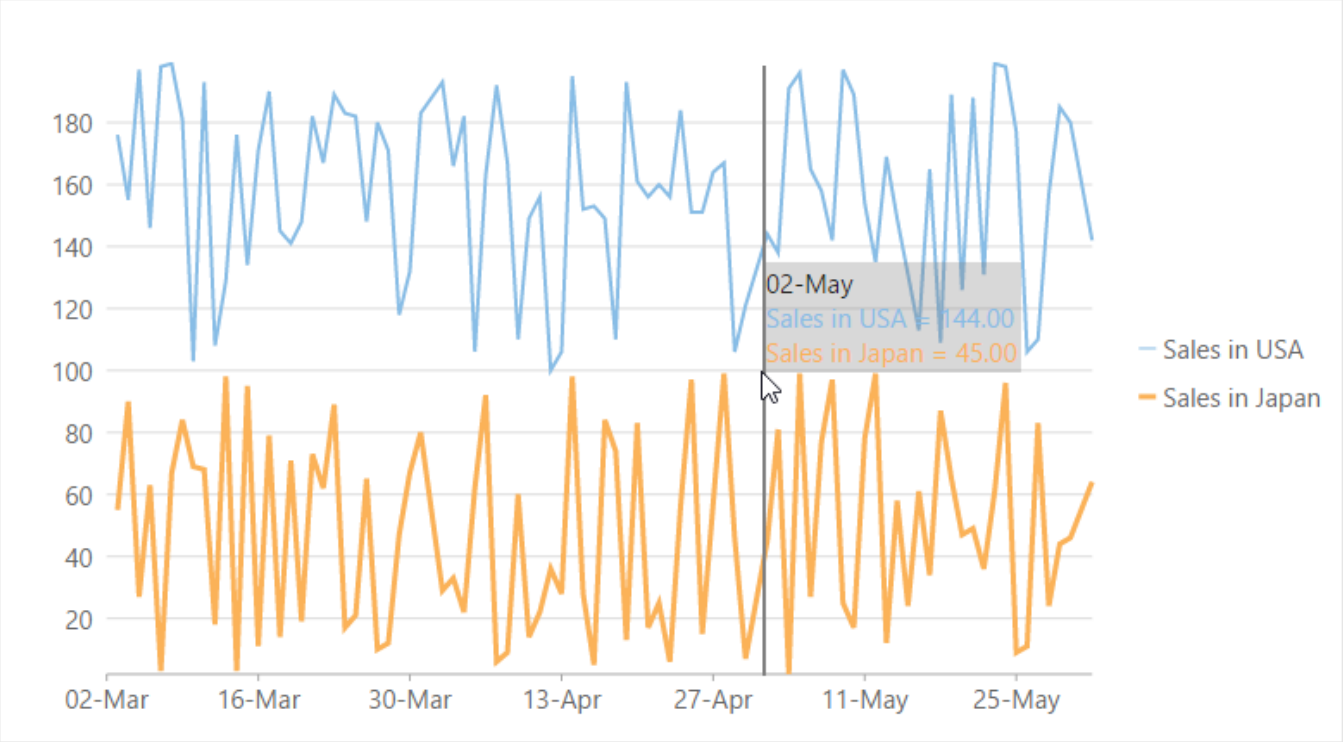
```
<c1-flex-chart-datalabel position="LabelPosition.Top" content="{y}"></c1-flex-chart-datalabel>
```

Markers

Markers are the symbols used to display data points when the mouse is hovered over the data series. In FlexChart, you can add line markers using AddLineMarker method. LineMarkerLines property allows you to set line markers to:

- **Vertical**: Shows vertical line markers.
- **Horizontal**: Shows horizontal line markers.
- **Both**: Sets vertical as well as horizontal line markers.
- **None**: Shows no line.

The image below shows vertical line marker with marker content to display the value of data points on the FlexChart.



The following code example demonstrates how to add vertical line markers and marker content to the FlexChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

```
@using MVCFlexChart.Models;
@using C1.Web.Mvc.Chart;
@model IEnumerable<FruitSale>

@*Add content to display with marker*@
<script type="text/javascript">
    function lineMarkerContent(hitInfo, pt) {
        var html = '', chart = hitInfo.series ? hitInfo.series.chart : undefined;
        if (!chart || !chart.series) {
            return html;
        }
        chart.series.forEach(function (s, i) {
            var ht = s.hitTest(new wijmo.Point(pt.x, NaN)), hostEle = s.hostElement,
polyline;

            polyline = s.hostElement ? s.hostElement.getElementsByTagName("polyline")
[0] : undefined;

            if (polyline && ht.x && ht.y !== null) {
                if (i == 0) {
                    html += wijmo.Globalize.formatDate(ht.x, 'dd-MMM');
                }
                html += '<div style="color:' + polyline.getAttribute('stroke') + '">'
```

```
+ ht.name + ' = ' + ht.y.toFixed(2) + '</div>';
            }
        });
        return html;
    }
</script>

@*Add marker*@
<div style="width: 780px">
    @(Html.C1().FlexChart().Bind("Date", Model).ChartType(ChartType.Line).Series(sers
=>
{
    sers.Add().Binding("SalesInUSA").Name("Sales in USA");
    sers.Add().Binding("SalesInJapan").Name("Sales in Japan");
})
    .AxisX(x => x.Format("dd-MMM")).Tooltip(tp => tp.Content(""))
    .AddLineMarker(lm => lm
    .Alignment(LineMarkerAlignment.Auto)
    .Lines(LineMarkerLines.Vertical)
    .Interaction(LineMarkerInteraction.Move).Content("lineMarkerContent")))
</div>
```

Tag Helpers

HTML copyCode

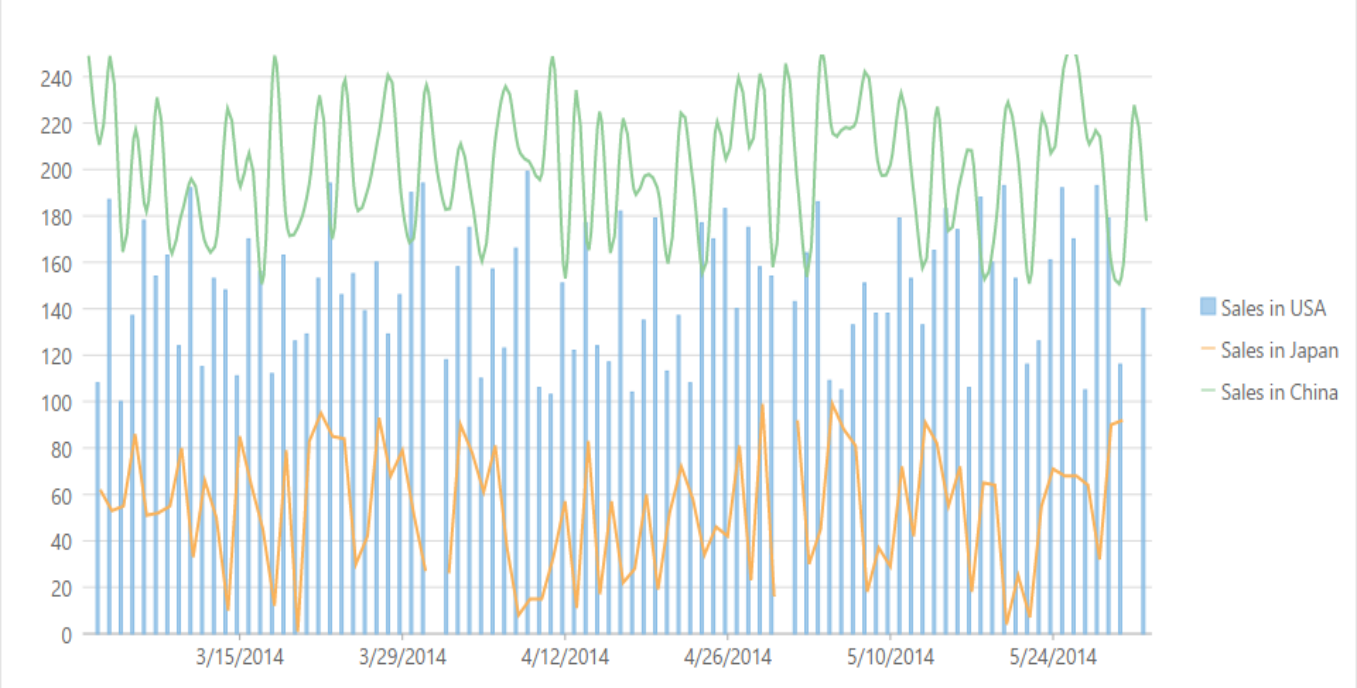
```
@using C1.Web.Mvc.Chart;
@using TagFlexGrid.Models;

<script type="text/javascript">
    function lineMarkerContent(hitInfo, pt) {
        var html = '', chart = hitInfo.series ? hitInfo.series.chart : undefined;
        if (!chart || !chart.series) {
            return html;
        }
        chart.series.forEach(function (s, i) {
            var ht = s.hitTest(new wijmo.Point(pt.x, NaN)),
        hostEle = s.hostElement, polyline;

            polyline = s.hostElement ? s.hostElement
        .getElementsByTagName("polyline")[0] : undefined;

            if (polyline && ht.x && ht.y !== null) {
                if (i == 0) {
                    html += wijmo.Globalize.formatDate(ht.x, 'dd-MMM');
                }
                html += '<div style="color:' + polyline.getAttribute('stroke')
         + '">' + ht.name + ' = ' + ht.y.toFixed(2) + '</div>';
            }
        });
        return html;
```

```
    }
</script>

<c1-flex-chart
                binding-x="Date"
                chart-type="ChartType.Line"
                width="780px"
                interpolate-nulls="true">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales In USA">
      </c1-flex-chart-series>
    <c1-flex-chart-series binding="SalesInJapan" name="Sales In Japan" >
    </c1-flex-chart-series>

    <c1-flex-chart-axis c1-property="AxisX" format="dd-MMM"></c1-flex-chart-axis>
    <c1-flex-chart-tooltip content=""></c1-flex-chart-tooltip>
    <c1-line-marker alignment="LineMarkerAlignment.Auto"
                    lines="LineMarkerLines.Vertical"
                    drag-content="true"
                    interaction="LineMarkerInteraction.Move"
                    content="lineMarkerContent">
    </c1-line-marker>
</c1-flex-chart>
```

Mixed Charts

You can add multiple series to your charts and set a different `ChartType` for all the series. Such charts are helpful in analyzing complex chart data on a single canvas. The same data can be used with different visualizations or related data can be displayed together to convey trends.

The following image shows a FlexChart with multiple series.



The following code example demonstrates how to add multiple chart series with different `ChartType`s to the FlexChart.

In Code

HTML Helpers

Razor copyCode

```
// Initialize FlexChart
@(Html.C1().FlexChart()
.Bind("Date", Model)

//Add Series to the chart
.Series(sers =>
  {
      //Add the first series
      sers.Add()
     .Binding("SalesInUSA")
     .Name("Sales in USA")
     .ChartType(C1.Web.Mvc.Chart.ChartType.Column);

      //Add the second series
      sers.Add()
     .Binding("SalesInJapan")
     .Name("Sales in Japan")
     .ChartType(C1.Web.Mvc.Chart.ChartType.Line);

      //Add the third series
      sers.Add()
     .Binding("SalesInChina")
     .Name("Sales in China")
     .ChartType(C1.Web.Mvc.Chart.ChartType.Spline);
  })
)
```

Tag Helpers

HTML copyCode

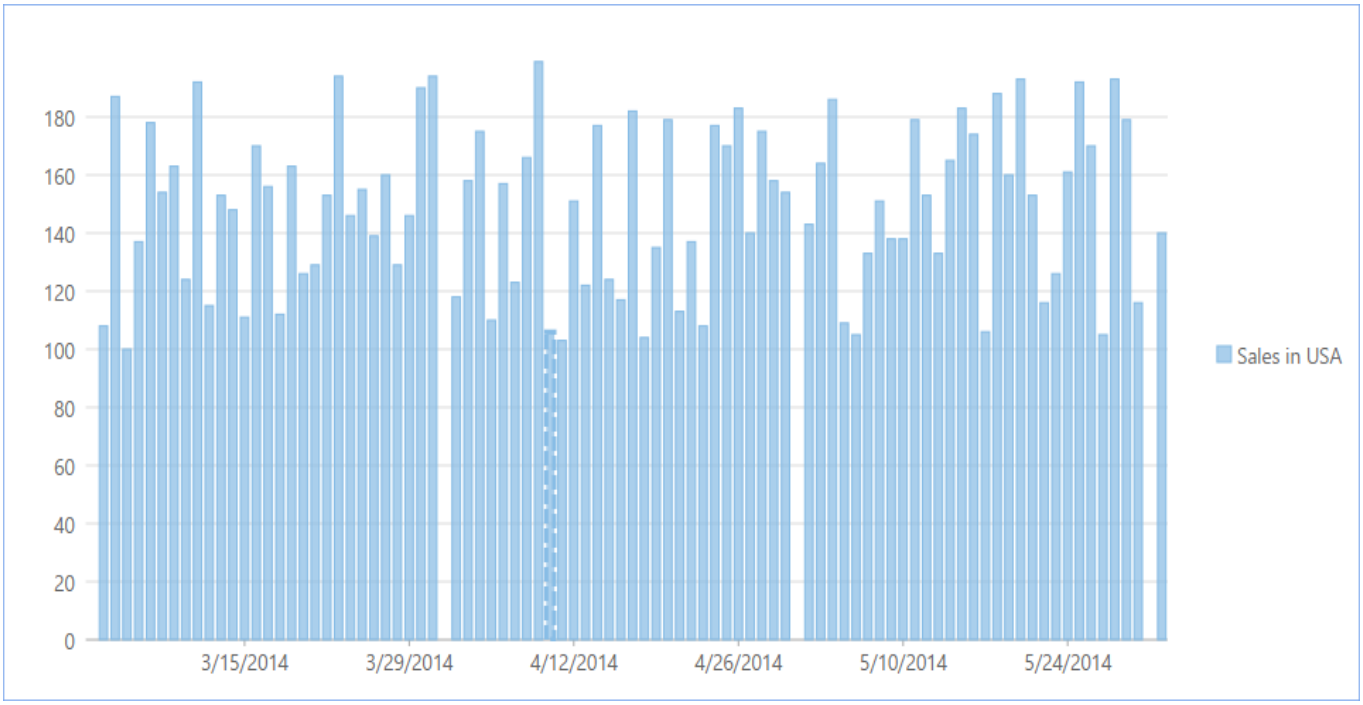
```
<c1-flex-chart binding-x="Name" chart-type="ChartType.Column" >
    <c1-items-source source-collection="Model"></c1-items-source>

    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA" chart-
type="ChartType.Column"> </c1-flex-chart-series>
    <c1-flex-chart-series binding="SalesInJapan" name="Sales in Japan" chart-
type="ChartType.Line"> </c1-flex-chart-series>
    <c1-flex-chart-series binding="SalesInChina" name="Sales in China" chart-
type="ChartType.Spline">
        </c1-flex-chart-series>
</c1-flex-chart>
```

Selection

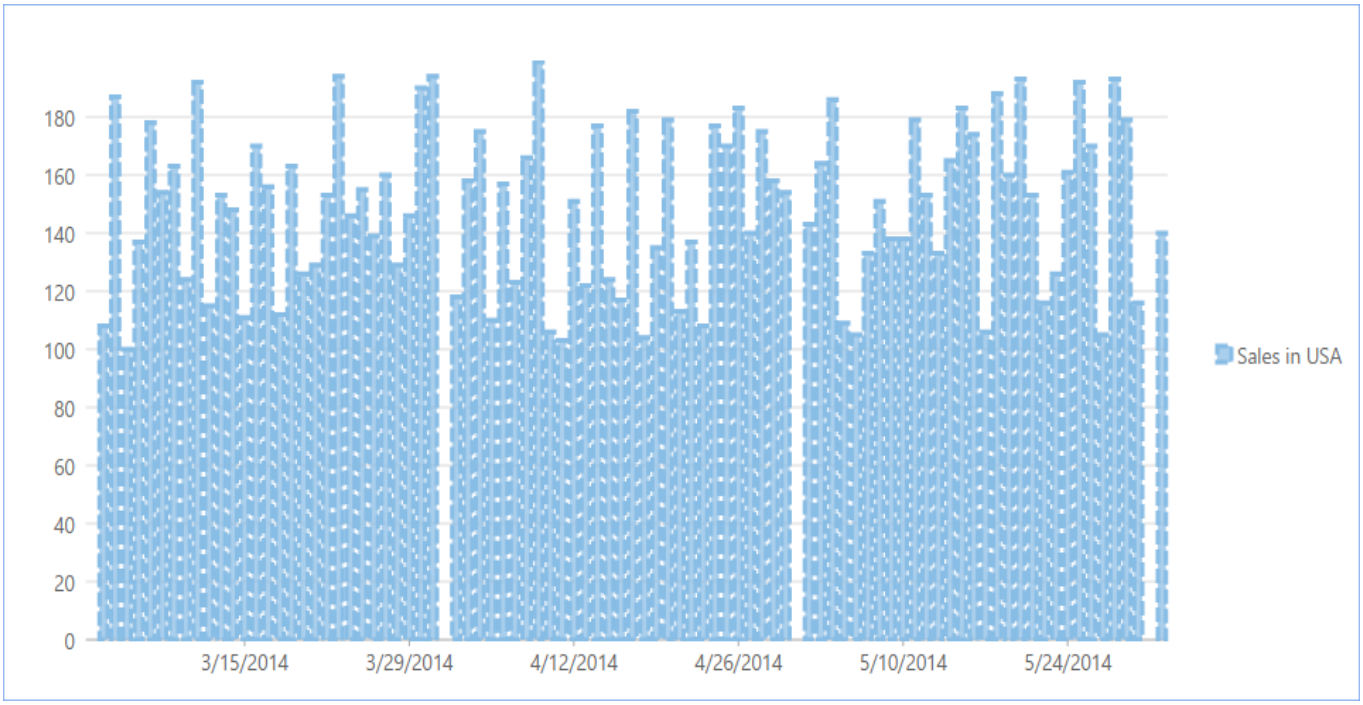
You can choose what element of the FlexChart should be selected when the user clicks on any region in a FlexChart by setting the `SelectionMode` property. This property provides three options:

- **None**: Does not select any element.
- **Point**: Highlights the point that the user clicks.
- **Series**: Highlights the series that the user clicks.

The images below show how the FlexChart appears after these properties have been set.



When SelectionMode is set to **Point**



When SelectionMode is set to **Series**

The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.SelectionMode(C1.Web.Mvc.Chart.SelectionMode.Series)
```

Tag Helpers

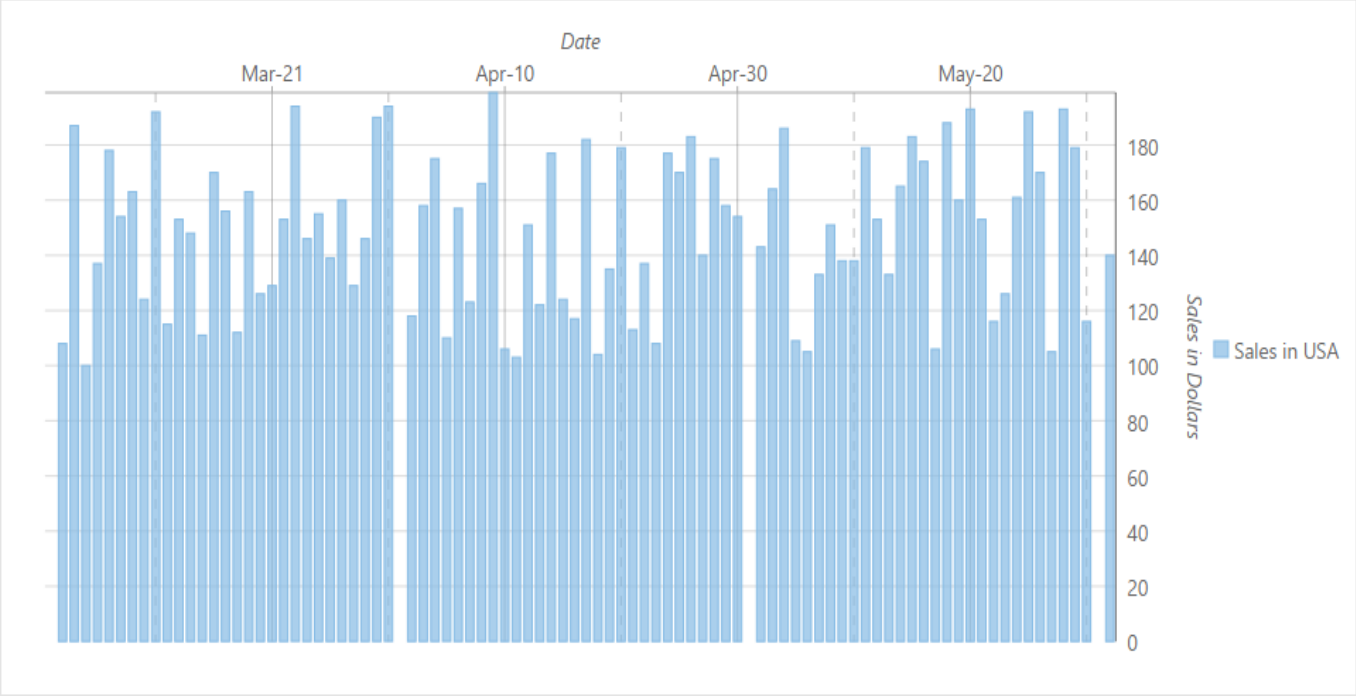
HTML copyCode

```
<c1-flex-chart selection-mode="SelectionMode.Series"></c1-flex-chart>
```

Customize Axes

An axis is composed of several elements, such as labels, line, tick marks and titles. There are several properties available in FlexChart that let you customize these elements, for both X and Y axes. You can even change the position of the axes with the help of Position property.

The image below shows a FlexChart with customized axes.



The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.AxisY(axis => axis
        .Position(C1.Web.Mvc.Chart.Position.Right)
        .MajorGrid(true).MajorUnit(20)
        .AxisLine(true)
        .Labels(true)
        .Title("Sales in Dollars")
    )
.AxisX(axis=>axis
        .Position(C1.Web.Mvc.Chart.Position.Top)
        .AxisLine(true)
        .MajorGrid(true)
        .MajorUnit(20)
        .MinorGrid(true)
        .Labels(true)
        .Title("Date")
        .Format("MMM-dd")
    )
```

Tag Helpers

HTML copyCode

```
<c1-flex-chart binding-x="Name" chart-type="ChartType.Column" >
    <c1-items-source source-collection="Model"></c1-items-source>

    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA" chart-
type="ChartType.Column">
        <c1-flex-chart-axis c1-property="AxisX" position="Position.Right"
             major-grid="true" major-unit="20" axis-line="true" labels="true"
title="Sales in Dollars" ></c1-flex-chart-axis>
    </c1-flex-chart-series>
    <c1-flex-chart-axis c1-property="AxisY" format="dd-MMM" position="Position.Top"
 major-grid="true" major-unit="20" axis-line="true" labels="true" title="Date"></c1-
flex-chart-axis>

</c1-flex-chart>
```

Hit Test

The HitTest() method is used to determine X and Y coordinates, as well as the index of a point on the FlexChart where the mouse hovers.

The following image shows how labels below the chart display the X, Y coordinate values and the index of the hit point on FlexChart.

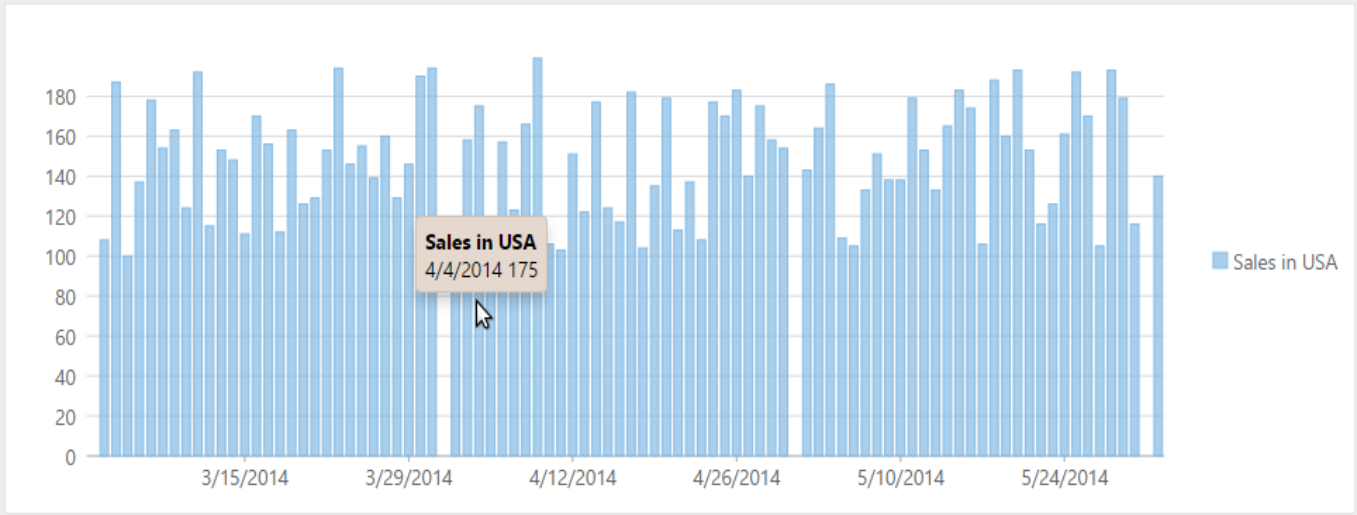


Chart element: PlotArea
Series name: Sales in USA
Point index: 33
x coordinate: 41733.03
y coordinate: 76.30

The following code examples demonstrate how to display the X, Y coordinate values and index of the hit point using simple labels.

In Code

1. Instantiate a FlexChart as shown below.

HTML Helpers

Razor copyCode

```
<div id="flexChart1" style="height: 300px"></div>
@(Html.C1().FlexChart("#flexChart1")
.Bind("Date", Model)
.ChartType(C1.Web.Mvc.Chart.ChartType.Column)
.Series(sers => sers
      .Add()
      .Binding("SalesInUSA")
      .Name("Sales in USA")
   )
)
```

Tag Helpers

HTML copyCode

```
<div id="HitTest" style="height: 300px"></div>
<c1-flex-chart id="HitTest" chart-type="ChartType.LineSymbols">
    <c1-flex-chart-series binding-x="X" binding="Y" name="cos(x)">
        <c1-items-source source-collection="cos"></c1-items-source>
    </c1-flex-chart-series>
    <c1-flex-chart-series binding-x="X" binding="Y" name="sin(x)">
```

```
        <c1-items-source source-collection="sin"></c1-items-source>
    </c1-flex-chart-series>
</c1-flex-chart>
```

2. Add a tag just below the FlexChart to display the information of the hit point as shown below.

Razor copyCode

```
<div id="info"></div>
```

3. Add the following script that retrieves the X, Y coordinate values and index of the hit point and displays them in the division added just below the chart.

JavaScript copyCode

```
<script type="text/javascript">
    c1.mvc.Utils.documentReady(function ()
    {
        var chart = wijmo.Control.getControl("#flexChart1"),
            formatHitInfo = function (hitInfo, pt)
            {
                var s = '<div><b>Chart element</b>: ' +
wijmo.chart.ChartElement[hitInfo.chartElement] + '</div>';
                if (hitInfo.series)
                {
                    s += '<div><b>Series name</b>: ' + hitInfo.series.name;
                    if (hitInfo.pointIndex !== null) {
                        s += '<div><b>Point index</b>: ' + hitInfo.pointIndex +
'</div>';
                        if (hitInfo.chartElement ==
wijmo.chart.ChartElement.PlotArea)
                        {
                            s += '<div><b>x coordinate</b>: ' + pt.x.toFixed(2)
+ '</div>';
                            s += '<div><b>y coordinate</b>: ' + pt.y.toFixed(2)
+ '</div>';
                        }
                    }
                }
                return s;
            };
        chart.hostElement.onmousemove = function (e)
        {
            var hit = chart.hitTest(e);
            var info = document.getElementById("info");
            info.innerHTML = formatHitInfo(hit, chart.pointToData(e));
        };
    });
</script>
```

Annotations

Annotations are used to mark important news or events that can be attached to a specific data point on FinancialChart. Users can hover over the event to display the full annotation text. FinancialChart's Annotation lets a user add different types of annotations on the chart, including text, shapes, images. There are various built-in shapes

which are supported as annotations in FlexChart, such as **Circle**, **Ellipse**, **Image**, **Line**, **Polygon**, **Rectangle**, **Square** and **Text**.

You can specify the position of annotation on FlexChart by setting the AnnotationPosition property to **Bottom**, **Center**, **Left**, **Right** or **Top**. To specify the attachment of annotation in FlexChart, you can use the AnnotationAttachment property, and set its value to:

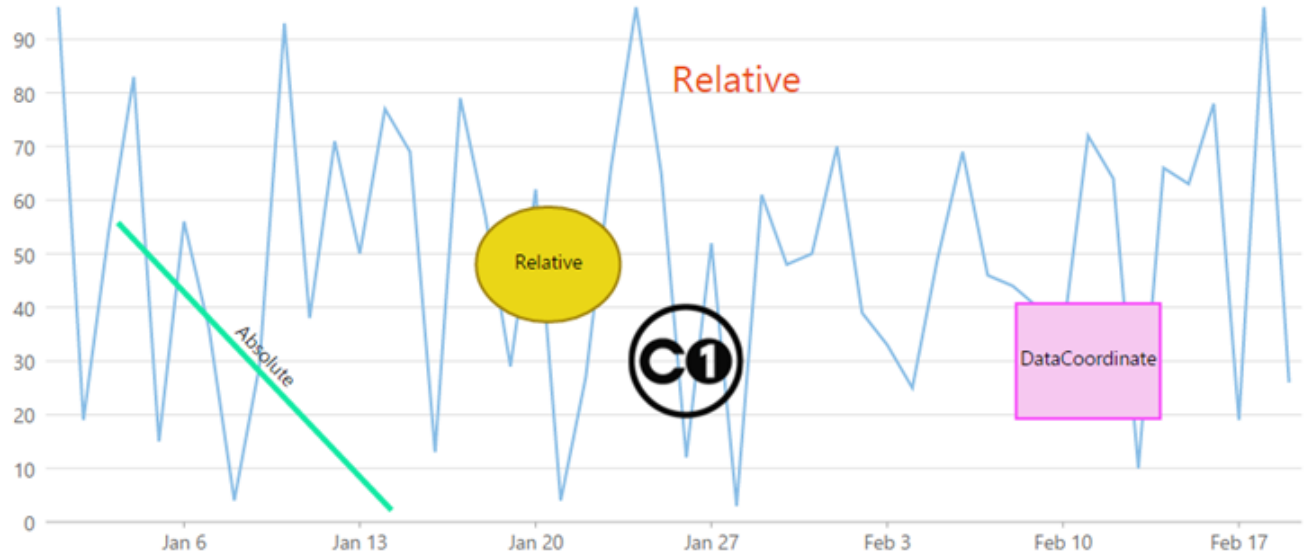
- **Absolute**: The annotation point is specified in control's pixel coordinates
- **DataCoordinate**: Annotation point is specified in data coordinates.
- **DataIndex**: Coordinates of the annotation point are defined by the data series index and the data point index.
- **Relative**: Annotation point is specified as a relative position inside the control where (0,0) is the top left corner and (1,1) is the bottom right corner.

This section describes how to add annotations to a FlexChart control in an MVC web application.

This topic comprises of three steps:

- **Step 1: Create a Datasource for FlexChart**
- **Step 2: Initialize a FlexChart control and add Annotations to it**
- **Step 3: Build and Run the Project**

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



Step 1: Create a Datasource for FlexChart

1. Add a new class to the folder **Models** (for example: `Sale.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes that serve as a datasource for the FlexChart control.

C# copyCode

```
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; }

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

    /// <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 startDate = new DateTime(dt.Year, i % 12 + 1, 25);
            var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60,
i % 60);

            return new Sale
            {
                ID = i + 1,
                Start = startDate,
                End = endDate,
                Country = country,
                Product = product,
                Color = color,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Discount = Math.Round(rand.NextDouble() / 4, 2),
                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;
```

```
    }

    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;
    }
}


public class BasicSale
{
    public int Sale { get; set; }
    public DateTime Date { get; set; }

    public BasicSale(int sale, DateTime date)
    {
        Sale = sale;
        Date = date;
    }
    public static List<BasicSale> GetBasicSales()
    {
        List<BasicSale> list = new List<BasicSale>();
        int[] sales = {
        96, 19, 54, 83, 15, 56, 36, 4, 29, 93,
        38, 71, 50, 77, 69, 13, 79, 57, 29, 62,
        4, 27, 66, 96, 65, 12, 52, 3, 61, 48, 50,
        70, 39, 33, 25, 49, 69, 46, 44, 40, 35,
        72, 64, 10, 66, 63, 78, 19, 96, 26};
        for (int i = 0; i < sales.Length; i++)
        {
            list.Add(new BasicSale(sales[i], new DateTime(2014, i / 31 + 1, i %
31 + 1)));
        }
        return list;
    }
}
```

Back to Top

Step 2: Initialize a FlexChart control and add Annotations to it

Complete the following steps to initialize a FlexChart control and add annotations to it.

Add a new Controller

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: `AnnotationController`).
    3. Click **Add.**
4. Replace the method `Index()` with the following method.

C# copyCode

```
public ActionResult Index()
{
    return View(BasicSale.GetBasicSales());
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller `AnnotationController` to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.
6. Instantiate a FlexChart control in the view QuickStart as shown below.

> The example uses C1logo image to display image annotation for FlexChart. The image is available on your system in *C:\Users\<username>\Documents\ComponentOne Samples\Studio for ASP.NET MVC\MVC\MvcExplorer\Content\images\c1logo.png*.

HTML Helpers

Razor copyCode

```
@using MVCFinancialChart.Models
@using C1.Web.Mvc
@using C1.Web.Mvc.Chart
@using System.Drawing
@model IEnumerable<BasicSale>

@*Add data to be displayed on tooltip*@

@{
    var lineTooltip = "This is line annotation. </br> start: { x: 50, y: 150 } end: { x: 240, y: 350 } </br> attachment: Absolute";
    var textTooltip = "This is text annotation.</br> point: { x: 0.55, y: 0.15 } </br> attachment: Relative";
    var ellipseTooltip = "This is ellipse annotation.</br> point: { x: 0.4, y: 0.5 }</br> attachment: Relative";
    var rectTooltip = "This is rectangle annotation.</br> point:{ x: new DateTime(2014, 2, 11), y: 30 }</br> attachment: DataCoordinate";
    var imageTooltip = "This is image annotation.</br> point:{x: new DateTime(2014, 1, 26), y: 30}</br> attachment: DataCoordinate";
}
```

```
@*Initialize a FlexChart*@

@(Html.C1().FlexChart().Bind(Model)
.BindingX("Date").Series(series =>
{
    series.Add().Binding("Sale").ChartType(ChartType.Line);
})
//Add Annotation layer
.AddAnnotationLayer(layer =>
{
    //Add line annotation
    layer.AddLine(line => line.Attachment(AnnotationAttachment.Absolute)
        .Position(AnnotationPosition.Center)
        .Content("Absolute")
        .Start(new DataPoint(50, 150))
        .End(new DataPoint(240, 350))
        .Tooltip(lineTooltip)
        .Style(style => style.Stroke("#17EAA5").StrokeWidth(4)
        ));
    //Add text annotation
    layer.AddText(text => text.Attachment(AnnotationAttachment.Relative)
        .Content("Relative")
        .Position(AnnotationPosition.Center)
        .Point(new DataPoint(0.55, 0.15))
        .Tooltip(textTooltip)
        .Style(style => style.Fill("#EA4C17").FontSize(26))
        );
    //Add ellipse annotation
    layer.AddEllipse(ellipse =>
ellipse.Attachment(AnnotationAttachment.Relative)
        .Position(AnnotationPosition.Center)
        .Point(new DataPoint(0.4, 0.5))
        .Width(100).Height(80)
        .Content("Relative")
        .Tooltip(ellipseTooltip)
        .Style(style => style.Fill("#EAD617").Stroke("#A1840D"))
        );
    //Add rectangle annotation
    layer.AddRectangle(rectangle =>
rectangle.Attachment(AnnotationAttachment.DataCoordinate)
        .Point(new DataPoint(new DateTime(2014, 2, 11), 30))
        .Content("DataCoordinate")
        .Position(AnnotationPosition.Center)
        .Tooltip(rectTooltip)
        .Style(style => style.Fill("#F6C8F0").Stroke("#F947FB"))
        );
    //Add image annotation
    layer.AddImage(image =>
image.Attachment(AnnotationAttachment.DataCoordinate)
        .Href("/Content/images/c1logo.png")
```

```
        .Point(new DataPoint(new DateTime(2014, 1, 26), 30))
        .Width(80).Height(80)
        .Tooltip(imageTooltip)
        );
})
)
```

Tag Helpers

HTML

```
//TagHelperExplorer is the name of the MVC application.

@using TagHelperExplorer.Models
@using C1.Web.Mvc
@using C1.Web.Mvc.Chart
@model IEnumerable<BasicSale>
@{

    var lineTooltip = "This is line annotation. </br> start: { x: 50, y: 150 }
end: { x: 240, y: 350 } </br> attachment: Absolute";
    var textTooltip = "This is text annotation.</br> point: { x: 0.55, y: 0.15 }
</br> attachment: Relative";
    var ellipseTooltip = "This is ellipse annotation.</br> point: { x: 0.4, y:
0.5 }</br> attachment: Relative";
    var rectTooltip = "This is rectangle annotation.</br> point:{ x: new
DateTime(2014, 2, 11), y: 30 }</br> attachment: DataCoordinate";
    var imageTooltip = "This is image annotation.</br> point:{x: new
DateTime(2014, 1, 26), y: 30}</br> attachment: DataCoordinate";
    var lineStyle = new SVGStyle { Stroke = "#17EAA5", StrokeWidth = 4 };
    var textStyle = new SVGStyle { Fill = "#EA4C17", FontSize = 26 };
    var ellipseStyle = new SVGStyle { Fill = "#EAD617", Stroke = "#A1840D" };
    var rectStyle = new SVGStyle { Fill = "#F6C8F0", Stroke = "#F947FB" };
}
<c1-flex-chart binding-x="Date">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="Sale" chart-type="ChartType.Line">
    </c1-flex-chart-series>
    <c1-annotation-layer>
        <c1-annotation-line start="new DataPoint(50,150)" end="new
DataPoint(240, 350)"
                            tooltip="@lineTooltip" style="lineStyle"
content="Absolute">
        </c1-annotation-line>
        <c1-annotation-text attachment="AnnotationAttachment.Relative"
                            point="new DataPoint(0.55,0.15)" content="Relative"
                            tooltip="@textTooltip" style="textStyle">
        </c1-annotation-text>
        <c1-annotation-ellipse attachment="AnnotationAttachment.Relative"
                               width="100" height="80" content="Relative"
                               point="new DataPoint(0.4,0.5)"
```

```
                                tooltip="@ellipseTooltip" style="ellipseStyle">
        </c1-annotation-ellipse>
        <c1-annotation-rectangle
attachment="AnnotationAttachment.DataCoordinate"
                                point="new DataPoint(new DateTime(2014, 2, 11),
30)"
                                content="DataCoordinate" tooltip="@rectTooltip"
style="rectStyle">
        </c1-annotation-rectangle>
        <c1-annotation-image attachment="AnnotationAttachment.DataCoordinate"
                             href="/Content/images/c1icon.png"
                             point="new DataPoint(new DateTime(2014, 1, 26),
30)"
                             width="80" height="80" tooltip="@imageTooltip">
        </c1-annotation-image>
    </c1-annotation-layer>
</c1-flex-chart>
```

Back to Top

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Annotation/Index**) in the address bar of the browser to see the view.

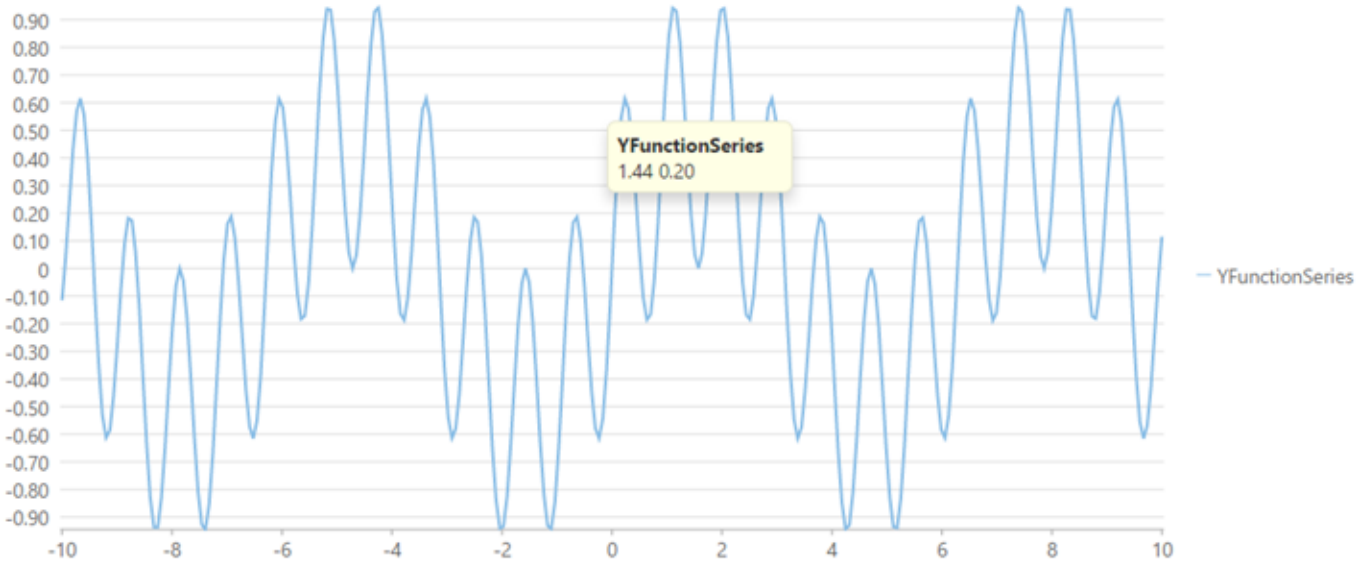
Back to Top

Function Series

FlexChart and FinancialChart let a user create analytical charts with function series using different formulae. You can add Y function series or Parametric function series using **AddYFunctionSeries** and **AddParametricFunctionSeries**. YFunctionSeries calculates y by a given function, and PfunctionSeries calculates X and Y by given x and y functions. Users can plot any type of curve basis the values and formula. In these examples, different formulae are used that compute the sine and cos values, and return a value to be displayed on the chart.

The image below shows how FlexChart appears with Y function series.

Y Function Series



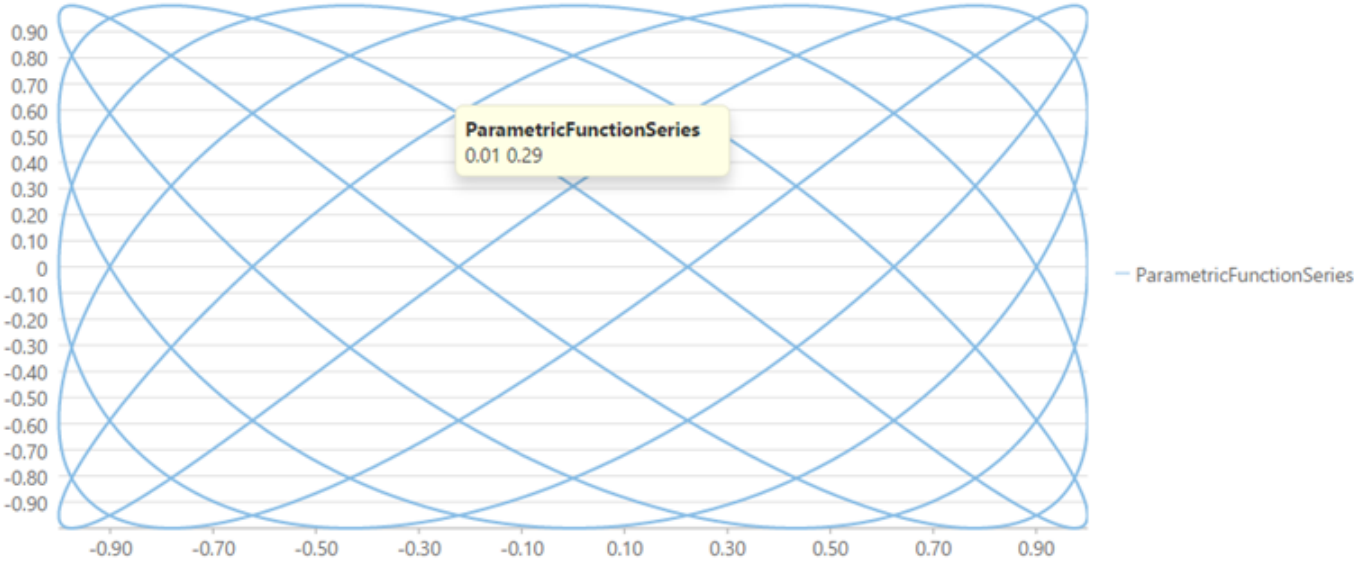
The following code example demonstrates how to use Y function series to analyze data on FinancialChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Chart

<div>
    <div>
        <script type="text/javascript">
            function yFunc(value) {
                return Math.sin(4 * value) * Math.cos(3 * value);
            }
        </script>
        @(Html.C1().FlexChart().Legend(Position.Right)
            .Series(ses =>
            {
                ses.AddYFunctionSeries("YFunctionSeries").Min(-
10).Max(10).SampleCount(300).Func("yFunc");
            }))
    </div>
</div>
```

Parametric Function Series



The following code example demonstrates how to use parametric function series to analyze data on FinancialChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

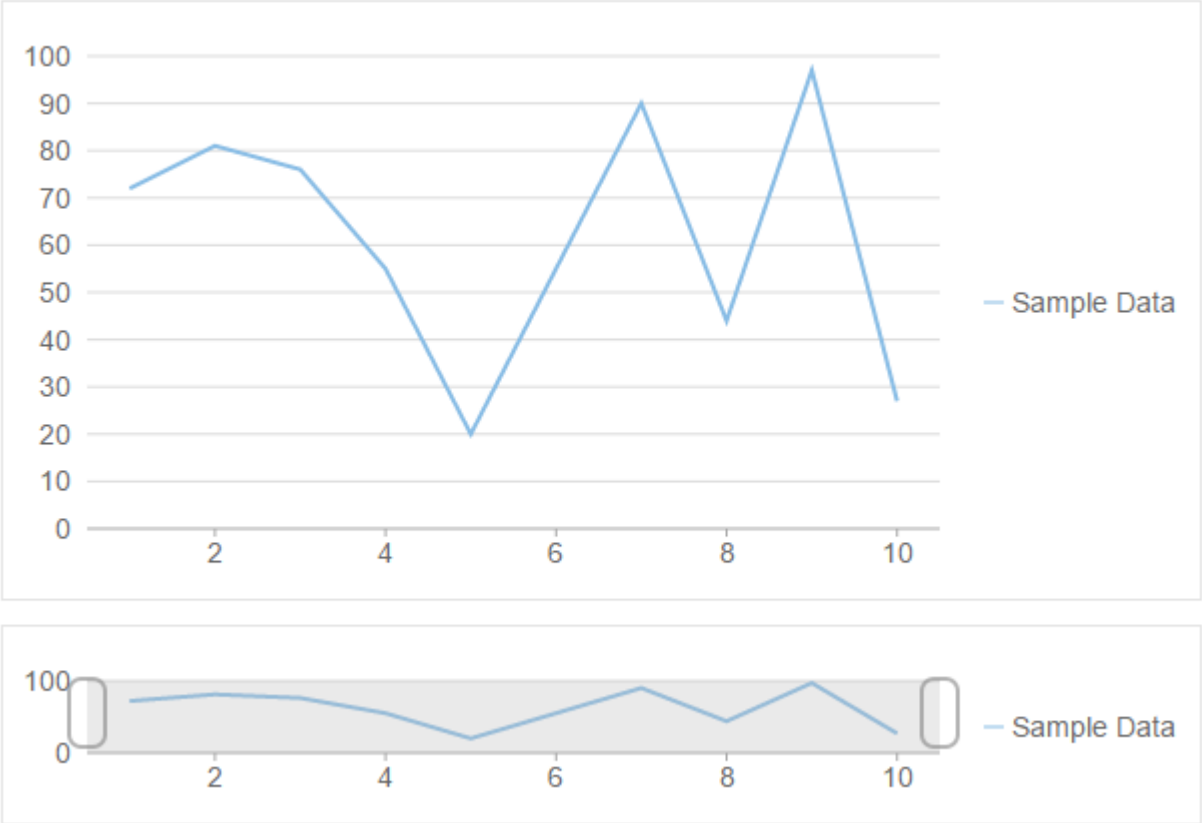
```
@using C1.Web.Mvc.Chart

<div>
    <div>
        <script type="text/javascript">
            var xParam = 5, yParam = 7;
            function xFunc(value) {
                return Math.cos(value * xParam);
            }
            function yFunc(value) {
                return Math.sin(value * yParam);
            }
        </script>
        @(Html.C1().FlexChart().Legend(Position.Right)
            .Series(ses =>
            {
                ses.AddParametricFunctionSeries("ParametricFunctionSeries")
                    .Max(2 * Math.PI)
                    .SampleCount(1000)
                    .XFunc("xFunc")
                    .YFunc("yFunc");
            }))
    </div>
</div>
```

Range Selector

FlexChart's Range Selector lets you select a range of numeric data with lower value thumb and upper value thumb. These thumbs define the start and end values of the range. On dragging the thumb towards left (or down) on the range bar, you reduce its value, and dragging it towards the right (or up) increases the value on the range bar. You can set the minimum and maximum bounds for thumb values using `Min` and `Max` properties.

The thumb value must be more than the minimum bound and less than the maximum bound set for the range selector. You can also set the position of a range selector using the `Orientation` property.

The image below shows how the FlexChart appears after these properties have been set.



The following code example demonstrates how to display thumb values using a range selector on the FlexChart.

In Code

HTML Helpers

Example Title

```
@model List<MathPoint>
@(Html.C1().FlexChart().Id("trendChart")
    .Bind("X", Model)
    .AxisY(ay => ay.Min(0).Max(100))
    .Legend(Position.Right)
    .Width(600)
    .Height(300)
    .LegendToggle(true)
    .Series(ss =>
    {
```

```
        ss.Add(ChartType.Line, "Sample Data").Binding("Y");

    })
)
<script type="text/javascript">
    function rangeChanged(rs, ui) {
        updateStChartRange(rs.min, rs.max);
    }
</script>

@(Html.C1().FlexChart()

    .Bind("X", Model)
    .AxisY(ay => ay.Min(0).Max(100))
    .Legend(Position.Right)
    .Width(600)
    .Height(100)
    .LegendToggle(true)
    .Series(ss =>
    {
        ss.Add(ChartType.Line, "Sample Data").Binding("Y");

    })
     .AddRangeSelector(rs => rs.Min(0).Max(100).OnClientRangeChanged("rangeChanged"))
     )
```

Tag Helpers

Example Title

```
<c1-flex-chart binding-x="Name" chart-type="ChartType.Line" legend-
position="Position.Right"
             width="600px" height="300px" legend-toggle="true">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="Y" name="Sample Data"></c1-flex-chart-series>
</c1-flex-chart>
  <script type="text/javascript">
    function rangeChanged(rs, ui) {
    updateStChartRange(rs.min, rs.max);
    }
</script>

<c1-flex-chart binding-x="Name" chart-type="ChartType.Line" legend-
position="Position.Right"
               width="600px" height="100px" legend-toggle="true">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="Y" name=Sample Data"></c1-flex-chart-series>

    <c1-range-selector is-visible="true" min="0" max="100" range-
changed="rangeChanged" ></c1-range-selector>
</c1-flex-chart>
```

Trendline

Trendlines are used to represent trends in data and to examine problems of prediction. Trendlines are commonly used with price charts or financial charts, but they can also be used with a variety of technical analysis charts such as MACD (moving average convergence/divergence) which is a trading indicator used in technical analysis of stock prices, or RSI (relative strength index) which is a technical indicator used in the analysis of financial markets.

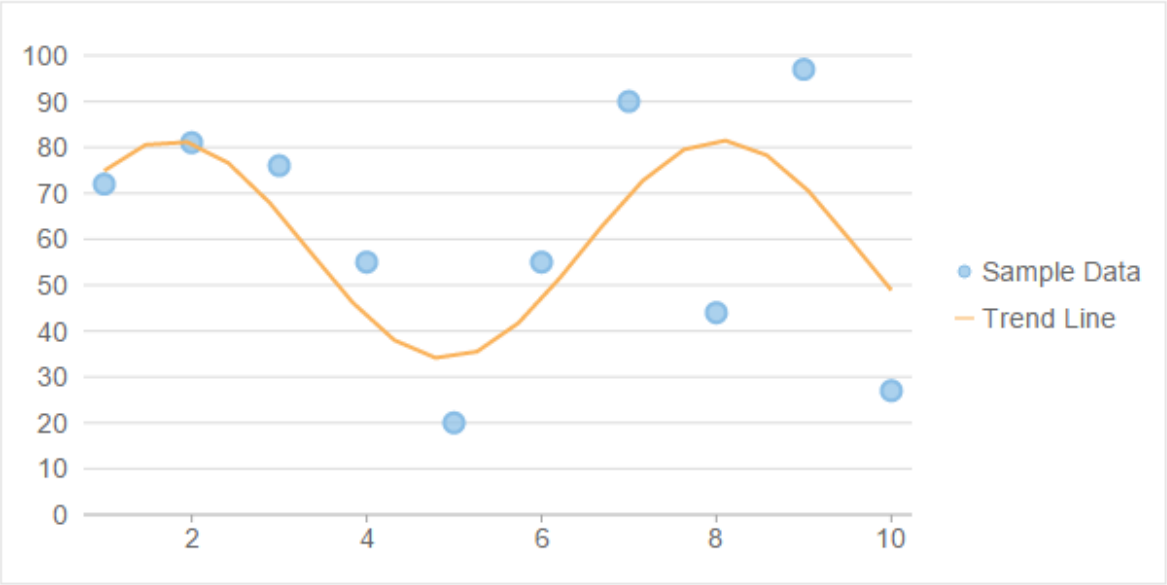
Types of Trendlines

The following table displays the supported FitTypes. Each trend type is drawn based on the calculation formula of its type.

Type	Description
Average X	Calculates the average value of X from the chart data and draws a trendline.
Average Y	Calculates the average value of Y from the chart data and draws a trendline.
Exponential	A curved line that is convenient to use when data values rise or fall at increasingly higher rates. You cannot create an exponential trendline if your data contains zero or negative values.
Fourier	A way to display a wave like function as a combination of simple sine waves. It is created by using the fourier series formula.
Linear	A linear trendline is a best-fit straight light. Your data is linear if the data point pattern resembles a line, and a linear trendline is a good fit if the R-squared value is at or near 1.
Logarithmic	A best fit curved line used for better visualization of data. Used when the rate of change in the data increases or decreases quickly and then levels out. It can also use positive and negative values.
Max X	Takes the maximum value of X from the chart and draws a trendline using it.
Max Y	Takes the maximum value of Y from the chart and draws a trendline using it.
Min X	Takes the minimum value of X from the chart and draws a trendline using it.
Min Y	Takes the minimum value of Y from the chart and draws a trendline using it
Polynomial	A twisted line that is used when data oscillates. It is useful for analyzing gains and losses over a large data set.
Power	A curved line that is best used with data sets that compare calculation that increase at a peculiar rate. For example, the acceleration of a vehicle at one-second intervals.

You can add trendlines to your charts to display trends of data. Using this, you can easily analyze the increase or decrease of your Y data over your X data.

The image below shows how trendline, with FitType set to Fourier, appears on the FlexChart.



The following code example demonstrates how to add a trendline to the FlexChart. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Example Title

```
@using C1.Web.Mvc.Chart
@model List<MathPoint>

@(Html.C1().FlexChart().Id("trendChart")
    .Bind("X", Model)
    .AxisY(ay => ay.Min(0).Max(100))
    .Legend(Position.Right)
    .Width(600)
    .Height(300)
    .LegendToggle(true)
    .Series(ss =>
    {
   ss.Add(ChartType.Scatter, "Sample Data").Binding("Y");
     ss.AddTrendLine("Trend
Line").Binding("Y").SampleCount(20).TrendLineOrder(3).FitType(TrendLineFitType.Fourier);
    }))
```

Tag Helpers

Example Title

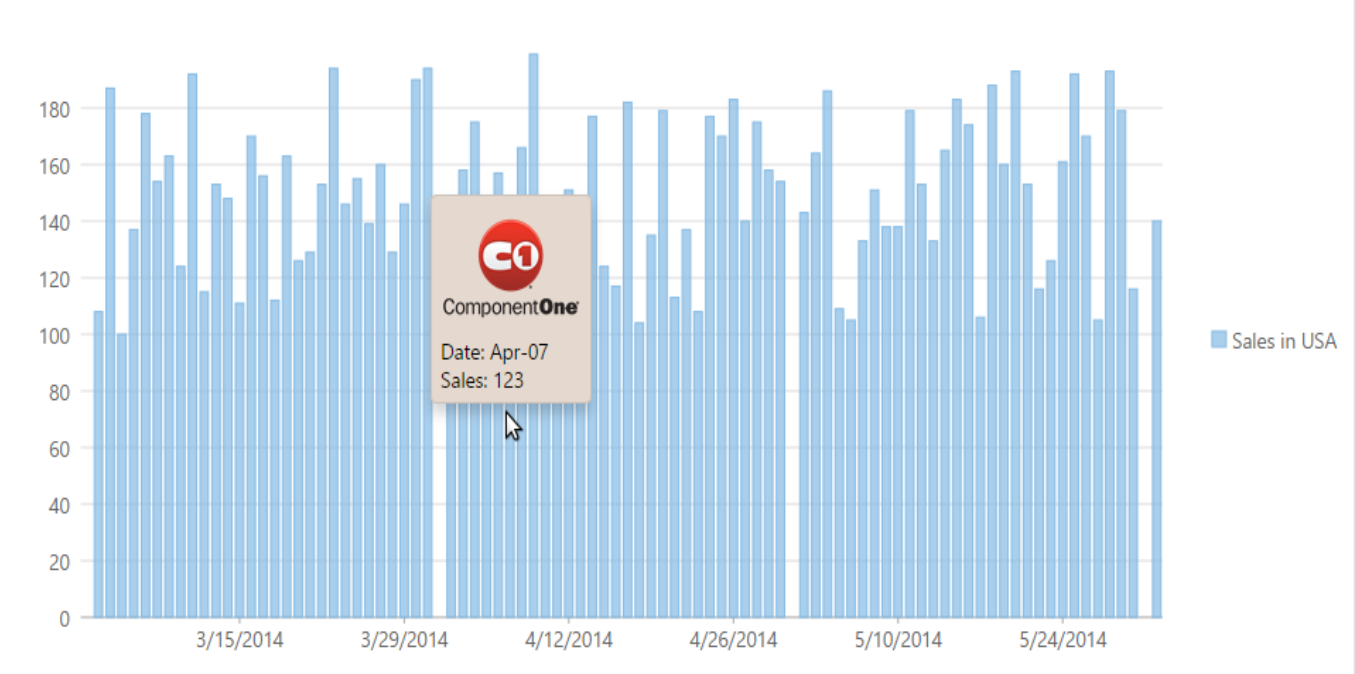
```
@model List<MathPoint>
<c1-flex-chart binding-x="Name"chart-type="ChartType.Scatter" legend-
position="Position.Right"
               width="600px" height="300px" legend-toggle="true">
```

```
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="Y" name="Sales in USA"></c1-flex-chart-series>
        <c1-flex-chart-trendline binding="Y" name="Trend Line" sample-count="20" trend-
line-order="3" fit-type="TrendLineFitType.Fourier">
        </c1-flex-chart-trendline>
</c1-flex-chart>
```

Tooltip

By default, a tooltip generally displays the name of the legend along with the X and Y values of the selected point. FlexChart allows you to display customized tooltips that may contain labels or images by setting the `Content` property.

The image below shows how a customized tooltip appears on FlexChart.



The following code examples demonstrate how to customize the tooltip.

In Code

To accomplish this, you must first create the content to be added to the tooltip. It can include labels, images and other similar elements.

Javascript copyCode

```
<script type="text/javascript">
    var tooltipContent = function (ht)
    {
     return "<img src='../C1.png' alt='Others'  />"
            + '<br/>' + "Date: " + wijmo.Globalize.format(ht.x, 'MMM-dd')
            + '<br/>' + "Sales: " + ht.y;
    };
</script>
```

Assign the view to the chart tooltip as shown below. The following example uses the sample created in the Quick Start

section.

HTML Helpers

Razor copyCode

```
.Tooltip(tooltip => tooltip.Content("tooltipContent"))
```

Tag Helpers

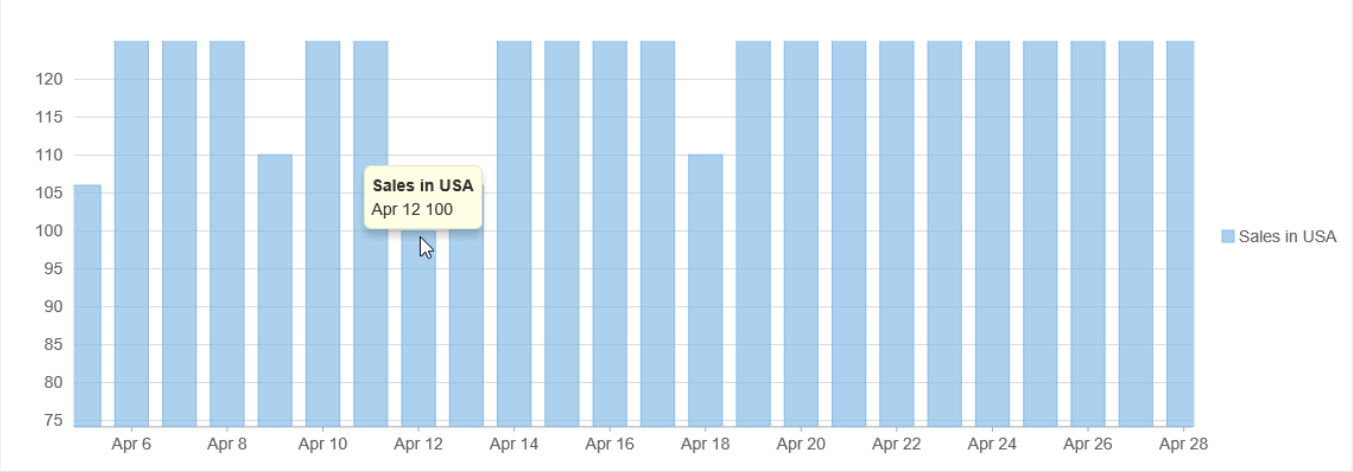
HTML copyCode

```
    <c1-flex-chart-tooltip content="tooltipContent"></c1-flex-chart-tooltip>
</c1-flex-chart>
```

Chart Gestures

In FlexChart, you can perform two types of chart interactions - zooming and panning. The zoom and pan features are especially important where there is large amount of data. Zoom allows you to zoom-in and zoom-out of the selected chart area while pan allows you to navigate through the chart area. Zooming and panning also supports gestures that allows you to perform these operations smoothly on your devices.

The image below shows how the FlexChart appears, after chart gesture for zooming is used in your browser.



The following code example demonstrates how to enable Chart Gesture for zooming and panning modes. This example uses the sample created in the Quick Start section.

ASP.NET

Razor

```
@using ChartAnimation.Models
@model IEnumerable<FruitSale>
@(Html.C1().FlexChart()
.Bind("Date", Model)
.ChartType(C1.Web.Mvc.Chart.ChartType.Column)
.Series(sers =>
{    sers.Add()
```

```
    .Binding("SalesInUSA")
    .Name("Sales in USA");
})
.SupportGestures(cg =>
cg.InteractiveAxes(InteractiveAxes.XY).MouseAction(MouseAction.Zoom)))
```

ASP.NET Core

Razor

```
<c1-flex-chart binding-x="Name"chart-type="ChartType.Column" >
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA">
    </c1-flex-chart-series>
    <c1-chart-gestures interactive-axes=”InteractiveAxes.XY” mouse-
action=”MouseAction.Zoom” />
</c1-flex-chart>
```

Chart Animation

The ChartAnimation provides built-in animation while loading and updating the chart. The animation can be configured by the user through several properties that include duration, easing function, and animation mode. Chart Animation property is available for FlexChart, FinancialChart and FlexPie.

The following code example demonstrates how to enable Chart Animation using several properties such as duration, easing, and animation mode . This example uses the sample created in the Quick Start section.

ASP.NET

Razor

```
@using ChartAnimation.Models
@model IEnumerable<FruitSale>
@(Html.C1().FlexChart()
.Bind("Date", Model)
.ChartType(C1.Web.Mvc.Chart.ChartType.Column)
.Series(sers =>
{       sers.Add()
   .Binding("SalesInUSA")
   .Name("Sales in USA");
})
.ShowAnimation(ca =>
ca.AnimationMode(AnimationMode.All).AxisAnimation(true).Duration(1000).Easing(Easing.Linear)))
)
```

ASP.NET Core

Razor

```
<c1-flex-chart binding-x="Name"chart-type="ChartType.Column" >
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales in USA">
    </c1-flex-chart-series>
```

```
    <c1-chart-animation animation-mode="AnimationMode.All" duration="1000" />
</c1-flex-chart>
```

Chart Animation Class

You can make use of the following properties to enable animation in your charts.

Name	Type	Description
AnimationMode	Animation Mode	Gets or sets whether the plot points animate one at a time, series by series, or all at once. The whole animation is still completed within the duration.
Axis Animation	Boolean	Gets or sets a value indicating whether animation is applied to the axis.
Duration	Integer	Gets or sets the length of entire animation in milliseconds.
Easing	Easing	Gets or sets the easing function applied to the animation.

AnimationMode Enumeration

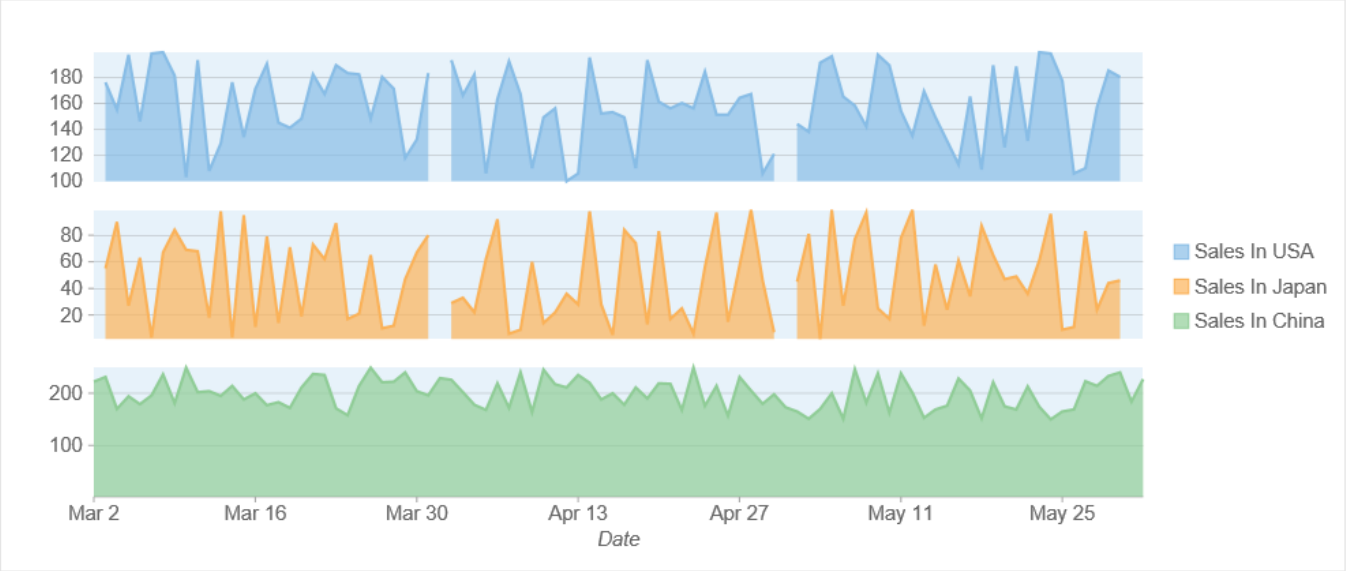
You can use the AnimationMode to apply the animation to the Points, Series or both.

Name	Description
All	All points and series are animated at once.
Point	Animation is performed point by point. Multiple series are animated simultaneously at the same time.
Series	Animation is performed series by series. Entire series is animated at once, following the same animation as the "All" option, but just one series at a time.

Plot Areas

In FlexChart, you can create different plot areas for different series within a single chart area. This increases the viewability of data by displaying each series in a separate plot area across one axis, keeping other axis fixed.

The following image displays a chart that consists of three plot areas across Y axis:



The following code example demonstrates how to create plot areas in FlexChart. This example uses the sample created in the Quick Start section.

HTML Helpers

Razor copyCode

```
@using C1MvcWebAppPlotAreas.Models
@model IEnumerable<FruitSale>

<script>
    var flexChart;
    c1.documentReady(function () {
        flexChart = wijmo.Control.getControl('#flexChart');

        addPlotArea(0, "plot1", { fill: 'rgba(136,189,230,0.2)' }, null);
        addPlotArea(1, "div1",null, 20);
        addPlotArea(2, "plot2", { fill: 'rgba(136,189,230,0.2)' }, null);
        addPlotArea(3, "div2", null, 20);
        addPlotArea(4, "plot3", { fill: 'rgba(136,189,230,0.2)' }, null);
        flexChart.series[1].axisY.plotArea =
flexChart.plotAreas.getPlotArea('plot2');
        flexChart.series[2].axisY.plotArea =
flexChart.plotAreas.getPlotArea('plot3');
    });

    function addPlotArea(row, name, style, height) {
        var plotArea = new wijmo.chart.PlotArea();
        plotArea.row = row;
        plotArea.name = name;
        if (style) {
            plotArea.style = style;
        }

        if (height) {
            plotArea.height = height;
        }

        flexChart.plotAreas.push(plotArea);
    }
</script>

@(Html.C1().FlexChart().Id("flexChart").ChartType(C1.Web.Mvc.Chart.ChartType.Area)
    .CssClass("chart").Bind(Model).BindingX("Date")
    .AxisX(axis => axis.Title("Date"))
    .Series(sers =>
    {
        sers.Add().Binding("SalesInUSA").Name("Sales In USA");
        sers.Add().Binding("SalesInJapan").Name("Sales In Japan").AxisY(axis =>
            axis.Position(C1.Web.Mvc.Chart.Position.Left).MajorGrid(true));
        sers.Add().Binding("SalesInChina").Name("Sales In China").AxisY(axis =>
            axis.Position(C1.Web.Mvc.Chart.Position.Left).MajorGrid(true));
    })
)
```

Tag Helpers

HTML

```
@using C1MvcWebAppPlotAreas.Models
@model IEnumerable<FruitSale>
<script>
    var flexChart;
    c1.documentReady(function () {
        flexChart = wijmo.Control.getControl('#flexChart');
        addPlotArea(0, "plot1", { fill: 'rgba(136,189,230,0.2)' }, null);
        addPlotArea(1, "div1",null, 20);
        addPlotArea(2, "plot2", { fill: 'rgba(136,189,230,0.2)' }, null);
        addPlotArea(3, "div2", null, 20);
        addPlotArea(4, "plot3", { fill: 'rgba(136,189,230,0.2)' }, null);
        flexChart.series[1].axisY.plotArea =
flexChart.plotAreas.getPlotArea('plot2');
        flexChart.series[2].axisY.plotArea =
flexChart.plotAreas.getPlotArea('plot3');
    });
    function addPlotArea(row, name, style, height) {
        var plotArea = new wijmo.chart.PlotArea();
        plotArea.row = row;
        plotArea.name = name;
        if (style) {
            plotArea.style = style;
        }
        if (height) {
            plotArea.height = height;
        }
        flexChart.plotAreas.push(plotArea);
    }
</script>
@(Html.C1().FlexChart().Id("flexChart").ChartType(C1.Web.Mvc.Chart.ChartType.Area)
    .CssClass("chart").Bind(Model).BindingX("Date")
    .AxisX(axis => axis.Title("Date"))
    .Series(sers =>
    {
        sers.Add().Binding("SalesInUSA").Name("Sales In USA");
        sers.Add().Binding("SalesInJapan").Name("Sales In USA").AxisY(axis =>
            axis.Position(C1.Web.Mvc.Chart.Position.Left).MajorGrid(true));
        sers.Add().Binding("SalesInChina").Name("Sales In USA").AxisY(axis =>
            axis.Position(C1.Web.Mvc.Chart.Position.Left).MajorGrid(true));
    })
)
<c1-flex-chart id="flexChart" chart-type="C1.Web.Mvc.Chart.ChartType.Area"
class="chart" binding-x="Date">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-axis c1-property="AxisX" title="Date"></c1-flex-chart-axis>
    <c1-flex-chart-series binding="SalesInUSA" name="Sales In USA"></c1-flex-chart-
```

```
series>
    <c1-flex-chart-series binding="SalesInJapan" name="Sales In Japan">
        <c1-flex-chart-axis c1-property="AxisY"
position="C1.Web.Mvc.Chart.Position.Left" major-grid="true"></c1-flex-chart-axis>
    </c1-flex-chart-series>
    <c1-flex-chart-series binding="SalesInChina" name="Sales In China">
        <c1-flex-chart-axis c1-property="AxisY"
position="C1.Web.Mvc.Chart.Position.Left" major-grid="true"></c1-flex-chart-axis>
    </c1-flex-chart-series>
</c1-flex-chart>
```

FlexChart ASP.NET Core Tags

FlexChart control supports the following ASP.NET Core Tags:

FlexChart class

- c1-flex-chart-axis
- binding
- binding-x
- chart-type
- class
- style
- c1-flex-chart-datalabel
- footer
- c1-flex-chart-title-style
- header
- c1-flex-chart-title-style
- height
- id
- interpolate-nulls
- item-formatter
- c1-items-source
- legend-position
- legend-toggle
- rendered
- selection-changed
- series-visibility-changed
- bubble-max-size
- bubble-min-size
- group-width
- palette
- plot-margin
- rotated
- selection-index
- selection-mode
- c1-flex-chart-series
- stacking
- symbol-size
- template-bindings
- c1-flex-chart-tooltip

- width

AxisX

- axis-line
- binding
- format
- item-formatter
- items-source
- label-angle
- labels
- major-grid
- major-tick-marks
- major-unit
- max
- min
- minor-grid
- minor-tick-marks
- minor-unit
- name
- range-changed
- origin
- overlapping-labels
- position
- reversed
- title

AxisY

- axis-line
- binding
- format
- item-formatter
- items-source
- label-angle
- labels
- major-grid
- major-tick-marks
- major-unit
- max
- min
- minor-grid
- minor-tick-marks
- minor-unit
- name
- range-changed
- origin
- overlapping-labels
- position
- reversed
- title

DataLabel

- border
- content

- position

FooterStyle

- fill
- font-family
- font-size
- foreground
- halign

HeaderStyle

- fill
- font-family
- font-size
- foreground
- halign

Series

- binding
- binding-x
- chart-type
- class
- ItemsSource
- name
- style
- symbol-marker
- symbol-size
- symbol-style
- visibility

Series.AxisX

- axis-line
- binding
- format
- item-formatter
- items-source
- label-angle
- labels
- major-grid
- major-tick-marks
- major-unit
- max
- min
- minor-grid
- minor-tick-marks
- minor-unit
- name
- range-changed
- origin
- overlapping-labels
- position
- reversed
- title

Series.AxisY

- axis-line
- binding
- format
- item-formatter
- items-source
- label-angle
- labels
- major-grid
- major-tick-marks
- major-unit
- max
- min
- minor-grid
- minor-tick-marks
- minor-unit
- name
- range-changed
- origin
- overlapping-labels
- position
- reversed
- title

Tooltip

- content
- gap
- hide-delay
- is-content-html
- show-delay
- threshold

LineMarker class

- position-changed
- alignment
- content
- drag-content
- drag-lines
- drag-threshold
- horizontal-position
- interaction
- is-visible
- lines
- series-index
- vertical-position

RangeSelector class

- on-client-range-changed
- is-visible
- max
- min
- orientation

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

- client-converter
- property-name

SortDescriptions

- ascending
- property

FlexGrid

The **FlexGrid** control provides a powerful and flexible way to display data from a datasource in tabular format. The **FlexGrid** control is a full-featured grid, providing various features including several selection modes, sorting, column reordering, grouping, filtering, editing, custom cells, XAML-style star-sizing columns, row and column virtualization, etc.

FlexGrid provides design flexibility with unbound columns, conditional formatting and cell level customization. This allows developers to create complex grid-based applications, and edit and update data bound to database.

	ID	Country	Product	Trends	Rank
	1	China	Gadget		★★★☆☆
	2	US	Widget		★☆☆☆☆
	3	China	Widget		★★★★☆
	4	France	Gadget		★★★★☆
	5	France	Gadget		★★★★★
	6	UK	Widget		★☆☆☆☆
	7	UK	Widget		★★★★☆
	8	Japan	Gadget		★★★★☆
	9	Canada	Gadget		★☆☆☆☆
	10	Korea	Gadget		★★★★☆
	11	Japan	Gadget		★☆☆☆☆
	12	Italy	Gadget		★★★★☆
	13	German	Gadget		★★★★★
	14	Italy	Widget		★★★★★
	15	Canada	Gadget		★☆☆☆☆

Key Features

- **AllowMerging:** Set the AllowMerging to true to enable merging of cells that contain the same content.
- **PageSize:** Set the value of PageSize property to the number of items you want on each page.
- **ChildItemsPath:** Set the value of ChildItemsPath property to represent grid in a tree view.
- **DisableServerRead:** Set the value of DisableServerRead property to True to disable scrolling of grid at server side.
- **HeadersVisibility:** Set the value of HeadersVisibility to None, All, Column or Row to specify the visibility of header in the grid.
- **ImeEnabled:** Set the value of ImeEnabled to enable the grid to support InputMethodEditors (IME). This property is valid for sites/applications in Japanese, Chinese, or any other languages that require IME support.

Quick Start: Add data to FlexGrid

This section describes how to add a FlexGrid control to your MVC application and populate data in it. The below example demonstrates local model binding in FlexGrid control. For detailed explanation on how to do remote binding in FlexGrid, see Remote Data Binding topic.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for FlexGrid**
- **Step 3: Add a FlexGrid control**
- **Step 4: Build and Run the Project**

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

	ID	Start	Product	Amount	Discount	Active
	1	1/25/2015	Gadget	$1,341.85	8 %	☐
	2	2/25/2015	Widget	$3,596.33	18 %	☐
	3	3/25/2015	Widget	$3,232.11	24 %	☐
	4	4/25/2015	Gadget	($2,008.99)	9 %	☐
	5	5/25/2015	Gadget	$2,568.01	3 %	☐
	6	6/25/2015	Widget	($3,476.95)	1 %	☐
	7	7/25/2015	Widget	$2,290.56	6 %	☐
	8	8/25/2015	Gadget	($4,146.76)	1 %	☐
	9	9/25/2015	Gadget	$4,917.55	14 %	☐
	10	10/25/2015	Gadget	$3,824.28	8 %	☐
	11	11/25/2015	Gadget	($4,257.83)	23 %	☐
	12	12/25/2015	Gadget	$1,095.08	20 %	☐
	13	1/25/2015	Gadget	$1,853.66	18 %	☐
	14	2/25/2015	Widget	$3,708.86	3 %	☐
	15	3/25/2015	Gadget	($3,447.73)	10 %	☐
*						☐

Back to Top

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for FlexGrid

1. Add a new class to the folder **Models** (for example: `Sale.cs`). See Adding controls to know how to add a new model.
2. Replace the following code in the new model to define the classes that serve as a datasource for the FlexGrid control.

Sale.cs copyCode

```
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; }

        /// <summary>
        /// Get the data.
        /// </summary>
        /// <param name="total"></param>
        /// <returns></returns>
        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;
        }
    }

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

```
    }
}
```

Back to Top

Step 3: Add a FlexGrid control

Complete the following steps to initialize a FlexGrid control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C# copyCode

```
using System.Collections;
using System.Globalization;
using System.Linq;
using System.Web.Mvc;
using C1.Web.Mvc;
using MVCFlexGrid.Models;
using System.Collections.Generic;
using System;
```

Note: Replace **MVCFlexGrid.Models;** with ***<YourMVCApplicationName>*.Models;** in the references.

5. Replace the method Index() with the following method.

IndexController.cs

C#

C# copyCode

```
public ActionResult Index()
{
    return View(Sale.GetData(15));
}
```

VB

VB

```
Public Function Index() As ActionResult
        Return View(Sale.GetData(15))
End Function
```

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view has been added for the controller.
6. In the Solution Explorer, double click `Index.cshtml` to open it.
7. Replace the default code of the **Views\Index.cshtml** file with the one given below to initialize a **FlexGrid** control.

HTML Helpers

Index.cshtml copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid

@model IEnumerable<Sale>

 // Instantiate FlexGrid and set its properties

@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .Height(450)
    .Width(700)
    .AllowAddNew(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .CssClass("grid")
    .Bind(Model)

 // Binding columns data to FlexGrid

     .Columns(bl =>
       {
            bl.Add(cb => cb.Binding("ID"));
            bl.Add(cb => cb.Binding("Start"));
            bl.Add(cb => cb.Binding("Product"));
            bl.Add(cb => cb.Binding("Amount").Format("c"));
            bl.Add(cb => cb.Binding("Discount").Format("p0"));
            bl.Add(cb => cb.Binding("Active"));
        })
)
```

Index.vbhtml

```
@Imports C1.Web.Mvc
@Imports C1.Web.Mvc.Fluent
@Imports C1.Web.Mvc.Grid
@ModelType IEnumerable(Of Sale)

@(Html.C1().FlexGrid(Of Sale)() _
.CssClass("grid") _
```

```
.Bind(Model) _
.AutoGenerateColumns(False) _
.Width(700) _
.Height("800px") _
.SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell) _
.Columns(Sub(bl)
                bl.Add(Sub(cb) cb.Binding("ProductName"))
                bl.Add(Sub(cb) cb.Binding("SupplierID"))
                bl.Add(Sub(cb) cb.Binding("QuantityPerUnit"))
                bl.Add(Sub(cb) cb.Binding("UnitPrice"))
                bl.Add(Sub(cb) cb.Binding("UnitsInStock"))
                bl.Add(Sub(cb) cb.Binding("UnitsOnOrder"))
                bl.Add(Sub(cb) cb.Binding("Discontinued"))
                End Sub) _)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

<c1-flex-grid auto-generate-columns="false" height="500px" width="800px"
class="grid"
               is-read-only="true" allow-add-new="true"
              allow-sorting="true"
              selection-mode="@((SelectionMode.Cell))" >
    <c1-items-source read-action-url="@Url.Action("Index_Bind")"></c1-items-
source>
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" format="p0"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>

</c1-flex-grid>
```

Step 4: Build and Run the Project

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

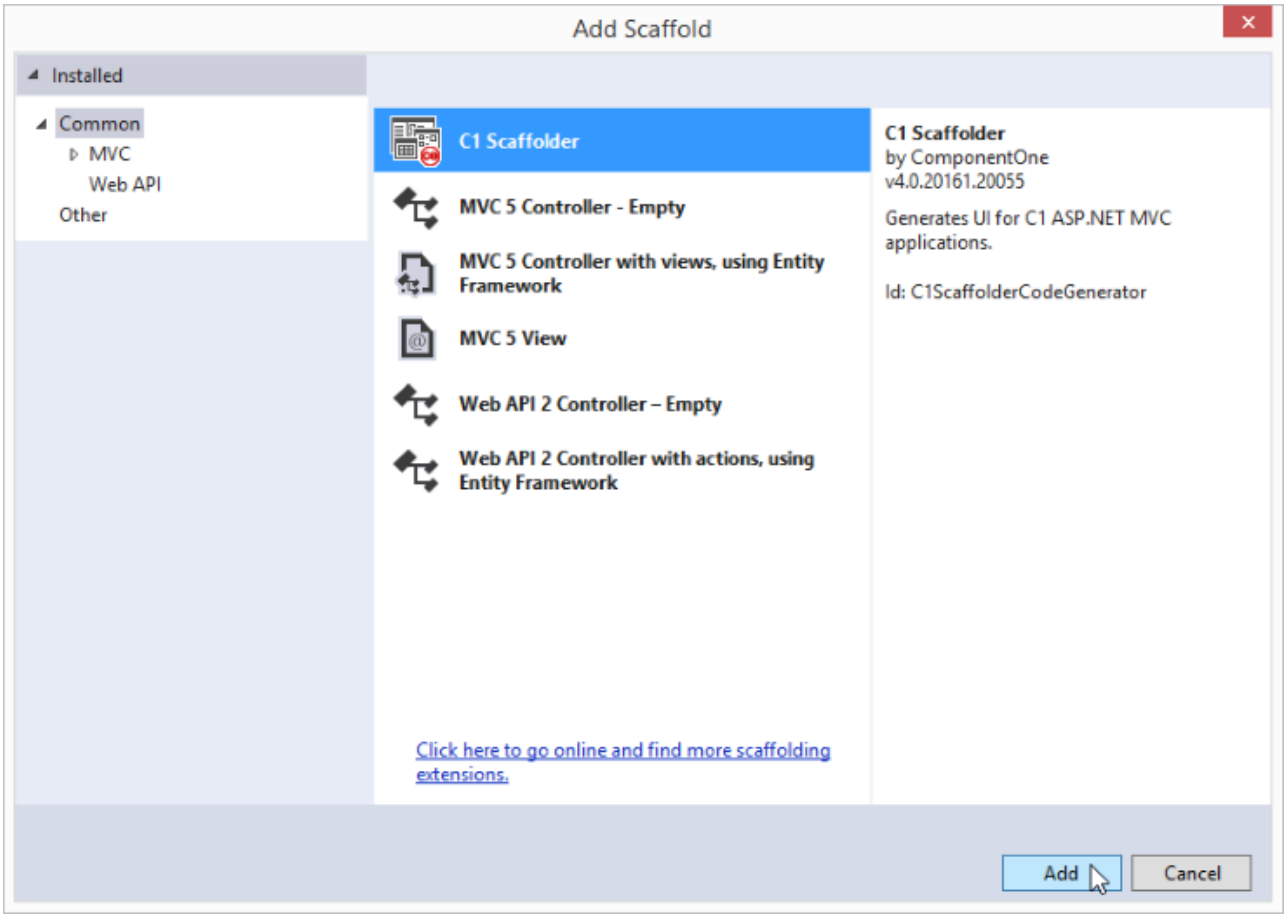
Back to Top

Features

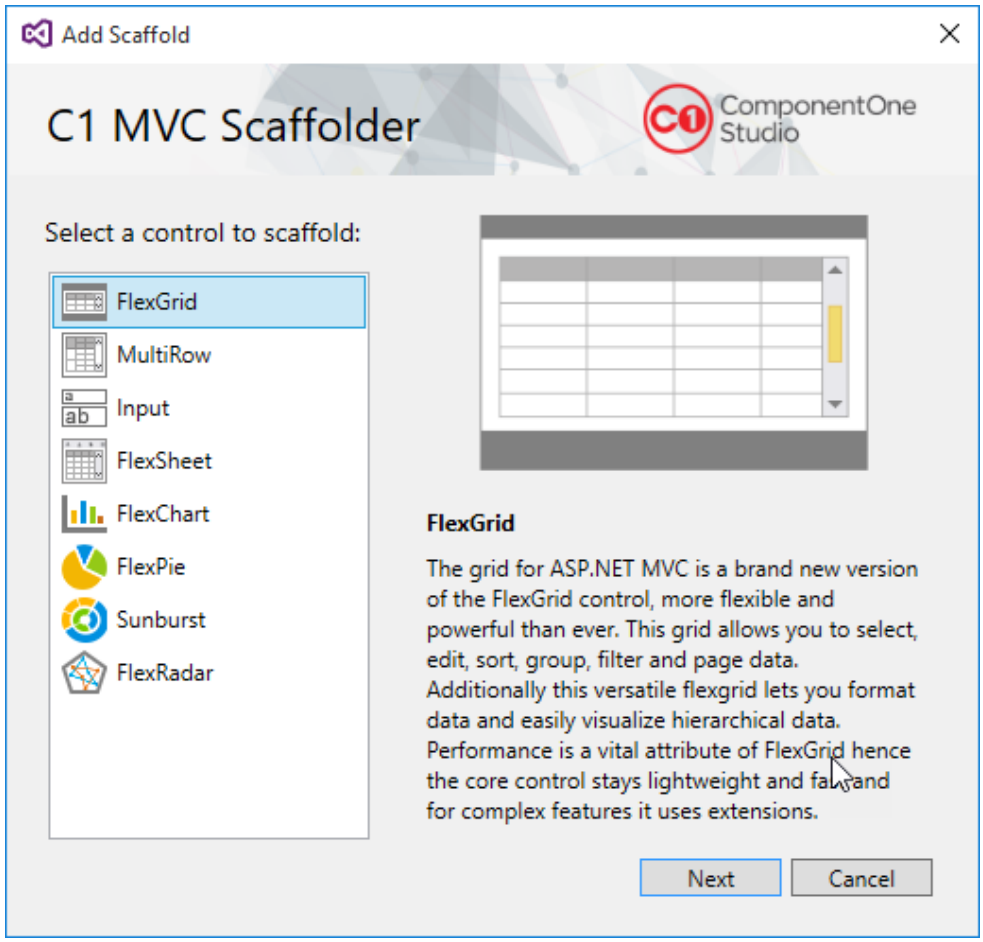
Scaffolding

The steps to scaffold **ComponentOne FlexGrid** control for ASP.NET MVC are as follows:

1. Configure the datasource. Refer the topic Data Source Configuration to see configuring a datasource in an application.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane. You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.



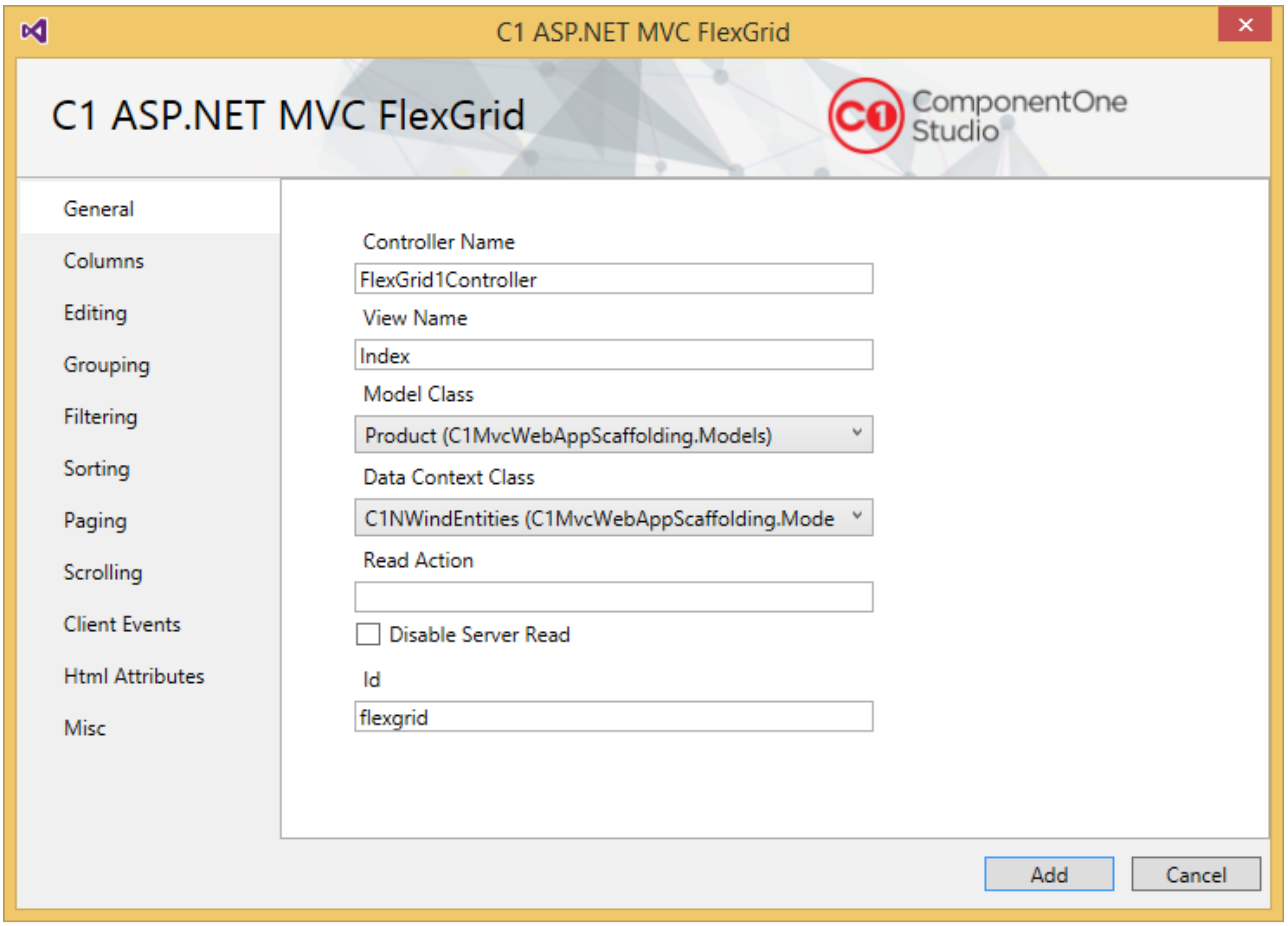
4. Click Add.
5. Select FlexGrid control and click Next.



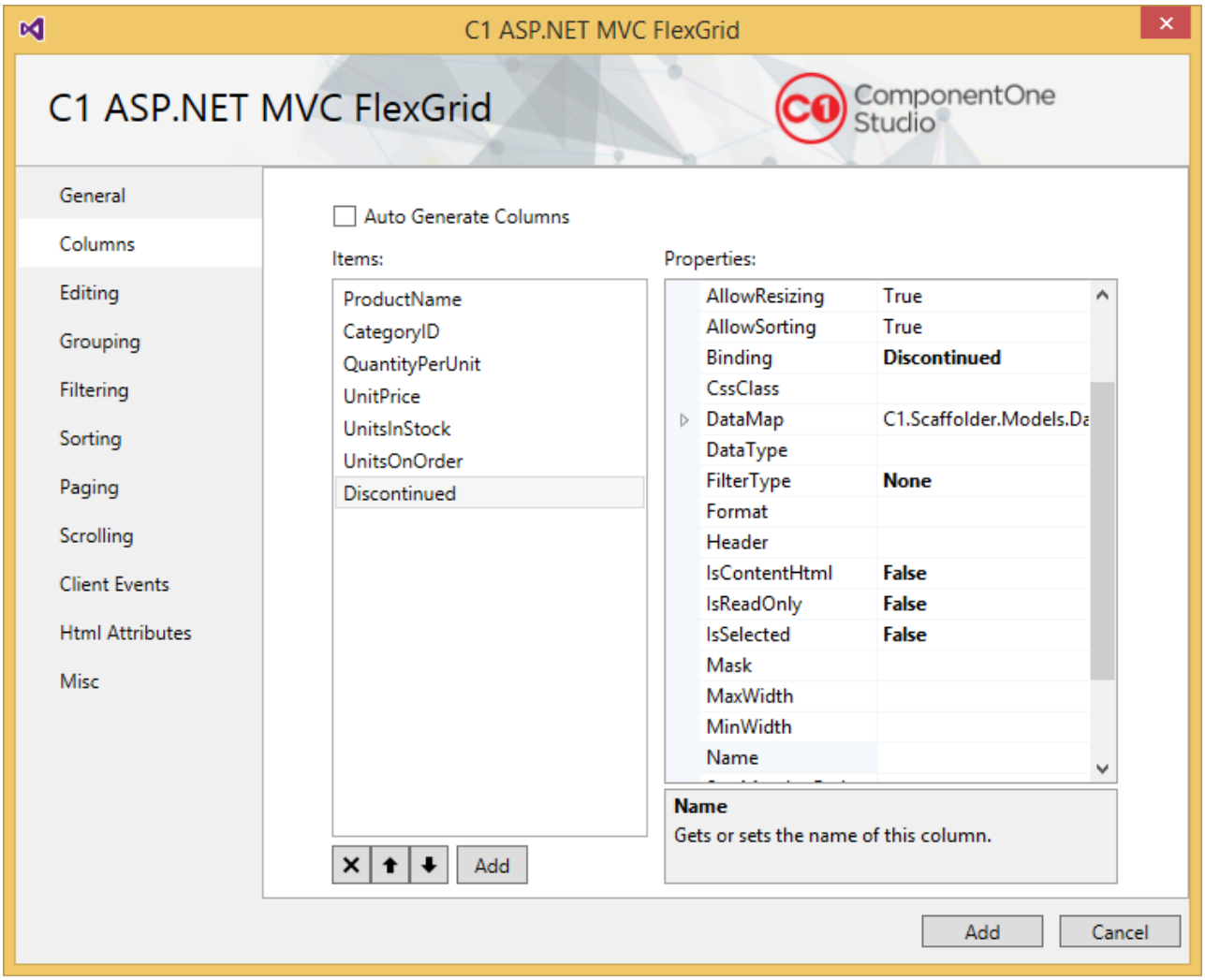
The **C1 ASP.NET MVC FlexGrid** wizard appears with the General tab selected by default.

6. In the **General** tab, specify the model as follows:

    1. Fill in the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in Step 1. In our case, we select Product to populate products in the FlexGrid.
    3. Select **Data Context Class** from the drop-down list. In our case, we select C1NWindEntities.



7. Go to **Columns** tab to specify the columns in the FlexGrid control. By default **Auto Generate Columns** is checked; if not, then you can add, delete, or move columns upward or downward in the sequence in which they should appear in the final view. In our case, we have selected columns as shown in the following image:



8. Go to **Editing** tab and check **Allow Edit** and **Allow Delete** check boxes.
9. Go to **Grouping** tab. In the **Group Settings**, check **Show Groups** and CategoryID check boxes from Group Descriptions, and give Group Header Format a name, say 'Group by Category ID'.
10. Go to **Filtering** tab and check **Allow Filtering** check box. Let the other settings be same as default.
11. Go to **Sorting** tab and let the setting be same as default - both **Allow Sorting** and **Show Sort** check boxes should be checked.
12. Go to **Client Events** and check **BeginningEdit** check box.
13. Click **Add**. You will notice that the Controller and View for the selected model is added to your project. The codes generated for the Controller and View are as follows:

FlexGrid1Controller.cs

FlexGrid1Controller.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Validation;
```

```
 using System.Web;
 using System.Web.Mvc;

// This code was generated by C1 Scaffolder.

namespace C1MvcWebAppScaffolding.Controllers
 {
     public partial class FlexGridController : Controller
     {
         private C1MvcWebAppScaffolding.Models.C1NWindEntities db = new
C1MvcWebAppScaffolding.Models.C1NWindEntities();
         public ActionResult Index()
         {
             var model = db.Products;
             return View(model);
         }

        public ActionResult FlexGrid_Update([C1JsonRequest]CollectionViewEditRequest
requestData)
         {
             return Update(requestData, db.Products);
         }

        private ActionResult Update(CollectionViewEditRequest requestData, DbSet
data) where T : class
         {
             return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
             {
                 string error = string.Empty;
                 bool success = true;
                 try
                 {
                     db.Entry(item as object).State = EntityState.Modified;
                     db.SaveChanges();
                 }
                 catch (DbEntityValidationException e)
                 {
                     error = string.Join(",", e.EntityValidationErrors.Select(result
=>
                     {
                         return string.Join(",", result.ValidationErrors.Select(err
=> err.ErrorMessage));
                     }));
                     success = false;
                 }
                 catch (Exception e)
                 {
                     error = e.Message;
                     success = false;
                 }
                 return new CollectionViewItemResult
                 {
                     Error = error,
                     Success = success && ModelState.IsValid,
                     Data = item
```

```
                };
            }, () => data.ToList()));
        }

        public ActionResult FlexGrid_Delete([C1JsonRequest]CollectionViewEditRequest
requestData)
        {
            return Delete(requestData, db.Products, item => new object[] {
item.ProductID });
        }

        private ActionResult Delete(CollectionViewEditRequest requestData, DbSet
data, Func getKeys) where T : class
        {
            return this.C1Json(CollectionViewHelper.Edit(requestData, item =>
            {
                string error = string.Empty;
                bool success = true;
                try
                {
                    var resultItem = data.Find(getKeys(item));
                    data.Remove(resultItem);
                    db.SaveChanges();
                }
                catch (DbEntityValidationException e)
                {
                    error = string.Join(",", e.EntityValidationErrors.Select(result
=>
                    {
                        return string.Join(",", result.ValidationErrors.Select(err
=> err.ErrorMessage));
                    }));
                    success = false;
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }
                return new CollectionViewItemResult
                {
                    Error = error,
                    Success = success && ModelState.IsValid,
                    Data = item
                };
            }, () => data.ToList()));
        }

    }
}
```

Index

HTML Helpers

Razor (Index.cshtml)

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@using C1.Web.Mvc.Grid
@model IEnumerable<C1MvcWebAppScaffolding.Models.Product>
    <script type="text/javascript">
        function beginningEdit(sender, e) {
            // Implement the event handler for beginningEdit.
        }
    </script>
@(Html.C1().FlexGrid<C1MvcWebAppScaffolding.Models.Product>()
    .Id("flexgrid")
    .Bind(bl => { bl.Bind(Model)
        .Update(Url.Action("FlexGrid_Update"))
        .Delete(Url.Action("FlexGrid_Delete"))
        .GroupBy("CategoryID")
    ;})
    .AutoGenerateColumns(false)
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ProductName"));
        bl.Add(cb => cb.Binding("CategoryID"));
        bl.Add(cb => cb.Binding("QuantityPerUnit"));
        bl.Add(cb => cb.Binding("UnitPrice"));
        bl.Add(cb => cb.Binding("UnitsInStock"));
        bl.Add(cb => cb.Binding("UnitsOnOrder"));
        bl.Add(cb => cb.Binding("Discontinued"));
    })
    .AllowDelete(true)
    .GroupHeaderFormat("Group by Category ID")
    .Filterable(f => { f
        .DefaultFilterType(FilterType.Both)
    ;})
    .OnClientBeginningEdit("beginningEdit")
)
```

Tag Helpers

HTML

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Grid
@addTagHelper *, C1.AspNetCore.Mvc
@model IEnumerable<C1MvcApplication1.Models.Product>
    <script type="text/javascript">
        function beginningEdit(sender, e) {
            // Implement the event handler for beginningEdit.
        }
    </script>
<c1-flex-grid id="flexgrid" auto-generate-columns="false" allow-delete="true" group-
header-format="Group by Category ID"
```

```
            beginning-edit="beginningEdit">
    <c1-items-source source-collection="Model" update-action-
url="@(Url.Action("FlexGrid_Update"))"
                     delete-action-url="@(Url.Action("FlexGrid_Delete"))"
group-by="CategoryID"></c1-items-source>
    <c1-flex-grid-column binding="ProductName"></c1-flex-grid-column>
    <c1-flex-grid-column binding="CategoryID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="QuantityPerUnit"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitPrice"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitsInStock"></c1-flex-grid-column>
    <c1-flex-grid-column binding="UnitsOnOrder"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discontinued"></c1-flex-grid-column>
    <c1-flex-grid-filter default-filter-type="FilterType.Both"></c1-flex-grid-filter>
</c1-flex-grid>
```

14. Run the project.

	ProductNa...	CategoryID	QuantityP...	UnitPrice	UnitsInSto...	UnitsOnOr...	Discontin...
	Group by Category ID						
	Chai	1	10 boxes x 20...	18	39	0	☑
	Guaraná Fa...	1	12 - 355 ml c...	4.50	20	0	☑
	Sasquatch Ale	1	24 - 12 oz bot...	14	111	0	☐
	Steeleye Stout	1	24 - 12 oz bot...	18	20	0	☐
	Côte de Blaye	1	12 - 75 cl bottles	263.50	17	0	☐
	Chartreuse ...	1	750 cc per bo...	18	69	0	☐
	Ipoh Coffee	1	16 - 500 g tins	46	17	10	☐
	Laughing Lu...	1	24 - 12 oz bot...	14	52	0	☐
	Outback Lager	1	24 - 355 ml b...	15	15	10	☐
	Rhönbräu Kl...	1	24 - 0.5 l bottles	7.75	125	0	☐
	Group by Category ID						
	Uncle Bob's ...	7	12 - 1 lb pkgs.	30	15	0	☑
	Tofu	7	40 - 100 g pkgs.	23.25	35	0	☐
	Rössle Sau...	7	25 - 825 g cans	45.60	26	0	☑
	Manjimup D...	7	50 - 300 g pkgs.	53	20	0	☐
	Longlife Tofu	7	5 kg pkg.	10	4	20	☐
	Group by Category ID						
	Mishi Kobe ...	6	18 - 500 g pkgs.	97	29	0	☑
	Alice Mutton	6	20 - 1 kg tins	39	0	0	☑
	Thüringer R...	6	50 bags x 30 ...	123.79	0	0	☑
	Perth Pasties	6	48 pieces	32.80	0	0	☑

Data Binding

Before you can use FlexGrid and it's features, you must bind data to your Flexgrid. Once you have bind the data to

your FlexGrid, you can use many different features such as filtering, sorting, selection mode, detail row and much more.

This section contains information about

Remote Data Binding
: Learn how to remotely bind data to your FlexGrid using C1JSONRequest.

Model Binding
: Learn how to add data to your FlexGrid using basic Model binding.

AJAX Data Binding
: Learn how to add data to your FlexGrid using basic Model binding.

Master Detail in FlexGrid
: Learn how FlexGrid can be customized to display Master-Detail data

Remote Data Binding

FlexGrid allows you to retrieve data directly using **C1JSONRequest**. This specifies remote data URLs that include the server, table and columns. The arrays returned are used as data sources for **CollectionView** objects.

`CollectionViewHelper` is a static class that enables collections to have editing, filtering, grouping, and sorting services. This class also includes the following methods:

- `Read()`: Retrieves data from the collection.
- `Edit()`: Enables excel-style editing in FlexGrid. See Excel-Style Editing for more information.
- `BatchEdit()`: Allows editing multiple items at a time. See Batch Editing for more information.

The `Bind` property in FlexGrid is used to bind it to a collection by passing an action URL method to carry out a specific operation.

This topic demonstrates how to retrieve data from an existing data source remotely. This is useful for developing data-intensive applications and scenarios for representing data as dashboards. Refer to **Quick Start: Add data to FlexGrid** for explanation of Local Model Binding.

The following image shows how the FlexGrid appears after making the C1JSON Request to fetch data from the model, Sale.cs, which was added to the application in the QuickStart:

	Start	End	Country	Product	Amount	Discount
	Jan 25 15	00:00	France	Widget	$877.93	8 %
	Feb 25 15	01:01	Korea	Widget	($3,788.14)	12 %
	Mar 25 15	02:02	German	Gadget	($2,446.92)	12 %
	Apr 25 15	03:03	German	Widget	($4,374.97)	18 %
	May 25 15	04:04	Japan	Gadget	$1,089.32	16 %
	Jun 25 15	05:05	China	Gadget	$1,341.85	8 %
	Jul 25 15	06:06	US	Widget	$3,596.33	18 %
	Aug 25 15	07:07	China	Widget	$3,232.11	24 %
	Sep 25 15	08:08	France	Gadget	($2,008.99)	9 %
	Oct 25 15	09:09	France	Gadget	$2,568.01	3 %

The following code examples demonstrate how to bind FlexGrid to fetch data from remote datasource:

In Code

RemoteBindController.cs

C# copyCode

```
public partial class FlexGridController : Controller
{
    public ActionResult RemoteBind_Read([C1JsonRequest] CollectionViewRequest<Sales>
requestData)
    {
        return this.C1Json(CollectionViewHelper.Read(requestData,
Sales.GetData(10)));
    }

    public ActionResult RemoteBind()
    {
        return View();
    }
}
```

RemoteBind.cshtml

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Grid
@(Html.C1().FlexGrid()
    .AutoGenerateColumns(false)
    .AllowSorting(true)
    .Bind(Url.Action("RemoteBind_Read"))
    .CssClass("grid")
    .SelectionMode(SelectionMode.Row)
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("ID").Visible(false));
        columns.Add(column => column.Binding("Start").Format("MMM d yy"));
        columns.Add(column => column.Binding("End").Format("HH:mm"));
        columns.Add(column => column.Binding("Country"));
        columns.Add(column => column.Binding("Product"));
        columns.Add(column => column.Binding("Amount").Format("c"));
        columns.Add(column => column.Binding("Discount").Format("p0"));
    })
)
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Grid
```

```
<c1-flex-grid auto-generate-columns="false" allow-sorting="true" is-read-only="true"
              class="grid" selection-mode="SelectionMode.Row">
    <c1-items-source read-action-url="@Url.Action("RemoteBind_Read")"></c1-items-
source>
    <c1-flex-grid-column binding="ID" is-visible="false" ></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start" format="MMM d yy"></c1-flex-grid-column>
    <c1-flex-grid-column binding="End" format="HH:mm"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" format="p0"></c1-flex-grid-column>
</c1-flex-grid>
```

Model Binding

This topic describes how to add a FlexGrid control to your MVC application and populate data in it. The below example demonstrates local model binding in FlexGrid control. Refer to Remote Data Binding for detailed explanation on how to do remote binding in FlexGrid.

This topic comprises of three steps:

- **Step 1: Create a Datasource for FlexGrid**
- **Step 2: Add a FlexGrid control**
- **Step 3: Build and Run the Project**

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

	ID	Start	Product	Amount	Discount	Active
	1	1/25/2015	Gadget	$1,341.85	8 %	☐
	2	2/25/2015	Widget	$3,596.33	18 %	☐
	3	3/25/2015	Widget	$3,232.11	24 %	☐
	4	4/25/2015	Gadget	($2,008.99)	9 %	☐
	5	5/25/2015	Gadget	$2,568.01	3 %	☐
	6	6/25/2015	Widget	($3,476.95)	1 %	☐
	7	7/25/2015	Widget	$2,290.56	6 %	☐
	8	8/25/2015	Gadget	($4,146.76)	1 %	☐
	9	9/25/2015	Gadget	$4,917.55	14 %	☐
	10	10/25/2015	Gadget	$3,824.28	8 %	☐
	11	11/25/2015	Gadget	($4,257.83)	23 %	☐
	12	12/25/2015	Gadget	$1,095.08	20 %	☐
	13	1/25/2015	Gadget	$1,853.66	18 %	☐
	14	2/25/2015	Widget	$3,708.86	3 %	☐
	15	3/25/2015	Gadget	($3,447.73)	10 %	☐
*						☐

Back to Top

Step 1: Create a Datasource for FlexGrid

1. Add a new class to the folder **Models** (for example: `Sale.cs`). See Adding controls to know how to add a new model.
2. Replace the following code in the new model to define the classes that serve as a datasource for the FlexGrid control.

Sale.cs copyCode

```
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; }
```

```
        /// <summary>
        /// Get the data.
        /// </summary>
        /// <param name="total"></param>
        /// <returns></returns>
        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;
        }
    }

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

Back to Top

Step 2: Add a FlexGrid control

Complete the following steps to initialize a FlexGrid control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

```
C#                                                                copyCode
using System.Collections;
using System.Globalization;
using System.Linq;
using System.Web.Mvc;
using C1.Web.Mvc;
using MVCFlexGrid.Models;
using System.Collections.Generic;
using System;
```

> **Note:** Replace **MVCFlexGrid.Models;** with **<*YourMVCApplicationName*>.Models;** in the references.

5. Replace the method Index() with the following method.

IndexController.cs

```
C#                                                                copyCode
public ActionResult Index()
{
    return View(Sale.GetData(15));
}
```

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view has been added for the controller.
6. In the Solution Explorer, double click `Index.cshtml` to open it.
7. Replace the default code of the **Views\Index.cshtml** file with the one given below to initialize a **FlexGrid** control.

HTML Helpers

```
Razor (Index.cshtml)                                              copyCode
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
```

```
@model IEnumerable<Sale>

 // Instantiate FlexGrid and set its properties

@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .Height(450)
    .Width(700)
    .AllowAddNew(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .CssClass("grid")
    .Bind(Model)

 // Binding columns data to FlexGrid

     .Columns(bl =>
       {
            bl.Add(cb => cb.Binding("ID"));
            bl.Add(cb => cb.Binding("Start"));
            bl.Add(cb => cb.Binding("Product"));
            bl.Add(cb => cb.Binding("Amount").Format("c"));
            bl.Add(cb => cb.Binding("Discount").Format("p0"));
            bl.Add(cb => cb.Binding("Active"));
        })
)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

<c1-flex-grid auto-generate-columns="false" height="500px" width="800px"
class="grid"
               is-read-only="true" allow-add-new="true"
              allow-sorting="true"
              selection-mode="@((SelectionMode.Cell))" >
    <c1-items-source read-action-url="@Url.Action("Index_Bind")"></c1-items-
source>
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" format="p0"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>

</c1-flex-grid>
```

Step 3: Build and Run the Project

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

Back to Top

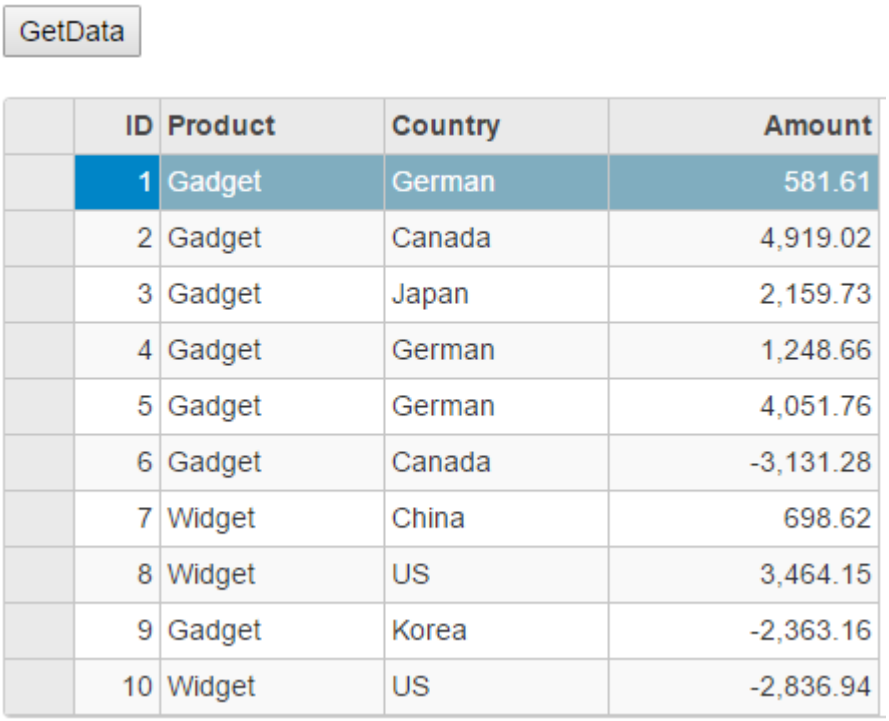
AJAX Data Binding

FlexGrid provide multiple options for data binding. AJAX helps you to load data in the background and display it on the webpage, without reloading the complete webpage. It provides a smoother user experience while you are working with paging, sorting or filtering features in FlexGrid. This topic describes how to bind a FlexGrid at client side using an AJAX call.

This topic comprises the following steps:

- **Step 1: Create a Model**
- **Step 2: Add a Controller Action**
- **Step 3: Add FlexGrid to View**
- **Step 4: Make an AJAX call using JavaScript**
- **Step 5: Build and Run the Project**

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



Back to Top

Step 1: Create a Model

1. Add a new class to the folder **Models** (For example: `Sale.cs`. For more information on how to add a new model, see Adding Controls.
2. Replace the following code in the `Sale.cs` model. We are using **Sale** class to represent sales order data in the database. Each instance of **Sale** object will correspond to a record in the FlexGrid control.

Sale.cs copyCode

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
namespace AjaxDataBinding.Models
{
    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; }

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

        /// <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 startDate = new DateTime(dt.Year, i % 12 + 1, 25);
                var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i %
60, i % 60);

                return new Sale
                {
                    ID = i + 1,
                    Start = startDate,
                    End = endDate,
                    Country = country,
                    Product = product,
                    Color = color,
```

```
                    Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Discount = Math.Round(rand.NextDouble() / 4, 2),
                    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;
        }
        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;
        }
    }
    public class MonthData
    {
        public int Month { get; set; }
        public double Data { get; set; }
    }
}
```

Back to Top

Step 2: Add a Controller Action

1. Open the `HomeController.cs` from the Controllers folder.
2. Add the following code to return JSON data.

C#

```
[HttpPost]
        public JsonResult GetData()
        {
            List<Sale> saleList = Sale.GetData(10).ToList<Sale>();
            return Json(saleList);
        }
```

Back to Top

Step 3: Add FlexGrid to View

1. Open the `Index.html` from View/Home folder.
2. Replace the content of this file with the following code to declare a FlexGrid control. The code below declares different columns and properties of the FlexGrid, but does not bind data to the control.

HTML Helpers

Razor

```
@(Html.C1().FlexGrid()
        .Id("fg")
        .AutoGenerateColumns(false)
        .IsReadOnly(false)
        .AutoClipboard(true)
        .AllowSorting(true)
        .AllowAddNew(false)
        .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
        .Height(500)
        .Columns(bl =>
            {
                bl.Add(cb => cb.Binding("ID").Width("0.4*").IsReadOnly(true));
                bl.Add(cb =>
cb.Binding("Country").Header("Country").Width("*").Name("Country"));
                bl.Add(cb =>
cb.Binding("Amount").Header("Amount").Width("*").Name("Amount"));
                bl.Add(cb =>
cb.Binding("Product").Header("Product").Width("*").Name("Product"));
             }))
```

Tag Helpers

HTML

```
<c1-flex-grid id="fg" height="500px" is-read-only="true" auto-clipboard="true"
              auto-generate-columns="false" allow-add-new="false"
              allow-sorting="true" selection-mode="@((SelectionMode.Cell))">
<c1-flex-grid-column name="ID" width="0.4*"></c1-flex-grid-column>
<c1-flex-grid-column name="Product" header="Product" width="*"></c1-flex-grid-
column>
<c1-flex-grid-column name="Country" header="Country" width="*"></c1-flex-grid-
column>
<c1-flex-grid-column name="Amount" header="Amount" width="*"></c1-flex-grid-
column>
</c1-flex-grid>
```

Back to Top

Step 4: Make an AJAX call using JavaScript

We'll bind data to the control on the client side using JavaScript to make an AJAX call to the Action created in the **HomeController.cs** file. When we bind the grid at client-side in the Load function below, instead of assigning the result data of AJAX call to FlexGrid **itemSource** property, we should update the sourceCollection of FlexGrid's **collectionView** at client-side to retain the server-side features of collectionView. We have also added a code check for any errors in date values of JSON data.

1. In the **Solution Explorer**, click the **Views** folder.

2. Open the `Index.html` file inside the **Home** folder and copy the following JavaScript code.

Index.html

```
<script type="text/javascript">
   function parseDate(strDate) {
        var date = strDate.match(/\d+/g);
        return new Date(parseInt(date));
    }

    function Load() {
        $.ajax({
            type: "POST",
            url: "/Home/GetData",
            dataType: "json",
            success: function (result) {
                var flex = wijmo.Control.getControl("#fg"),
                    cv = flex.collectionView;
                //flex.itemsSource = result;
                try {
                    cv._isFillingData = true;
                    cv.deferUpdate(function () {
                        cv.sourceCollection.clear();
                        result.forEach(function (item) {
                            item.Start = parseDate(item.Start);
                            item.End = parseDate(item.End);
                            cv.sourceCollection.push(item);
                        });
                    })
                } finally {
                    cv._isFillingData = false;
                }
            },
            error: function (err) {

            }
        });
    }
</script>
```

3. Add the following code to call the above script on a button click.

Index.html

```
<input type="button" id="btload" value="Get Data" onclick="Load()" />
```

Back to Top

Step 5: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project and then, click **Get Data** button to load and display the data in the FlexGrid.

Back to Top

Master Detail in FlexGrid

The **FlexGrid** can be customized to display Master-Detail data. This interface shows data in two grids. The grid with master data shows a collection of objects, while detail grid displays details specific to the object selected by the user in master grid. Changing the selection in master grid will update the detail grid accordingly.

This topic demonstrates how to display master detail data using two related data tables. The following example uses Entity Framework model to bind FlexGrid with Customers and Orders data from local NorthWind database file **C1NWind.mdf**.

The following image shows a FlexGrid control displaying data in Master-Detail relationship. The master grid displays summarised Customers data in four columns. However, the detail grid details the information of orders placed by a particular customer selected in the master grid. The two tables are related through CustomerID column.

Customers:

CustomerID	CompanyName	Country	City
ALFKI	Alfreds Futterkiste	Germany	Berlin
ANATR	Ana Trujillo Emparedados y helados	Mexico	México D.F.
ANTON	Antonio Moreno Taquería	Mexico	México D.F.
AROUT	Around the Horn	UK	London
BERGS	Berglunds snabbköp	Sweden	Luleå
BLAUS	Blauer See Delikatessen	Germany	Mannheim
BLONP	Blondel père et fils	France	Strasbourg
BOLID	Bólido Comidas preparadas	Spain	Madrid
BONAP	Bon app'	France	Marseille
BOTTM	Bottom-Dollar Markets	Canada	Tsawassen

All orders of the selected Customer:

OrderID	EmployeeID	OrderDate	ShippedDate	RequiredDate	Freight	ShipVia	ShipName	ShipCountry	ShipCity
10,643	6	9/25/1995	10/3/1995	10/23/1995	29.46	1	Alfreds Futter…	Germany	Berlin
10,692	4	11/3/1995	11/13/1995	12/1/1995	61.02	2	Alfred's Futter…	Germany	Berlin
10,702	4	11/13/1995	11/21/1995	12/25/1995	23.94	1	Alfred's Futter…	Germany	Berlin
10,835	1	2/15/1996	2/21/1996	3/14/1996	69.53	3	Alfred's Futter…	Germany	Berlin
10,952	1	4/15/1996	4/23/1996	5/27/1996	40.42	1	Alfred's Futter…	Germany	Berlin
11,011	3	5/9/1996	5/13/1996	6/6/1996	1.21	1	Alfred's Futter…	Germany	Berlin

The following code examples demonstrate how to display Master-Detail data in a FlexGrid control.

In Code

MasterDetailController.cs

C# copyCode

```
public class MasterDetailController : Controller
{
    private C1NWindEntities db = new C1NWindEntities();
    public ActionResult MasterDet()
    {
        return View(db.Customers.Take(10).ToList());
    }
     public ActionResult DetailData([C1JsonRequest] CollectionViewRequest<Order> requestData)
    {
        string customerID = requestData.ExtraRequestData["CustomerID"].ToString();
        return this.C1Json(CollectionViewHelper.Read(requestData, db.Orders.Where(s => s.CustomerID ==
customerID).ToList()));
    }
}
```

MasterDetail.cshtml

Initialize the FlexGrid to display Customer data for master interface.

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Grid
@using MVCFlexGrid.Models
@model IEnumerable<Customer>

<h4>Customers:</h4>
```

```
@(Html.C1().FlexGrid().Id("Customer").AutoGenerateColumns(false).IsReadOnly(true).SelectionMode(SelectionMode.Row)
.Bind(Model)
.Columns(cs =>
    cs.Add(c => c.Binding("CustomerID").Width("*"))
    .Add(c => c.Binding("CompanyName").Width("2*"))
    .Add(c => c.Binding("Country").Width("2*"))
    .Add(c => c.Binding("City").Width("2*"))
))
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Grid
@using MVCFlexGrid.Models
@model IEnumerable<Customer>

<h4>Customers:</h4>
<c1-flex-grid id="Customer" auto-generate-columns="false" is-read-only="true" selection-mode="SelectionMode.Row">
    <c1-flex-grid-column binding="CustomerID" width="*" />
    <c1-flex-grid-column binding="CompanyName" width="2*" />
    <c1-flex-grid-column binding="Country" width="2*" />
    <c1-flex-grid-column binding="City" width="2*" />
    <c1-items-source source-collection="Model" />
</c1-flex-grid>
```

Initialize the FlexGrid to display Order data for detail interface.

HTML Helpers

Razor copyCode

```
@(Html.C1().FlexGrid().Id("Orders").AutoGenerateColumns(false).IsReadOnly(true)
.Bind(i => i.Bind(Url.Action("DetailData")).OnClientQueryData("getCustomer"))
.Columns(cs =>
    cs.Add(c => c.Binding("OrderID"))
    .Add(c => c.Binding("EmployeeID"))
    .Add(c => c.Binding("OrderDate"))
    .Add(c => c.Binding("ShippedDate"))
    .Add(c => c.Binding("RequiredDate"))
    .Add(c => c.Binding("Freight"))
    .Add(c => c.Binding("ShipVia"))
    .Add(c => c.Binding("ShipName"))
    .Add(c => c.Binding("ShipCountry"))
    .Add(c => c.Binding("ShipCity"))
))
```

Tag Helpers

HTML copyCode

```
<h4>All orders of the selected Customer:</h4>
<c1-flex-grid id="Orders" auto-generate-columns="false" is-read-only="true">
    <c1-flex-grid-column binding="OrderID" />
    <c1-flex-grid-column binding="EmployeeID" />
    <c1-flex-grid-column binding="OrderDate" />
    <c1-flex-grid-column binding="ShippedDate" />
    <c1-flex-grid-column binding="RequiredDate" />
    <c1-flex-grid-column binding="Freight" />
    <c1-flex-grid-column binding="ShipVia" />
    <c1-flex-grid-column binding="ShipName" />
    <c1-flex-grid-column binding="ShipCountry" />
    <c1-flex-grid-column binding="ShipCity" />
    <c1-items-source read-action-url="@Url.Action("DetailData")" query-data="getCustomer" />
</c1-flex-grid>
```

Javascript to display order data corresponding to the selected CustomerID in the master FlexGrid.

JavaScript copyCode

```
<script type="text/javascript">
    var grid, cv, detail, cvDetail, currentItem;
    c1.mvc.Utils.documentReady(function () {
        grid = wijmo.Control.getControl("#Customer");
        cv = grid.collectionView;
        cv.currentChanged.addHandler(getOrders);
        cv.moveCurrentToFirst();
    });

    function getOrders() {
        detail = wijmo.Control.getControl("#Orders");
        cvDetail = detail.collectionView;
        cvDetail.refresh();
    }

    function getCustomer(sender, e) {
        if (e.extraRequestData == null) {
            e.extraRequestData = {};
        }

        grid = wijmo.Control.getControl("#Customer");
        if (grid) {
            currentItem = grid.collectionView.currentItem;
            e.extraRequestData["CustomerID"] = currentItem ? currentItem.CustomerID : "";
        }
    }
</script>
```

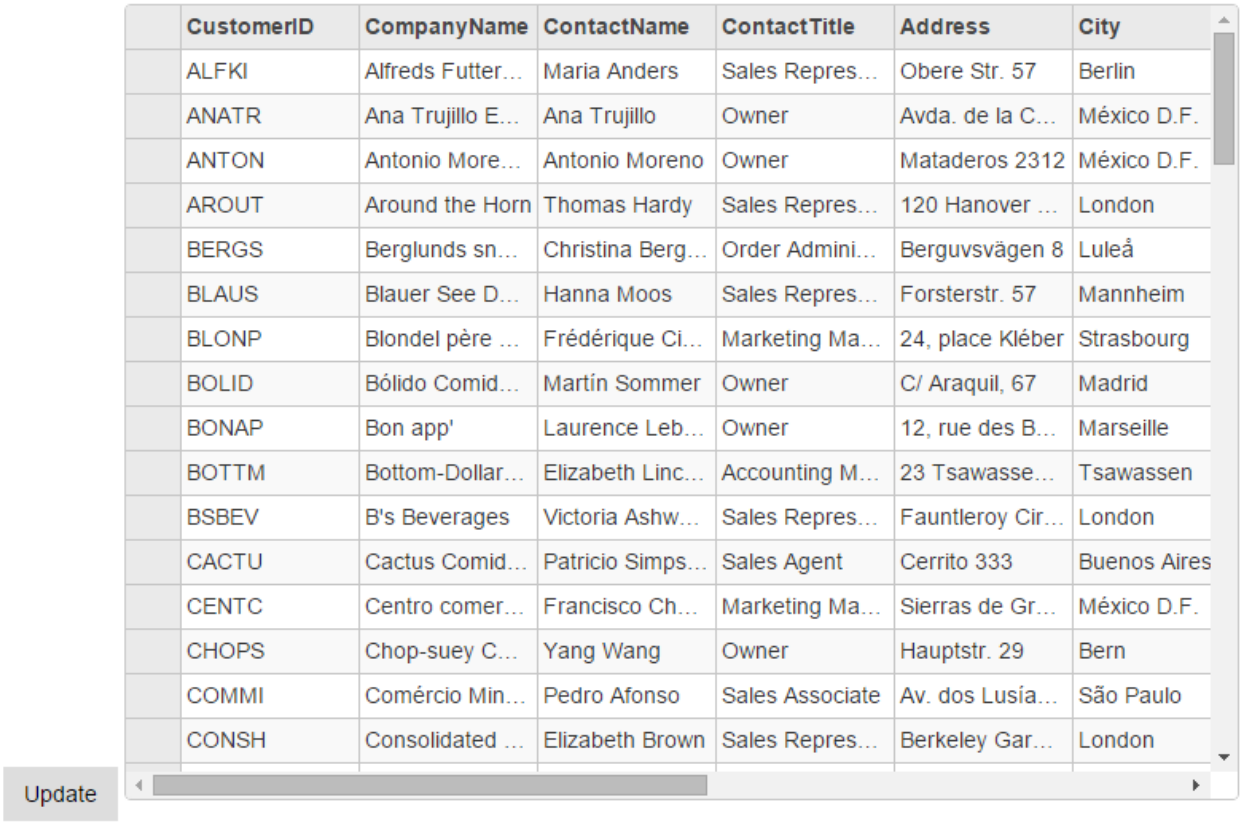
Editing

Batch Editing

FlexGrid provides support for Batch Editing, which lets a user update, create or remove multiple items, and commit it to the data source. With this, the user can perform multiple modifications on grid using sorting, paging or filtering. To enter the FlexGrid in the **BatchEditing** mode, a user needs to provide the BatchEditing action url.

Note: To disable data update while performing sorting, filtering or page operations, set the DisableServerRead property of FlexGrid ItemsSource to True. Once you set the property, data will only be submitted when you call collectionView commit method explicitly from client-side.

The following image shows how the FlexGrid appears after setting the **BatchEditing action url** property. The example provides two ways to commit data to the server, one by clicking Enter key after updating the content of a cell, and other by clicking the Update button.



CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futter...	Maria Anders	Sales Repres...	Obere Str. 57	Berlin
ANATR	Ana Trujillo E...	Ana Trujillo	Owner	Avda. de la C...	México D.F.
ANTON	Antonio More...	Antonio Moreno	Owner	Mataderos 2312	México D.F.
AROUT	Around the Horn	Thomas Hardy	Sales Repres...	120 Hanover ...	London
BERGS	Berglunds sn...	Christina Berg...	Order Admini...	Berguvsvägen 8	Luleå
BLAUS	Blauer See D...	Hanna Moos	Sales Repres...	Forsterstr. 57	Mannheim
BLONP	Blondel père ...	Frédérique Ci...	Marketing Ma...	24, place Kléber	Strasbourg
BOLID	Bólido Comid...	Martín Sommer	Owner	C/ Araquil, 67	Madrid
BONAP	Bon app'	Laurence Leb...	Owner	12, rue des B...	Marseille
BOTTM	Bottom-Dollar...	Elizabeth Linc...	Accounting M...	23 Tsawasse...	Tsawassen
BSBEV	B's Beverages	Victoria Ashw...	Sales Repres...	Fauntleroy Cir...	London
CACTU	Cactus Comid...	Patricio Simps...	Sales Agent	Cerrito 333	Buenos Aires
CENTC	Centro comer...	Francisco Ch...	Marketing Ma...	Sierras de Gr...	México D.F.
CHOPS	Chop-suey C...	Yang Wang	Owner	Hauptstr. 29	Bern
COMMI	Comércio Min...	Pedro Afonso	Sales Associate	Av. dos Lusía...	São Paulo
CONSH	Consolidated ...	Elizabeth Brown	Sales Repres...	Berkeley Gar...	London

Update

The following code examples demonstrate how to enable BatchEditing in the FlexGrid:

In Code

BatchEditingController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();
public ActionResult BatchEditing(CollectionViewRequest<Customer> requestData)
{
    return View(db.Customers.ToList());
}

public ActionResult
GridBatchEdit([C1JsonRequest]CollectionViewBatchEditRequest<Customer> requestData)
{
    return this.C1Json(CollectionViewHelper.BatchEdit<Customer>(requestData,
batchData =>
    {
        IList<CollectionViewItemResult<Customer>> itemresults = new
List<CollectionViewItemResult<Customer>>();
        string error = string.Empty;
        bool success = true;
        try
        {
            if (batchData.ItemsCreated != null)
            {
                batchData.ItemsCreated.ToList<Customer>().ForEach(st =>
```

```
            {
                db.Customers.Add(st);
                itemresults.Add(new CollectionViewItemResult<Customer>
                {
                    Error = "",
                    Success = ModelState.IsValid,
                    Data = st
                });
            });
        }
        if (batchData.ItemsDeleted != null)
        {
            batchData.ItemsDeleted.ToList<Customer>().ForEach(customer =>
            {
                Customer fCustomer = db.Customers.Find(customer.CustomerID);
                db.Customers.Remove(fCustomer);
                itemresults.Add(new CollectionViewItemResult<Customer>
                {
                    Error = "",
                    Success = ModelState.IsValid,
                    Data = customer
                });
            });
        }
        if (batchData.ItemsUpdated != null)
        {
            batchData.ItemsUpdated.ToList<Customer>().ForEach(customer =>
            {
                db.Entry(customer).State = EntityState.Modified;
                itemresults.Add(new CollectionViewItemResult<Customer>
                {
                    Error = "",
                    Success = ModelState.IsValid,
                    Data = customer
                });
            });
        }
        db.SaveChanges();
    }
    catch (Exception e)
    {
        error = e.Message;
        success = false;
    }
    return new CollectionViewResponse<Customer>
    {
        Error = error,
        Success = success,
        OperatedItemResults = itemresults
    };
}, () => db.Customers.ToList<Customer>()));
```

```
}
```

BatchEditing.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models

<script type="text/javascript">
    function batchUpdate() {
        var batchEditGrid = wijmo.Control.getControl('#batchEditGrid'),
            cv = batchEditGrid.collectionView;
        cv.commit();
    }
</script>
<input type="button" value="Update" class="btn" onclick="batchUpdate()" />
@(Html.C1().FlexGrid<Customer>()
    .Id("batchEditGrid")
    .Bind(ib => ib.BatchEdit(Url.Action("GridBatchEdit"))
    .DisableServerRead(true)
    .Bind(Model))
    .Width(700)
    .Height(500)
    .AllowAddNew(true)
    .AllowDelete(true)
    .CssClass("grid")
)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models

<script type="text/javascript">
    function batchUpdate() {
        var batchEditGrid = wijmo.Control.getControl('#batchEditGrid'),
            cv = batchEditGrid.collectionView;
        cv.commit();
    }
</script>

<input type="button" value="Update" class="btn" onclick="batchUpdate()" />

<c1-flex-grid id="batchEditGrid" auto-generate-columns="false" class="grid"
              allow-add-new="true" allow-delete="true">
    <c1-flex-grid-column binding="CategoryID" is-read-only="true"></c1-flex-grid-
column>
    <c1-flex-grid-column binding="CategoryName"></c1-flex-grid-column>
```

```
    <c1-flex-grid-column binding="Description" width="*"></c1-flex-grid-column>
    <c1-items-source disable-server-read="true" read-action-
url="@Url.Action("BatchEditing_Bind")"
     batch-edit-action-url="@Url.Action("GridBatchEdit")"></c1-items-source>
</c1-flex-grid>
```

Excel Style Editing

FlexGrid has built-in support for Excel-like, fast, in-cell editing. There is no need to add extra columns with 'Edit' buttons that switch between display and edit modes.

Users can start editing by typing into any cell. This puts the cell in quick-edit mode. In this mode, pressing a cursor key finishes the editing and moves the selection to a different cell. You can also perform editing by pressing F2 or by clicking a cell twice. This puts the cell in full-edit mode. In this mode, pressing a cursor key moves the caret within the cell text. To finish editing and move to another cell, the user must press the Enter, Tab, or Escape key. Data is automatically updated when editing finishes.

You can disable editing at the Grid, Column, or Row levels using the IsReadOnly property of the Grid, Column, or Row objects. In this example, we make the ID column read-only.

The following image shows how the FlexGrid appears after enabling editing through **CollectionView**. The example uses Sale.cs model, which was added to the application in the QuickStart:

	ID	Country	Product	Color	Amount	Discount	Active
	1	UK	Widget	Black	($3,476.95)	1 %	☐
	2	UK	Widget	Red	$2,290.56	6 %	☐
	3	Japan	Gadget	White	($4,146.76)	1 %	☐
	4	Canada	Gadget	Red	$4,917.55	14 %	☐
	5	Korea	Gadget	Black	$3,824.28	8 %	☐
	6	Japan	Gadget	White	($4,257.83)	23 %	☐
	7	Italy	Gadget	Black	$1,095.08	20 %	☐
	8	German	Gadget	Green	$1,853.66	18 %	☐
	9	Italy	Widget	White	$3,708.86	3 %	☐
	10	Canada	Gadget	Green	($3,447.73)	10 %	☐
	11	German	Gadget	White	($165.64)	4 %	☐
	12	China	Widget	White	$160.62	8 %	☐
	13	Korea	Widget	Red	($2,575.30)	17 %	☐
	14	Italy	Widget	Red	$3,288.12	19 %	☐
	15	Korea	Gadget	Red	($4,563.28)	2 %	☐
	16	Canada	Widget	White	($4,839.71)	19 %	☐
	17	US	Gadget	Red	$544.13	10 %	☐
	18	Canada	Widget	Red	$538.45	4 %	☐
	19	Japan	Gadget	Red	$1,493.10	19 %	☐
	20	US	Widget	Black	($2,838.54)	14 %	☐

The following example demonstrates **CRUD** (Create, Read, Update and Delete) operations while grid is in editing

mode. Use the code below to enable excel-style editing in the FlexGrid:

In Code

Include the following references in your controller.

C#

```
using C1.Web.Mvc;
using MVCFlexGrid.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using C1.Web.Mvc.Serializition;
using System.Data;
using System.Collections;
using System.Globalization;
using System.Data.Entity;
```

Replace the default action method with the following code:

Create.cs

C# copyCode

```
public ActionResult Editing()
{
    return View(db.Customers.ToList());
}
public ActionResult GridCreate([C1JsonRequest]CollectionViewEditRequest<Customer>
requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Customer>(requestData, customer =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            db.Customers.Add(customer);
            db.SaveChanges();
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<Customer>
        {
            Error = error,
            Success = success && ModelState.IsValid,
```

```
            Data = customer
        };
    }, () => db.Customers.ToList<Customer>());
}
```

Update.cs

C# copyCode

```
public ActionResult Editing()
{
    return View(db.Customers.ToList());
}
public ActionResult GridUpdate([C1JsonRequest]CollectionViewEditRequest<Customer>
requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Customer>(requestData, customer =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            db.Entry(customer).State = EntityState.Modified;
            db.SaveChanges();
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<Customer>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = customer
        };
    }, () => db.Customers.ToList<Customer>());
}
```

Delete.cs

C# copyCode

```
public ActionResult Editing()
{
    return View(db.Customers.ToList());
}
public ActionResult GridDelete([C1JsonRequest]CollectionViewEditRequest<Customer>
requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Customer>(requestData, customer =>
```

```
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            Customer fCustomer = db.Customers.Find(customer.CustomerID);
            db.Customers.Remove(fCustomer);
            db.SaveChanges();
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
        return new CollectionViewItemResult<Customer>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = customer
        };
    }, () => db.Customers.ToList<Customer>()));
}
```

Make sure that you have defined 'db' object before action method in your controller as folllows:

C#

```
private C1NWindEntities db = new C1NWindEntities();
```

Editing.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
<div class="collapsed-content collapse">
</div>
@(Html.C1().FlexGrid<Customer>()
    .Id("editGrid")
    .Bind(
        ib => ib.Bind(Model)
          //Action Url for updating an item in the grid.
        .Update(Url.Action("GridUpdate"))
          // Action Url for adding a new item to the grid.
        .Create(Url.Action("GridCreate"))
          // Action Url for deleting an item from the grid.
        .Delete(Url.Action("GridDelete"))
    ).Columns(columns =>
        {
        columns.Add(column =>
column.Binding("Start").Width("80").MaxWidth(160).MinWidth(40));
```

```
        columns.Add(column =>
column.Binding("Product").Width("2*").AllowResizing(false));
        columns.Add(column => column.Binding("Amount").Format("c"));
        columns.Add(column => column.Binding("Amount2").Format("c"));
    })
    .Bind(Model)
    .AllowAddNew(true)
    .AllowDelete(true)
    .CssStyle("height", "400px")
)
```

Tag Helpers

HTML copyCode

```
<c1-flex-grid id="inlineEditGrid" is-read-only="true" selection-
mode="C1.Web.Mvc.Grid.SelectionMode.None"
              auto-generate-columns="false" item-formatter="itemFormatter"
style="height:400px">
    <c1-items-source read-action-url="@Url.Action("GridBindCustomer")"
                     update-action-url="@Url.Action("GridUpdateCustomer")"
                     create-action-url="@Url.Action("GridCreateCustomer")"
                     delete-action-url="@Url.Action("GridDeleteCustomer")"></c1-
items-source>
    <c1-flex-grid-column binding="CustomerID" width="80" align="right" is-read-
only="true"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country" name="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Address" width="*" name="Address"></c1-flex-grid-
column>
    <c1-flex-grid-column name="Buttons" width="170"></c1-flex-grid-column>
</c1-flex-grid>
```

Grouping

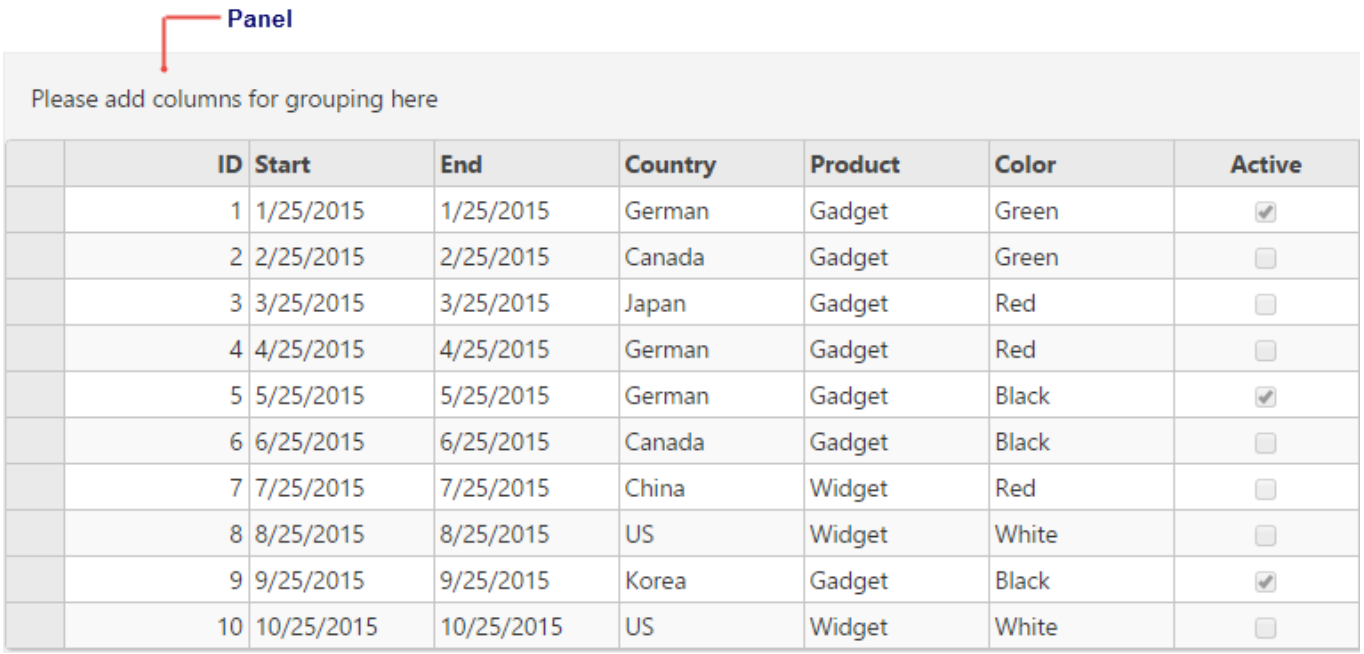
Grouping at Run-time

FlexGrid gives end users the flexibility to group and ungroup grid data as per the requirement. This is supported by **Group Panel** control, which provides drag and drop grouping UI to the grid. It enables you to drag desired columns from the FlexGrid into the panel for grouping.

You can easily move groups within the panel to change the grouping order of the grid data. Moreover, clicking a group marker in the panel enables sorting of the groups, based on the particular column entries.

Use MaxGroups property to set the maximum number of groups you want to allow in Group Panel. Set the Placeholder property to the string you want to display in the panel when it contains no groups. Set the HideGroupedColumns property to a boolean value, to specify whether you want to hide the grouped columns from the grid.

The following image shows how the FlexGrid appears on using Group Panel. The example uses **Sale.cs** model added in the Quick Start Section.



	ID	Start	End	Country	Product	Color	Active
	1	1/25/2015	1/25/2015	German	Gadget	Green	☑
	2	2/25/2015	2/25/2015	Canada	Gadget	Green	☐
	3	3/25/2015	3/25/2015	Japan	Gadget	Red	☐
	4	4/25/2015	4/25/2015	German	Gadget	Red	☐
	5	5/25/2015	5/25/2015	German	Gadget	Black	☑
	6	6/25/2015	6/25/2015	Canada	Gadget	Black	☐
	7	7/25/2015	7/25/2015	China	Widget	Red	☐
	8	8/25/2015	8/25/2015	US	Widget	White	☐
	9	9/25/2015	9/25/2015	Korea	Gadget	Black	☑
	10	10/25/2015	10/25/2015	US	Widget	White	☐

The following image shows how the FlexGrid appears on adding groups to the panel, at the top of the grid.

The following code examples demonstrate how to enable Group Panel in the FlexGrid:

In Code

GroupPanelController.cs

C# copyCode

```
public ActionResult GroupPanel(FormCollection data)
 {
   return View(Sale.GetData(10));
 }
```

GroupPanel.cshtml

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Grid
@using MVCFlexGrid.Models
@model IEnumerable<Sale>

@(Html.C1().FlexGrid<Sale>()
    .Id("ovFlexGrid").ShowGroupPanel(s => s
        .MaxGroups(Convert.ToInt32(4))
        .Placeholder("Please add columns for grouping here")
        .HideGroupedColumns(Convert.ToBoolean(false)))
    .AutoGenerateColumns(false)
    .Bind(Model)
    .PageSize(10)
    .CssClass("grid")
    .IsReadOnly(true)
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ID"));
        bl.Add(cb => cb.Binding("Start"));
        bl.Add(cb => cb.Binding("End"));
        bl.Add(cb => cb.Binding("Country"));
        bl.Add(cb => cb.Binding("Product"));
        bl.Add(cb => cb.Binding("Color"));
        bl.Add(cb => cb.Binding("Active"));
    })
)
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Grid
@using TagFlexGrid.Models

@model IEnumerable<Sale>

<c1-flex-grid id="ovFlexGrid" auto-generate-columns="false" is-read-only="true"
class="grid">
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="End"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Color"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>
    <c1-items-source read-action-url="@Url.Action("GroupPanel_Bind")"></c1-items-
```

```
source>
    <c1-flex-grid-group-panel max-groups="4"
                              placeholder="Please add columns for grouping here"
                              hide-grouped-columns(false)>
    </c1-flex-grid-group-panel>
</c1-flex-grid>
```

Grouping through Code

FlexGrid supports column-wise grouping of grid data through the CollectionView class as a datasource. You can configure group description by using the GroupBy method in view. You can also add grouping using JavaScript by adding GroupDescription objects to the collectionView.groupDescriptions property. Here, the column for grouping the grid data is specified in the code.

You can customize the text displayed in group header rows using the GroupHeaderFormat property. By default, it displays the name of the group, followed by the current group and the number of items in the group. To format the aggregated data in the group header for a particular column, you can set the format property on each Column object.

The following image shows how the FlexGrid appears on using the GroupBy method. The example uses table, **Customer** from the **NWind.mdb** database.

CustomerID	ContactName	Address	City	Region	PostalCode	Country
Country: **Germany12** (1 items)						
ALFKI	Maria Anders	Obere Str. 57	Berlin	null	12209	Germany12
Country: **Mexico** (3 items)						
ANATR	Ana Trujillo	Avda. de la C...	México D.F.		05021	Mexico
ANTON	Antonio Moreno	Mataderos 2312	México D.F.		05023	Mexico
CENTC	Francisco Ch...	Sierras de Gr...	México D.F.		05022	Mexico
Country: **UK** (4 items)						
AROUT	Thomas Hardy	120 Hanover ...	London		WA1 1DP	UK
BSBEV	Victoria Ashw...	Fauntleroy Cir...	London		EC2 5NT	UK
CONSH	Elizabeth Brown	Berkeley Gar...	London		WX1 6LT	UK

1 / 5

The following code examples demonstrate how to enable Grouping in the FlexGrid:

In Code

GroupingController.cs

C# copyCode

```
public ActionResult Grouping()
{
    var nwind = new C1NWindEntities();
    return View(nwind.Customers);
}
```

Grouping.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@model IEnumerable<Customer>

@(Html.C1().FlexGrid<Customer>()
.Id("pagingGrid")
.AutoGenerateColumns(false)
.Height(300)
.Width(840)
.IsReadOnly(true)
.Bind(Model)
.GroupBy("Country")
    .Columns(columns =>
        {
            columns.Add(column => column.Binding("CustomerID"));
            columns.Add(column => column.Binding("ContactName"));
            columns.Add(column => column.Binding("Address"));
            columns.Add(column => column.Binding("City"));
            columns.Add(column => column.Binding("Region"));
            columns.Add(column => column.Binding("PostalCode").Format("p0"));
            columns.Add(column => column.Binding("Country"));
        })
)
@Html.Partial("_Pager", "pagingGrid")
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Grid
@using TagFlexGrid.Models
@model IEnumerable<Customer>


<c1-flex-grid id="pagingGrid" height="350px" is-read-only="true" show-groups="true"
              auto-generate-columns="false" group-by="@(new string[] { "Country" })">
    <c1-items-source read-action-url="@Url.Action("Grouping_Bind")"></c1-items-
source>
    <c1-flex-grid-column binding="CustomerID" ></c1-flex-grid-column>
    <c1-flex-grid-column binding="ContactName"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Address"></c1-flex-grid-column>
    <c1-flex-grid-column binding="City"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Region"></c1-flex-grid-column>
    <c1-flex-grid-column binding="PostalCode" format="p0"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
</c1-flex-grid>
@Html.Partial("_Pager", "pagingGrid")
```

Columns

Unbound Columns in FlexGrid

FlexGrid supports selective column binding, which means the control can be customized to contain unbound columns apart from data bound columns. While the bound columns display data from a datasource, unbound columns contain custom data. The following example demonstrates this customization, using ItemFormatter.

In the below example, the values displayed in the cells of an unbound column of FlexGrid are calculated using values from the cells of other columns that are bound to a data source. The below code examples make use of rich client side API to customize FlexGrid cells through itemFormatter property.

The following image shows a FlexGrid having an unbound column **TargetValue**. This column displays the values obtained after dividing the values in **Amount** column by hundred.

The following code examples demonstrate how to add an unbound column in a FlexGrid control, and assign calculated values to its cells. The example uses sales data from **Sale.cs** model added in QuickStart.

In Code

UnboundcolumnController.cs

C# copyCode

```
public ActionResult Index()
{
    return View(Sale.GetData(15));
}
```

Unboundcolumn.cshtml

Include the MVC references as shown below.

C# copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>
```

Add the client script for itemFormatter function, to customize the cells of unbound column.

JavaScript copyCode

```
<script>
    function itemFormatter(panel, r, c, cell) {
        var flex = panel.grid;
        var col = panel.columns[c];
        //Unbound column which displays calculated value
        if (col.name == "TargetValue") {
            var calculatedData = parseFloat(flex.getCellData(r, 3, false) / 100);
            flex.setCellData(r, col.index, calculatedData, true);
        } //Unbound column which displays calculated value
    }
</script>
```

Then, initialize the FlexGrid control using the following code.

HTML Helpers

Razor copyCode

```
<h2>Sales Report</h2>
@(Html.C1().FlexGrid<Sale>()
.AutoGenerateColumns(false)
.Bind(Model)
.AllowResizing(AllowResizing.None)
.ItemFormatter("itemFormatter")
.GroupBy("Country")
.Columns(columns =>
    {
        columns.Add(column => column.Binding("ID"));
        columns.Add(column => column.Binding("Country"));
        columns.Add(column => column.Binding("Product"));
        columns.Add(column =>
column.Binding("Amount").Format("c").Width("*").Align("center").Aggregate(Aggregate.Sum));
        columns.Add(column => column.Header("TargetValue").Name("TargetValue"));
    }))
```

Tag Helpers

HTML copyCode

```
<h2>Sales Report</h2>
<c1-flex-grid id="Sales" auto-generate-columns="false" allow-resizing="AllowResizing.None"
item-formatter="itemFormatter" group-by="Country">
```

```
    <c1-flex-grid-column binding="ID" width="*" />
    <c1-flex-grid-column binding="Country" width="2*" />
    <c1-flex-grid-column binding="Product" width="2*" />
    <c1-flex-grid-column binding="Amount" width="*" format="c" align="center"
aggregate="Aggregate.Sum" />
    <c1-flex-grid-column header="TargetValue" name="TargetValue" width="2*" />
    <c1-items-source source-collection="Model" />
</c1-flex-grid>
```

Show or Hide Grid Columns on Client-Side

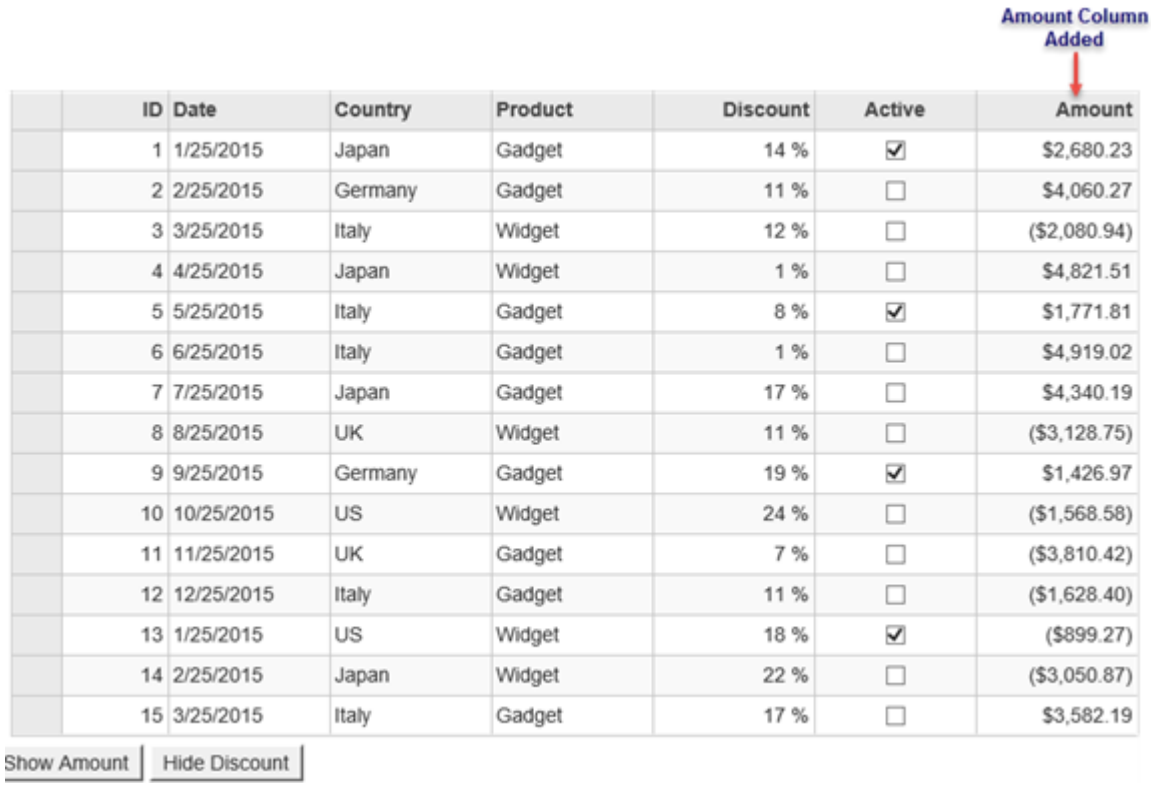
The FlexGrid control enables users to make specific grid columns visible or hidden on server side, using Visible property of FlexGrid. However, this operation can also be accomplished on the client side. Developers may find it appropriate to show a hidden column of grid on button click by end users, in their MVC application.

For instance, while analyzing the Sales data for your organization using FlexGrid, you are willing to view limited columns at first instance. Additional details, such as Amount for sales in respective months, you want to appear only when desired. This can be achieved on client side using JavaScript.

The below example depicts this scenario, where on button click you can make the Amount column visible.

	ID	Date	Country	Product	Discount	Active
	1	1/25/2015	Japan	Gadget	14 %	☑
	2	2/25/2015	Germany	Gadget	11 %	☐
	3	3/25/2015	Italy	Widget	12 %	☐
	4	4/25/2015	Japan	Widget	1 %	☐
	5	5/25/2015	Italy	Gadget	8 %	☑
	6	6/25/2015	Italy	Gadget	1 %	☐
	7	7/25/2015	Japan	Gadget	17 %	☐
	8	8/25/2015	UK	Widget	11 %	☐
	9	9/25/2015	Germany	Gadget	19 %	☑
	10	10/25/2015	US	Widget	24 %	☐
	11	11/25/2015	UK	Gadget	7 %	☐
	12	12/25/2015	Italy	Gadget	11 %	☐
	13	1/25/2015	US	Widget	18 %	☑
	14	2/25/2015	Japan	Widget	22 %	☐
	15	3/25/2015	Italy	Gadget	17 %	☐

Show Amount | Hide Discount



Similarly, on clicking **Hide Discount** button, you can hide the Discount column from the Grid.

You can use JavaScript in View of your MVC application as shown below, to hide or show any columns on button click. The below code extends application created in Quick Start topic.

HTML Helpers

Razor

```
<script>
function showAmount() {
                var grid = wijmo.Control.getControl("#hFlexGrid");
                //by column index
                grid.columns[6].visible = true;
                //or by column name
                grid.columns.getColumn("Amount").visible = true;
                }
                </script>
<script>
function hideDiscount() {
                var grid = wijmo.Control.getControl("#hFlexGrid");
                //by column index
                //grid.columns[4].visible = false;
                //or by column name
                grid.columns.getColumn("Discount").visible = false;
                }
</script>
<div class="container">
```

```
<div>
@(Html.C1().FlexGrid<Sale>().Id("hFlexGrid")
                .AutoGenerateColumns(false)
                .Bind(Model)
                .Columns(bl =>{
                bl.Add(cb => cb.Binding("ID").Header("ID"));
                bl.Add(cb => cb.Binding("End").Header("End").Format("MMM d yy"));
                bl.Add(cb => cb.Binding("Country").Header("Country"));
                bl.Add(cb => cb.Binding("Product").Header("Product"));
                bl.Add(cb => cb.Binding("Discount").Format("p0"));
                bl.Add(cb => cb.Binding("Active"));
                bl.Add(cb => cb.Binding("Amount").Format("n0").Visible(false));
                })
                )
</div>
</div>
                <button title="SetValue" onclick="showAmount()">Show Amount</button>
        <button title="SetValue" onclick="hideDiscount()">Hide Discount</button>
```

Tag Helpers

HTML copyCode

```
<script>
    function showAmount() {
        var grid = wijmo.Control.getControl("#hFlexGrid");
        //by column index
        grid.columns[6].visible = true;
        //or by column name
        grid.columns.getColumn("Amount").visible = true;
    }
</script>
<script>
    function hideDiscount() {
        var grid = wijmo.Control.getControl("#hFlexGrid");
        //by column index
        grid.columns[4].visible = false;
        //or by column name
        grid.columns.getColumn("Discount").visible = false;
    }
</script>

<div class="container">
    <div>
        <c1-flex-grid id="hFlexGrid" auto-generate-columns="false" height="700px"
width="800px" allow-add-new="true" class="grid">
            <c1-items-source source-collection="Model"></c1-items-source>
            <c1-flex-grid-column binding="ID" header="ID"></c1-flex-grid-column>
            <c1-flex-grid-column binding="Date" header="Date"></c1-flex-grid-column>
            <c1-flex-grid-column binding="Country" header="Country"></c1-flex-grid-
```

```
column>
            <c1-flex-grid-column binding="Product" header="Product"></c1-flex-grid-
column>
            <c1-flex-grid-column binding="Discount" header="Discount" format="p0">
</c1-flex-grid-column>
            <c1-flex-grid-column binding="Active"></c1-flex-grid-column>
            <c1-flex-grid-column binding="Amount" format="c" is-visible="false"></c1-
flex-grid-column>
        </c1-flex-grid>
    </div>
</div>

<button title="SetValue" onclick="showAmount()">Show Amount</button>
<button title="SetValue" onclick="hideDiscount()">Hide Discount</button>
```

Back to Top

Save and Load Column Layout

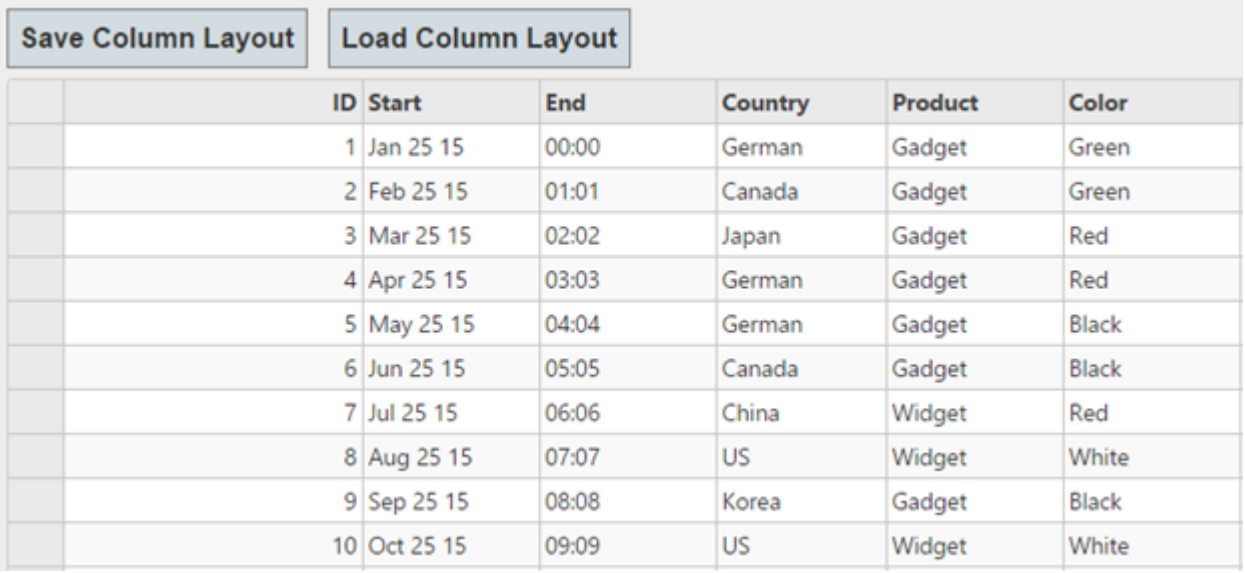
FlexGrid has a client ColumnLayout property that saves the column layout in the browser's localStorage. ColumnLayout gets or sets a JSON string containing a list of grid columns and their properties. It doesn't support the datamap column in the FlexGrid.

To see how it works, you can try doing the following:

1. Resize some columns and drag some to new positions.
2. Click the "Save Column Layout" button to save the changes to local storage.
3. Refresh the page to restore the original layout.
4. Click the "Load Column Layout" button to restore the layout from local storage.

> FlexGrid supports server-side saving\loading of column layout. This is useful in situations where column layout needs to be saved in a database for presenting different views to different users.

The following image shows how a FlexGrid when **ColumnLayout** property is used to save the layout of FlexGrid. The example uses **Sale.cs** model as its datasource, added in the QuickStart section.



The following code examples demonstrate how to set Save and Load Column layout in a FlexGrid:

In Code

SaveColumnLayoutController.cs

C# copyCode

```
public ActionResult Index()
{
    var model = Sales.GetData(500);
    return View(model);
}
```

SaveColumnLayout.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

<script>
    // Save or restore column layout in localStorage
    function saveColumnLayout() {
        var grid = wijmo.Control.getControl("#ovFlexGrid");
        localStorage['columns'] = grid.columnLayout;
    }
    function loadColumnLayout() {
        var grid = wijmo.Control.getControl("#ovFlexGrid"),
            columnLayout = localStorage['columns'];
        if (columnLayout) {
            grid.columnLayout = columnLayout;
        }
    }
</script>
<input type="button" value="Save Column Layout" class="btn"
onclick="saveColumnLayout()" />
<input type="button" value="Load Column Layout" class="btn"
onclick="loadColumnLayout()" />

@(Html.C1().FlexGrid<Sale>()
    .Id("ovFlexGrid")
    .AutoGenerateColumns(true)
    .AllowSorting(true)
    .SelectionMode(SelectionMode.Cell)
    .Bind(bl => bl.InitialItemsCount(100).Bind(Model))
    .CssClass("grid")
    .IsReadOnly(true)
```

```
)
```

Tag Helpers

HTML

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid;
@model IEnumerable<Sale>

<script>
    // Save or restore column layout in localStorage

    function saveColumnLayout() {
        var grid = wijmo.Control.getControl("#ovFlexGrid");
        localStorage['columns'] = grid.columnLayout;
    }
    function loadColumnLayout() {
        var grid = wijmo.Control.getControl("#ovFlexGrid"),
            columnLayout = localStorage['columns'];
        if (columnLayout) {
            grid.columnLayout = columnLayout;
        }
    }
</script>
<input type="button" value="Save Column Layout" class="btn"
onclick="saveColumnLayout()" />
<input type="button" value="Load Column Layout" class="btn"
onclick="loadColumnLayout()" />

<c1-flex-grid id="ovFlexGrid" auto-generate-columns="true" class="grid" is-read-
only="true" allow-sorting="true">
    <c1-items-source initial-items-count="100" source-collection="Model"></c1-items-
source>
</c1-flex-grid>
```

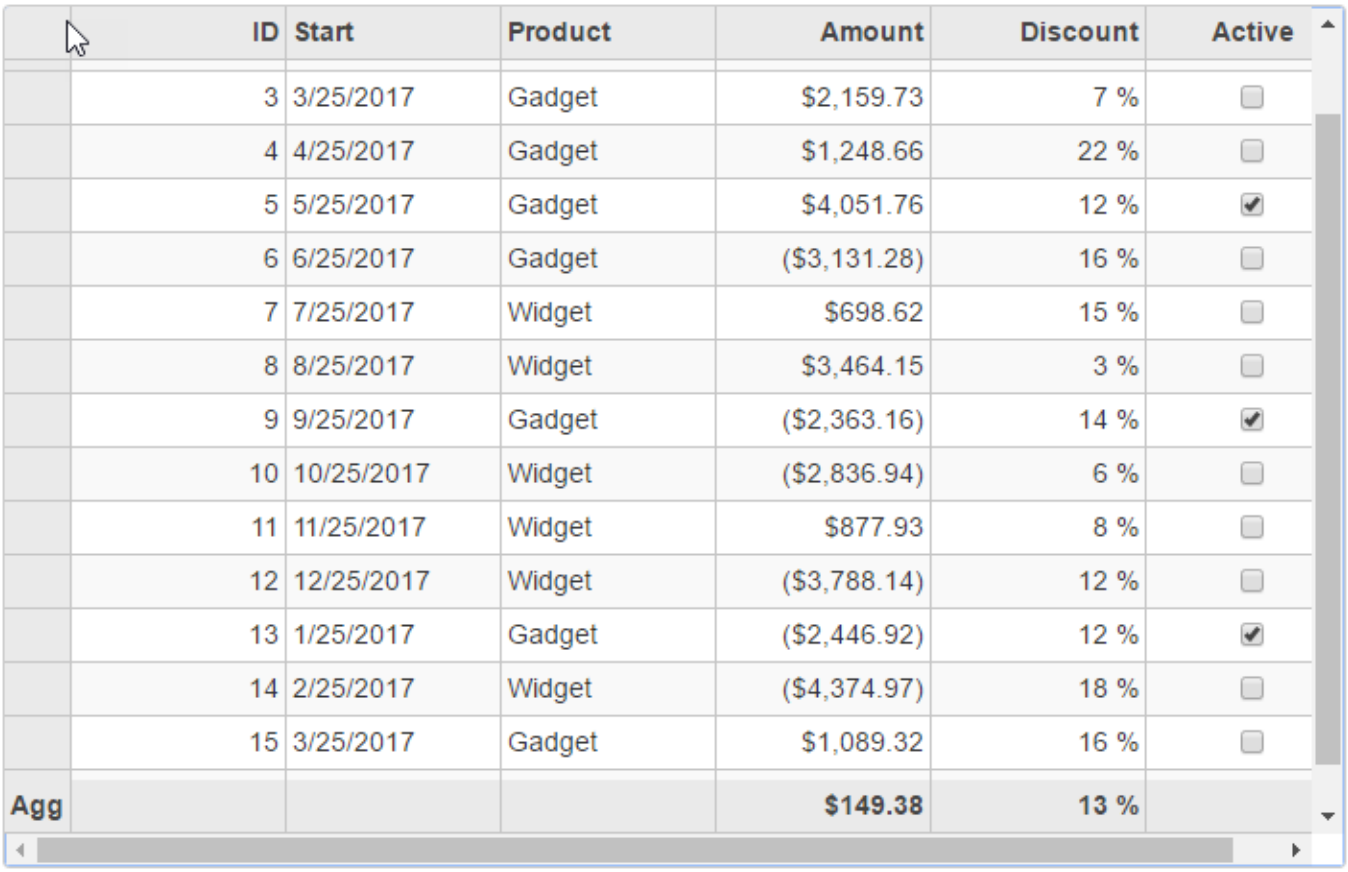
Column Footer Panel

FlexGrid provides support for column footers which allows you to show the group row to display the aggregate data. You can calculate the aggregate data and display the result in the column footer panel. To show the column footer panel in the FlexGrid, the ShowColumnFooters property should be set to **true**. You can also set the row header text of the group row; if not set, it will use the default value, a sigma character (Σ).

The ShowColumnFooter property shows the aggregate value for all the values for the current column:

1. When DisableServerRead is **true**, it fetches all the row data and calculate the aggregate for all the rows.
2. When DisableServerRead is **false** and InitialItemsCount is set, the grid using virtual scrolling and not all rows data are sent to client. Therefore, the aggregate value maybe incorrect, since it is only for partial rows.

The following image shows how the FlexGrid appears after setting the ShowColumnFooters property. The example uses Sale.cs model added in the QuickStart topic.



	ID	Start	Product	Amount	Discount	Active
	3	3/25/2017	Gadget	$2,159.73	7 %	☐
	4	4/25/2017	Gadget	$1,248.66	22 %	☐
	5	5/25/2017	Gadget	$4,051.76	12 %	☑
	6	6/25/2017	Gadget	($3,131.28)	16 %	☐
	7	7/25/2017	Widget	$698.62	15 %	☐
	8	8/25/2017	Widget	$3,464.15	3 %	☐
	9	9/25/2017	Gadget	($2,363.16)	14 %	☑
	10	10/25/2017	Widget	($2,836.94)	6 %	☐
	11	11/25/2017	Widget	$877.93	8 %	☐
	12	12/25/2017	Widget	($3,788.14)	12 %	☐
	13	1/25/2017	Gadget	($2,446.92)	12 %	☑
	14	2/25/2017	Widget	($4,374.97)	18 %	☐
	15	3/25/2017	Gadget	$1,089.32	16 %	☐
Agg				**$149.38**	**13 %**	

In Code

HTML Helpers

Razor

```
@using <ApplicationName>.Models
@using C1.Web.Mvc.Grid

@model IEnumerable<Sale>
<br />
@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .Height(450)
    .Width(700)
    .AllowAddNew(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .CssClass("grid")
    .ShowColumnFooters(true, "Agg")
    .Bind(Model)
     .Columns(bl =>
     {
         bl.Add(cb => cb.Binding("ID"));
         bl.Add(cb => cb.Binding("Start"));
         bl.Add(cb => cb.Binding("Product"));
         bl.Add(cb => cb.Binding("Amount").Format("c").Aggregate(Aggregate.Sum));
```

```
        bl.Add(cb => cb.Binding("Discount").Format("p0").Aggregate(Aggregate.Avg));
        bl.Add(cb => cb.Binding("Active"));
    })
)
```

Tag Helpers

HTML

```
@using <ApplicationName>.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

<c1-flex-grid auto-generate-columns="false" height="500px" width="800px" class="grid"
              is-read-only="true" allow-add-new="true"
              allow-sorting="true"
              selection-mode="@((SelectionMode.Cell))"
              show-column-footers="true" column-footers-row-header-text="Agg">
    <c1-items-source read-action-url="@Url.Action("Index_Bind")"></c1-items-source>
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c" aggregate="Sum"></c1-flex-grid-
column>
    <c1-flex-grid-column binding="Discount" format="p0" aggregate="Avg"></c1-flex-
grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>
</c1-flex-grid>
```

Unbound FlexGrid

Unbound FlexGrid using CollectionView Service

FlexGrid for MVC does not support unbound mode, however you can achieve this by adding code on the client side to set the data using JavaScript or TypeScript.

To achieve this we will be using CollectionViewService which binds to the model. At client side we get a reference of the CollectionViewService and add rows to the FlexGrid as per the number of items in the **Sale.cs** model. Finally we get the data of the three columns and add to FlexGrid using **setCellData** method.

The following image shows how you can set the data from the client side using CollectionViewService. The example uses Sale.cs model added in the QuickStart section.

	Product	Country	Amount
	Gadget	German	581.61
	Gadget	Canada	4,919.02
	Gadget	Japan	2,159.73
	Gadget	German	1,248.66
	Gadget	German	4,051.76
	Gadget	Canada	-3,131.28
	Widget	China	698.62
	Widget	US	3,464.15
	Gadget	Korea	-2,363.16
	Widget	US	-2,836.94

The following code examples demonstrate how to set the data by adding code to the client side.

App.ts (TypeScript file)

App.ts

```
var fg;
c1.documentReady(function () {
    populate();
})
function populate() {
    var fg = <wijmo.grid.FlexGrid>wijmo.Control.getControl("#fg");
    var cv = <wijmo.collections.CollectionView>c1.getService('cv');
    var total = cv.items.length;

    for (var i = 0; i <= total - 1; i++) {
        var obj = [cv.items[i].Product, cv.items[i].Country, cv.items[i].Amount];
        var row = new wijmo.grid.Row();
        fg.rows.push(row);
        for (var c = 0; c <= fg.columns.length - 1; c++) {
            fg.setCellData(i, c, obj[c]);
        }
    }
}
```

Index.cshtml

HTML Helpers

Razor

```
@using UnboundFlexGrid.Models
<script src="~/Scripts/app.js"></script>
@model IEnumerable<Sale>

@(Html.C1().CollectionViewService().Id("cv").Bind(Model))
```

```
@(Html.C1().FlexGrid().Id("fg").Height(400).AutoGenerateColumns(false).AllowAddNew(false)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .Columns( col => {
        col.Add(cb => cb.Name("Product").Header("Product"));
        col.Add(cb => cb.Name("Country").Header("Country"));
        col.Add(cb => cb.Name("Amount").Header("Amount"));
        }))
```

Tag Helpers

Razor

```
@using UnboundFlexGrid.Models
@using C1.Web.Mvc.Grid
<script src="~/Scripts/app.js"></script>
@model IEnumerable<Sale><c1-items-source id="cv" source-collection="Model"></c1-items-
source><c1-flex-grid id="fg" height="400px" auto-generate-columns="false" allow-
add-new="false" selection-mode="@((SelectionMode.Cell))">
<c1-flex-grid-column name="Product" header="Product"></c1-flex-grid-column>
<c1-flex-grid-column name="Country" header="Country"></c1-flex-grid-column>
<c1-flex-grid-column name="Amount" header="Amount"></c1-flex-grid-column>
</c1-flex-grid>
```

Unbound FlexGrid using AJAX Call

FlexGrid when in bound mode, the data in the grid is populated automatically from a particular data source. However, in case of an unbound FlexGrid, you need to add/remove the grid columns and rows using different method provided in the API.

This topic describes how to populate data in an unbound FlexGrid at client side using AJAX. The data from the server will be stored in a dictionary, which is serialized to JSON before sending it to the client.

This topic comprises the following steps:

- **Step 1: Configure MVC Application**
- **Step 2: Create a Model**
- **Step 3: Add a Controller Action**
- **Step 4: Add FlexGrid to View**
- **Step 5: Generate Data in FlexGrid using AJAX**
- **Step 6: Build and Run the Project**

The following image shows how you can populate data in an unbound FlexGrid at client side using AJAX. The example uses **Sale.cs** model added in the QuickStart section.

AJAX Load

	Product	Country	Amount
	Gadget	German	581.61
	Gadget	Canada	4,919.02
	Gadget	Japan	2,159.73
	Gadget	German	1,248.66
	Gadget	German	4,051.76
	Gadget	Canada	-3,131.28
	Widget	China	698.62
	Widget	US	3,464.15
	Gadget	Korea	-2,363.16
	Widget	US	-2,836.94

Step 1: Configure MVC Application

1. In Visual Studio, create a new MVC application using **C1 ASP.NET MVC 5 Web Application** template. For more information on how to create an MVC application, see Using ComponentOne Template topic.

2. In **ComponentOne ASP.NET MVC Application Wizard**, check **Enable Client IntelliSense** and **Add TS** options.

Back to Top

Step 2: Create a Model

1. Add a new class to the folder **Models** (For example: `Sale.cs`). For more information on how to add a new model, see Adding Controls.
2. Replace the following code in the `Sale.cs` model. We are using **Sale** class to represent sales order data in the database. Each instance of **Sale** object will correspond to a record in the FlexGrid control.

C#

```
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; }

private static List <string> COUNTRIES = new List<string> { "US", "UK", "Canada",
"Japan", "China", "France", "German", "Italy", "Korea", "Australia" };
private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget", "Doohickey"
};
        /// <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 startDate = new DateTime(dt.Year, i % 12 + 1, 25);
                var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60, i %
60);

                return new Sale
                {
                    ID = i + 1,
                    Start = startDate,
                    End = endDate,
                    Country = country,
                    Product = product,
                    Color = color,
                    Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Discount = Math.Round(rand.NextDouble() / 4, 2),
                    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;
        }
        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;
        }
    }
    public class MonthData
    {
        public int Month { get; set; }
        public double Data { get; set; }
    }
    public class BasicSale
    {
```

```
        public int Sale { get; set; }
        public DateTime Date { get; set; }

        public BasicSale(int sale, DateTime date)
        {
            Sale = sale;
            Date = date;
        }
        public static List<BasicSale> GetBasicSales()
        {
            List<BasicSale> list = new List<BasicSale>();
            int[] sales = {
            96, 19, 54, 83, 15, 56, 36, 4, 29, 93,
            38, 71, 50, 77, 69, 13, 79, 57, 29, 62,
            4, 27, 66, 96, 65, 12, 52, 3, 61, 48, 50,
            70, 39, 33, 25, 49, 69, 46, 44, 40, 35,
            72, 64, 10, 66, 63, 78, 19, 96, 26};
            for (int i = 0; i < sales.Length; i++)
            {
                list.Add(new BasicSale(sales[i], new DateTime(2014, i / 31 + 1, i % 31 +
1)));
            }
            return list;
        }
    }
```

Back to Top

Step 3: Add a Controller Action

1. Open the `HomeController.cs` from the Controllers folder.
2. Add the following code to create a dictionary with a unique key and employee object in the **GetDictionaryData** function. Also, you need to serialize the dictionary data to JSON using JavaScriptSerializer in **GetData** function.

C#

```
public ActionResult Index()
  {
  return View();
  }
  [HttpPost]
  public JsonResult GetData()
  {
  // return Json(JsonConvert.SerializeObject(GetDictionaryData()));
  JavaScriptSerializer sz = new JavaScriptSerializer();
  string str = sz.Serialize(GetDictionaryData());
  return Json(str);
  }
  private Dictionary<string,Sale> GetDictionaryData()
  {
  var dict = new Dictionary<string, Sale>();
  var sales = Sale.GetData(10);
  for(int i=0; i<=sales.Count()-1;i++)
  {
  dict.Add(i.ToString(), sales.ElementAt(i));
  }
  return dict;
  }
```

Back to Top

Step 4: Add FlexGrid to View

1. Open the `Index.html` from View/Home folder.
2. Replace the content of this file with the following code to configure a FlexGrid control. The code below declares different columns and properties of the FlexGrid, but does not bind data to the control.

HTML Helpers

Razor

```
@using <ApplicationName>.Models
<script src="~/Scripts/app.js"></script>

@* Loads from a dictionary *@
<input type="button" onclick="Load()" value="Ajax Load" />

@(Html.C1().FlexGrid().Id("fg").Height(400).AutoGenerateColumns(false).AllowAddNew(false)
  .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
  .Columns( col => {
  col.Add(cb => cb.Name("Product").Header("Product"));
  col.Add(cb => cb.Name("Country").Header("Country"));
  col.Add(cb => cb.Name("Amount").Header("Amount"));
}))
```

Tag Helpers

HTML

```
@using <ApplicationName>.Models
<script src="~/Scripts/app.js"></script>

@* Loads from a dictionary *@
<input type="button" onclick="Load()" value="Ajax Load" />

<c1-flex-grid id="fg" height="400px" auto-generate-columns="false" allow-add-new="false"
              selection-mode="@((SelectionMode.Cell))">
<c1-flex-grid-column name="Product" header="Product"></c1-flex-grid-column>
<c1-flex-grid-column name="Country" header="Country"></c1-flex-grid-column>
<c1-flex-grid-column name="Amount" header="Amount"></c1-flex-grid-column>
</c1-flex-grid>
```

Back to Top

Step 5: Generate Data in FlexGrid using AJAX

1. In the **Solution Explorer**, open the **app.ts** file from the **Scripts** folder.
2. In **app.ts** file, add the following script to populate data at the client side in the FlexGrid control.

app.ts

```
/// <reference path="typings/c1.mvc.core.lib.d.ts" />
function Load() {
    $.ajax({
        type: "POST",
        url: "/Home/GetData",
        dataType: "json",
```

```
        success: function (result) {
            var fg = <wijmo.grid.FlexGrid>wijmo.Control.getControl("#fg"); //get FlexGrid
reference
            var data = JSON.parse(result);  //parse the server Dictionary data to JSON
array
            fg.rows.clear();  //clear any existing rows from FlexGrid
         var j=0
            for (var i in data) {
                var obj = [data[i].Product, data[i].Country, data[i].Amount];
                var row = new wijmo.grid.Row();  // add row to FlexGrid
                fg.rows.push(row);
                for (var col = 0; col <= fg.columns.length - 1; col++) {
                    fg.setCellData(j, col, obj[col]);  //add data to FlexGrid Cell.
                }
                j++;
            }

        },
        error: function (err) {
            alert("err");
        }
    });
```

Back to Top

Step 6: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project and then, click **AJAX Load** button to load and display the data in the FlexGrid.

Back to Top

Unobtrusive Validation

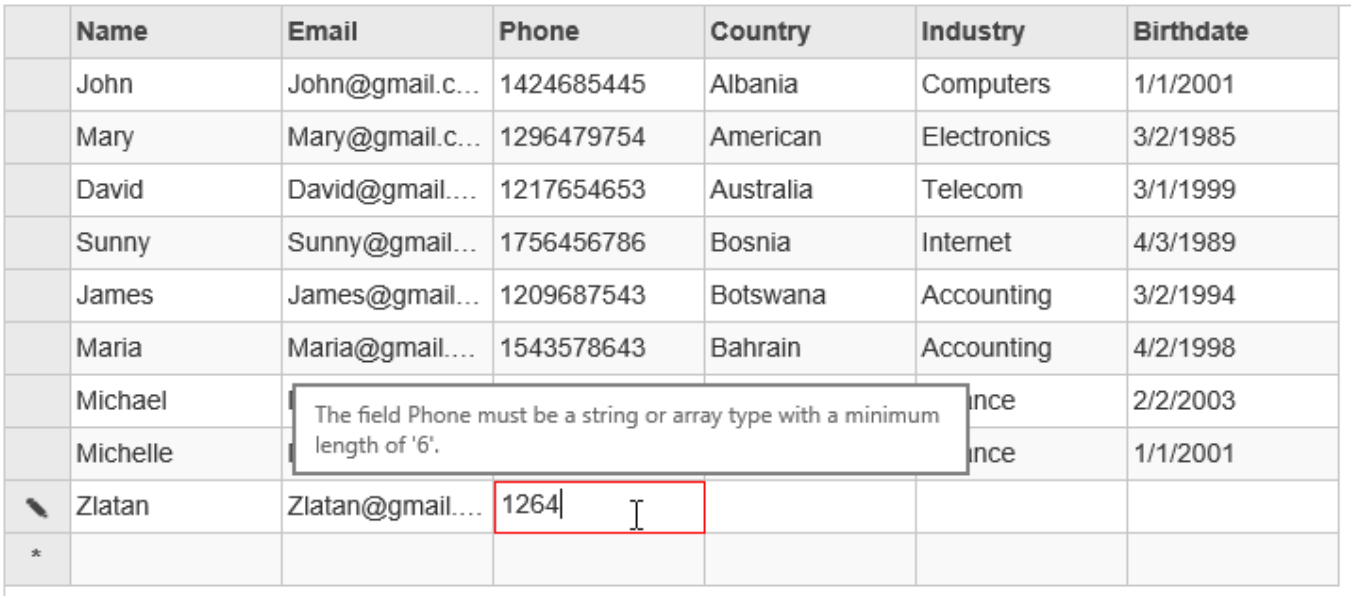
Unobtrusive validation support for FlexGrid helps you to validate the data that you enter in the FlexGrid at client side. Unobtrusive Validation can implement simple client-side validation without writing a bulk of validation code, and improves the user experience simply by adding the right attributes and including the script files.

In a common validation scenario, when we use a validation to validate any control and use client side validation, JavaScript code is generated and rendered as HTML on the web browser. However, with unobtrusive validation inline javascript is not generated for rendering to handle client side validation. Instead, it uses HTML5 data-* attributes for client side validations. Unobtrusive validation support is jQuery dependent.

Before implementing the below steps, you need to create a new MVC application using ComponentOne template or Visual Studio template.

This topic comprises of four steps:

- **Step 1: Configure your MVC application**
- **Step 2: Create Validations and Datasource for FlexGrid**
- **Step 3: Add a FlexGrid control**
- **Step 4: Build and Run the Project**



Step 1: Configure your MVC application

Using HTML Helpers

Complete the following steps to add the js file references to your application.

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click **_Layout.cshtml** to open it.
3. Add the following code between the <head></head> tags.

Razor

```
<script src="~/Scripts/jquery-1.10.2.js"></script>
<script src="~/Scripts/jquery.validate.js"></script>
<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
```

Using Tag Helpers

In case of tag helpers, references to the script files are automatically added to your application.

Step 2: Create Validations and Datasource for FlexGrid

Model - UserInfo.cs (Includes Validations)

Razor

```
using System.ComponentModel.DataAnnotations;
namespace UnobtrusiveValidation.Models
{
     public class UserInfo
    {
        [Required]
        [RegularExpression(pattern: "^[a-zA-Z0-9]{4,10}$", ErrorMessage = "The
username must be alphanumeric and contains 4 to 10 characters.")]
        public string Name { get; set; }
```

```
        [Required]
        [EmailAddress]
        public string Email { get; set; }
        [Required]
        [MinLength(6)]
        [MaxLength(16)]
        public string Phone { get; set; }
        [Required]
        public string Country { get; set; }
        [Required]
        public string Industry { get; set; }
        [Required]
        public DateTime Birthdate { get; set; }
    }
}
```

Back to Top

Model - UserData.cs (Includes Data)

Razor

```
public class UserData
    {
        public static List<UserInfo> Users
        {
            get
            {
                return new List<UserInfo>()
                {
                    new UserInfo(){ Name="John", Email="John@gmail.com",
Phone="1424685445", Country="Albania", Industry="Computers", Birthdate=
DateTime.Parse("2001/1/1")},
                    new UserInfo(){ Name="Mary", Email="Mary@gmail.com",
Phone="1296479754", Country="American", Industry="Electronics", Birthdate=
DateTime.Parse("1985/3/2")},
                    new UserInfo(){ Name="David", Email="David@gmail.com",
Phone="1217654653", Country="Australia", Industry="Telecom", Birthdate=
DateTime.Parse("1999/3/1")},
                    new UserInfo(){ Name="Sunny", Email="Sunny@gmail.com",
Phone="1756456786", Country="Bosnia", Industry="Internet", Birthdate=
DateTime.Parse("1989/4/3")},
                    new UserInfo(){ Name="James", Email="James@gmail.com",
Phone="1209687543", Country="Botswana", Industry="Accounting", Birthdate=
DateTime.Parse("1994/3/2")},
                    new UserInfo(){ Name="Maria", Email="Maria@gmail.com",
Phone="1543578643", Country="Bahrain", Industry="Accounting", Birthdate=
DateTime.Parse("1998/4/2")},
                    new UserInfo(){ Name="Michael", Email="Michael@gmail.com",
Phone="1215457467", Country="Argentina", Industry="Finance", Birthdate=
DateTime.Parse("2003/2/2")},
                    new UserInfo(){ Name="Michelle", Email="Michelle@gmail.com",
Phone="1534357546", Country="Bulgaria", Industry="Finance", Birthdate=
```

```
DateTime.Parse("2001/1/1")}
                };
            }
        }
    }
```

Back to Top

Step 3: Add a FlexGrid control

UnobtrusiveValidationController.cs

Razor

```
public class IndexController : Controller
    {
        private static List<UserInfo> users = UserData.Users;

        public IActionResult Index()
        {
            return View(users);
        }

    }
```

View - Index.cshtml

HTML Helpers

Razor

```
@using FlexGridValidationsHTMLHelpers.Models;
@model List<UserInfo>

@(Html.C1().FlexGrid<UserInfo>()
    .Id("flexGrid")
    .AutoGenerateColumns(false)
    .Columns(columns => columns
        .Add(c => c.Binding("Name"))
        .Add(c => c.Binding("Email"))
        .Add(c => c.Binding("Phone"))
        .Add(c => c.Binding("Country"))
        .Add(c => c.Binding("Industry"))
        .Add(c => c.Binding("Birthdate"))
    )
    .Bind(ib => ib.Bind(Model))
    .AllowAddNew(true)
    .AllowDelete(true)
    .CssStyle("height", "400px")
)
```

Tag Helpers

Razor

```
@using ValidationFlexGrid.Models
@model List<UserInfo>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
<br />
<c1-flex-grid id="flexGrid" auto-generate-columns="false" allow-add-new="true" allow-delete="true" height="400px">
    <c1-flex-grid-column binding="Name"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Email"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Phone"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Industry"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Birthdate"></c1-flex-grid-column>
    <c1-items-source source-collection="Model">
    </c1-items-source>
</c1-flex-grid>
```

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.

Back to Top

Limitations

- FlexGrid unobtrusive validation does not support Boolean data in check box cell because when you click it, it will end the editing directly.
- FlexGrid unobtrusive validation does not support bind to read action URL because cannot get model meta data from read action.
- FlexGrid unobtrusive validation does not support datamap column because can not get the actual value when editing.
- FlexGrid unobtrusive validation does not support cell template because the "onPrepareCellForEdit" not triggered.

Excel Import and Export

This topic explains how to export or import FlexGrid content to/from an Excel xlsx file. To export the FlexGrid content in Excel format, pass the FlexGrid instance to the **FlexGridXlsxConverter.save** method. This generates xlsx file content, which can be saved to local file or sent to a server. To import an Excel (xlsx file) content to your FlexGrid control, pass the FlexGrid instance and the xlsx file content to the **FlexGridXlsxConverter.load** method. The example uses **Sale.cs** model, which was added to the application in the QuickStart.

The following code examples demonstrate how to export or import FlexGrid content to/from an Excel xlsx file:

Add References

Before implementing the code below, you need to add the **jszip.min.js** library reference to import and export FlexGrid to/from an Excel xlsx file.

_Layout.cshtml

HTML

```
<!-- SheetJS library -->
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.2.1/jszip.min.js">
</script>
```

HomeController.cs

C#

```
public ActionResult Index()
{
         return View(Sale.GetData(500));
}
```

App.js

JavaScript

```
//Controls Declaration
var gFlexGrid = null,
    IncludeHeadersExport = null,
    IncludeHeadersImport = null;

//Controls Initialization
function InitialControls() {
    gFlexGrid = wijmo.Control.getControl("#gFlexGrid");
    IncludeHeadersImport = document.getElementById('IncludeHeadersImport');
    IncludeHeadersExport = document.getElementById('IncludeHeadersExport');
}

// export
function exportExcel() {
    if (gFlexGrid) {
        wijmo.grid.xlsx.FlexGridXlsxConverter.save(gFlexGrid, { includeColumnHeaders:
IncludeHeadersExport.checked }, 'FlexGrid.xlsx');
    }
};

// import
function importExcel() {
    if (gFlexGrid) {
        if ($('#importFile')[0].files[0]) {
            wijmo.grid.xlsx.FlexGridXlsxConverter.load(gFlexGrid, $('#importFile')
[0].files[0], { includeColumnHeaders: IncludeHeadersImport.checked });
        }
        else {
            alert('Select an Excel file to Import.');
        }
```

```
    }
};
```

Index.cshtml

HTML Helpers

Razor

```
@using C1MVCExcelImportExport.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

@{
    ViewBag.Title = "C1 ASP.NET MVC Excel Import Export";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<!-- App scripts -->
<script src="@Url.Content("~/Scripts/app.js")" type="text/javascript"></script>

<div class="container">
    <div class="row">
        <div class="col-md-6 col-xs-12">
            <div class="form-inline well well-lg">
                <input type="file" class="form-control" id="importFile"
accept="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" />
                <button class="btn btn-default"
onclick="importExcel()">Import</button>
                <br />
                <div class="checkbox">
                    <label>
                        <input id="IncludeHeadersImport" type="checkbox"
checked="checked"> Include Column Headers
                    </label>
                </div>
            </div>
        </div>
        <div class="col-md-6 col-xs-12">
            <div class="form-inline well well-lg">
                <a href="#" class="btn btn-default" id="export"
onclick="exportExcel()">Export</a>
                <div class="checkbox">
                    <label>
                        <input id="IncludeHeadersExport" type="checkbox"
checked="checked"> Include Column Headers
                    </label>
                </div>
            </div>
        </div>
    </div>
```

```
    <div class="row">
@(Html.C1().FlexGrid().Id("gFlexGrid").AutoGenerateColumns(false).AllowSorting(true)
.("max-height", "400px").IsReadOnly(true)
Bind(bl => bl.DisableServerRead(true).Bind(Model)).CssClass("grid")
Columns(bl =>
        {
        bl.Add(cb => cb.Binding("Country").Header("Country").Width("*"));
        bl.Add(cb => cb.Binding("Product").Header("Product").Width("*"));
        bl.Add(cb => cb.Binding("Color").Header("Color").Width("*"));
        bl.Add(cb => cb.Binding("Start").Header("Start").Width("*"));
        bl.Add(cb => cb.Binding("Amount").Header("Amount").Format("n0").Width("*")
        .Aggregate(C1.Web.Mvc.Grid.Aggregate.Sum));
        })
)
    </div>
</div>

<script type="text/javascript">
    c1.documentReady(function () {
        if (window["InitialControls"]) {
            window["InitialControls"]();
        }
    });
</script>
```

Tag Helpers

Razor

```
@using C1MVCExcelImportExport.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

@{
    ViewBag.Title = "C1 ASP.NET MVC Excel Import Export";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<!-- App scripts -->
<script src="@Url.Content("~/Scripts/app.js")" type="text/javascript"></script>

@section Settings{
    <div class="col-md-6 col-xs-12">
        <div class="form-inline well well-lg">
            <button class="btn btn-default" id="importBtn"
onclick="importGrid();">Import</button>
            <input type="file" id="importFile" class="form-control" />
        </div>
    </div>
    <div class="col-md-6 col-xs-12">
        <div class="form-inline well well-lg">
```

```
            <a download="FlexGrid.xlsx" class="btn btn-default" id="exportBtn"
onclick="exportGrid();">Export</a>
        </div>
    </div>
    <div class="checkbox-div">
        <label>
            <input type="checkbox" id="colHeaderCheckBox" class="checkbox"
checked="checked" /> Include Column Headers
        </label>
    </div>
}
<c1-flex-grid id="importExportFlexGrid" show-groups="true"
group-by="Product,Country,Amount"
              is-read-only="true" class="grid" auto-generate-columns="false">
    <c1-items-source initial-items-count="500" source-collection="Model"></c1-items-
source>
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start" header="Start Date" format="d"></c1-flex-
grid-column>
    <c1-flex-grid-column binding="End" header="End Date" format="d"></c1-flex-grid-
column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Color"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c" aggregate="Sum"></c1-flex-grid-
column>
    <c1-flex-grid-column binding="Amount2" header="Pending" format="c2"
aggregate="Sum"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" format="p1" aggregate="Avg"></c1-flex-
grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>
</c1-flex-grid>

<script type="text/javascript">
    c1.documentReady(function () {
        if (window["InitialControls"]) {
            window["InitialControls"]();
        }
    });
</script>
```

Paging

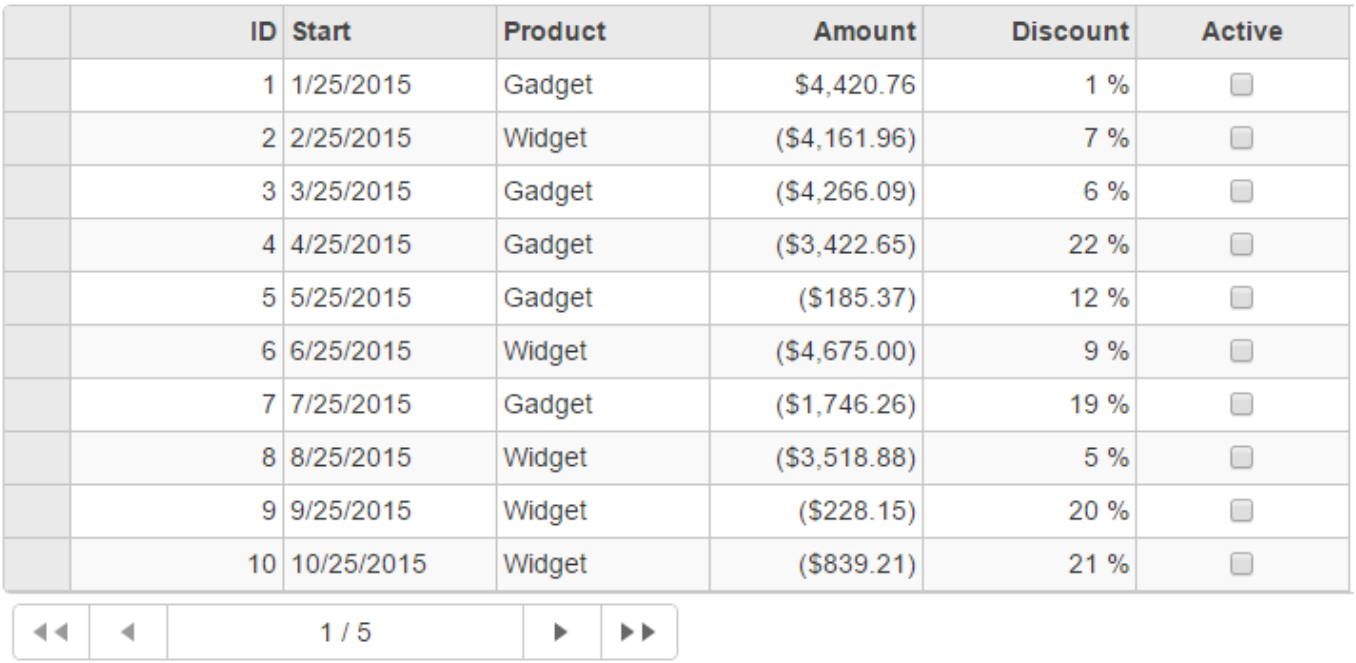
FlexGrid supports **Pager** control through which you can implement paging. Through paging, you can customize the number of items that should be displayed per page and provide a UI for navigating the pages in the grid.

To implement paging in FlexGrid using **Pager** control, you need to

- set the **Id** of the FlexGrid,
- set the **Owner** property of Pager control, and
- set the **PageSize** property.

By setting the **Owner** property of Pager control, the control binds to the FlexGrid and provides function to change the current page of the FlexGrid by clicking these buttons - '<<', '<', '>', '>>'. By setting the PageSize property, the specified number of items you want on each page are displayed.

The following image shows how the FlexGrid appears after setting the **PageSize** property. The example uses Sale.cs model, which was added to the application in the Quick Start:



	ID	Start	Product	Amount	Discount	Active
	1	1/25/2015	Gadget	$4,420.76	1 %	☐
	2	2/25/2015	Widget	($4,161.96)	7 %	☐
	3	3/25/2015	Gadget	($4,266.09)	6 %	☐
	4	4/25/2015	Gadget	($3,422.65)	22 %	☐
	5	5/25/2015	Gadget	($185.37)	12 %	☐
	6	6/25/2015	Widget	($4,675.00)	9 %	☐
	7	7/25/2015	Gadget	($1,746.26)	19 %	☐
	8	8/25/2015	Widget	($3,518.88)	5 %	☐
	9	9/25/2015	Widget	($228.15)	20 %	☐
	10	10/25/2015	Widget	($839.21)	21 %	☐

◀◀ ◀ 1 / 5 ▶ ▶▶

The following code examples demonstrate how to enable Paging using Pager control in FlexGrid.

In Code

HTML Helpers

Razor copyCode

```
@using C1MvcWebAppPager.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

@* Set the ID for FlexGrid *@
@(Html.C1().FlexGrid<Sale>()
    .Id("pagerFlexGrid")
    .AutoGenerateColumns(false)
    .IsReadOnly(true)
    .Height(450)
    .Width(700)
    .AllowAddNew(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .CssClass("grid")
    .Bind(Model)
    .PageSize(10)
    .Columns(bl =>
```

```
        {
            bl.Add(cb => cb.Binding("ID"));
            bl.Add(cb => cb.Binding("Start"));
            bl.Add(cb => cb.Binding("Product"));
            bl.Add(cb => cb.Binding("Amount").Format("c"));
            bl.Add(cb => cb.Binding("Discount").Format("p0"));
            bl.Add(cb => cb.Binding("Active"));
        })
)
@* Set the Owner property of Pager  *@
@(Html.C1().Pager().Owner("pagerFlexGrid"))
```

Tag Helpers

HTML

```
@using C1MvcWebAppPager.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>
@* Set the ID for FlexGrid *@
<c1-flex-grid id="pagerFlexGrid" auto-generate-columns="false" class="grid" is-
read-only="true" width="700px" height="450px">

    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount"> format="p0"</c1-flex-grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>
    <c1-items-source-collection="Model" page-size="10"></c1-items-source>
</c1-flex-grid>
@* Set the Owner property of Pager  *@
<c1-pager owner="pagerFlexGrid" disabled="false" style="width:600px"></c1-pager>
```

Merging

FlexGrid supports merging of cells that have the same content. To enable cell merging, set the AllowMerging property to indicate what part or parts of the grid you want to merge. You can set the AllowMerging property on specific row and column objects. Possible values are None, Cells, ColumnHeaders, RowHeaders, AllHeaders and All.

Once you have set the AllowMerging property to one of these possible values, the FlexGrid automatically merges cells, grouping the data visually.

The following image shows how the FlexGrid appears after setting the AllowMerging property to Cells. The example uses Sale.cs model, which was added to the application in the QuickStart:

	ID	Country	Product	Color	Amount	Discount
	1	China	Gadget	Black	$1,341.85	8 %
	2	US	Widget		$3,596.33	18 %
	3	China		Green	$3,232.11	24 %
	4	France	Gadget		($2,008.99)	9 %
	5	France		White	$2,568.01	3 %
	6	UK	Widget	Black	($3,476.95)	1 %
	7	UK		Red	$2,290.56	6 %
	8	Japan	Gadget	White	($4,146.76)	1 %
	9	Canada		Red	$4,917.55	14 %
	10	Korea		Black	$3,824.28	8 %
	11	Japan		White	($4,257.83)	23 %
	12	Italy		Black	$1,095.08	20 %
	13	German		Green	$1,853.66	18 %
	14	Italy	Widget	White	$3,708.86	3 %
	15	Canada	Gadget	Green	($3,447.73)	10 %

The following code examples demonstrate how to enable Merging in the FlexGrid:

In Code

MergingController.cs

C# copyCode

```
public ActionResult MergeCells()
{
    return View(Sales.GetData(15));
}
```

Merging.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

@(Html.C1().FlexGrid<Sale>()
    .AllowMerging(C1.Web.Mvc.Grid.AllowMerging.Cells)
    .AutoGenerateColumns(false)
    .IsReadOnly(true)
```

```
    .Bind(Model)
    .CssClass("grid")
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("ID").Width("80"));
        columns.Add(column => column.Binding("Country"));
        columns.Add(column => column.Binding("Product").AllowMerging(true));
        columns.Add(column => column.Binding("Color").AllowMerging(true));
        columns.Add(column => column.Binding("Amount").Format("c"));
        columns.Add(column => column.Binding("Discount").Width("100").Format("p0"));
    })
)
```

Tag Helpers

HTML copyCode

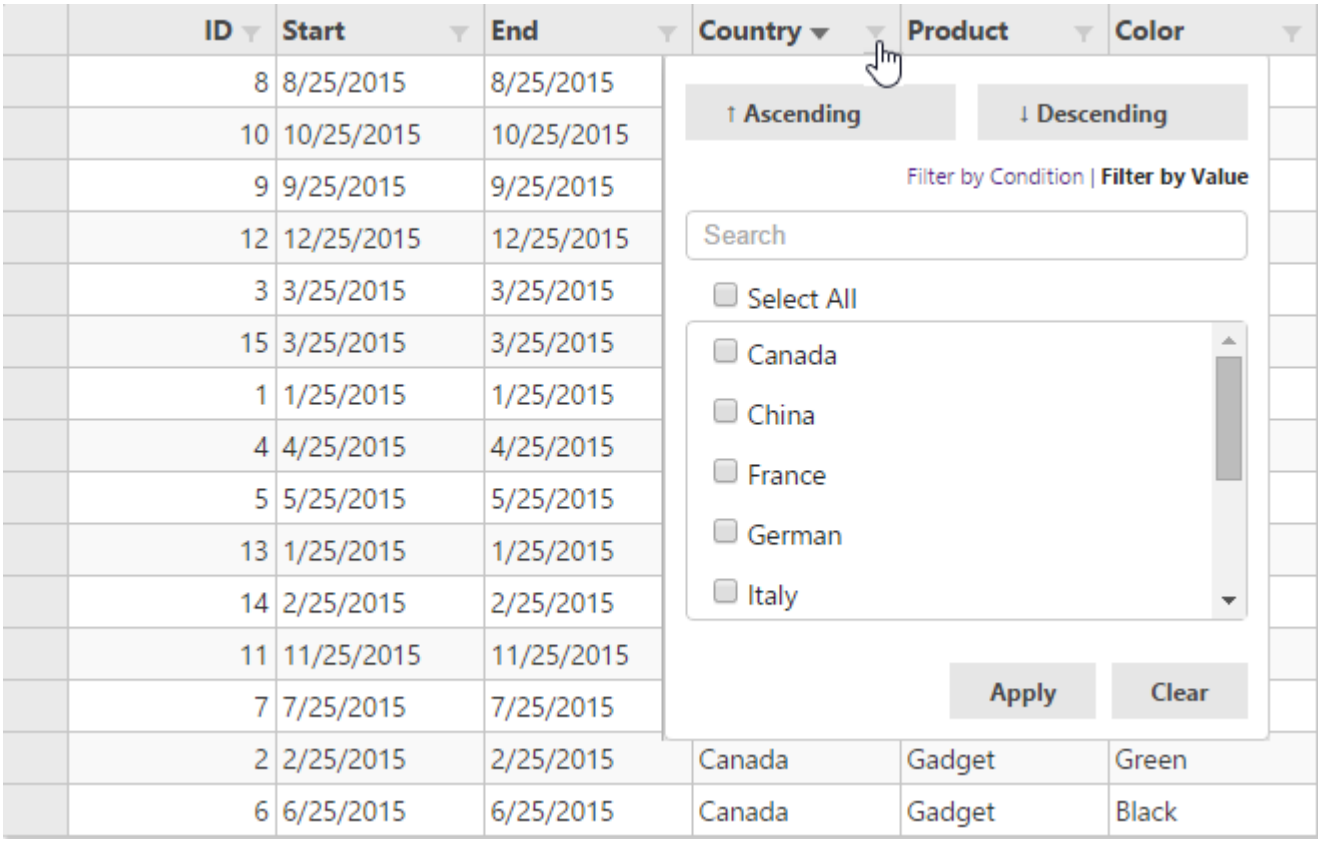
```
@using TagFlexGrid.Models
@using C1.Web.Mvc.Grid

@model IEnumerable<Sale>


<c1-flex-grid auto-generate-columns="false" allow-merging="AllowMerging.Cells"
              is-read-only="true" class="grid">
    <c1-flex-grid-column binding="ID" width="80"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" allow-merging="true"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Color" allow-merging="true"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" width="100" format="p0"></c1-flex-grid-
column>
    <c1-items-source read-action-url="@Url.Action("MergeCells_Bind")"></c1-items-
source>
</c1-flex-grid>
```

Filtering

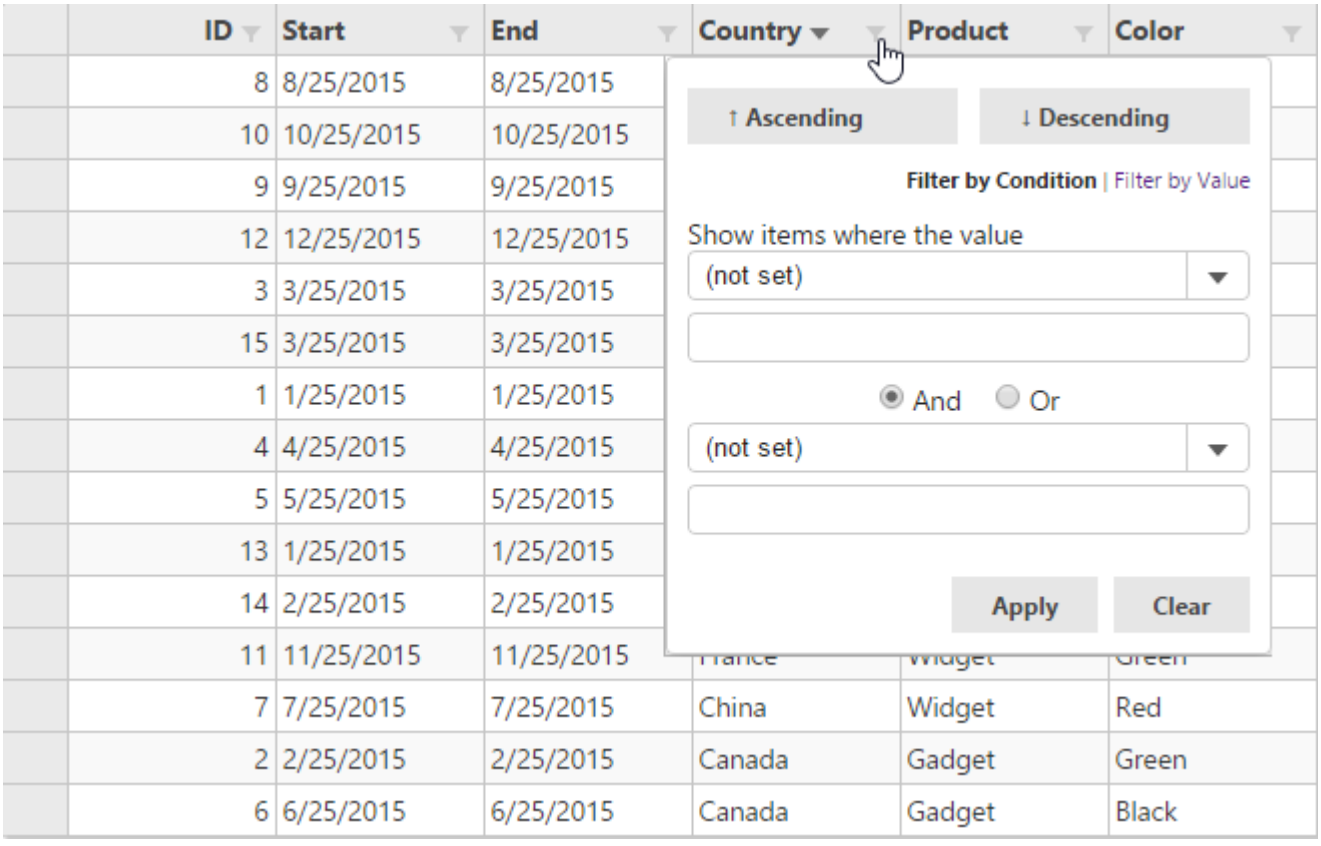
FlexGrid supports **filtering** for column entries, which is enabled through ICollectionView interface. It allows you to apply **Condition filters** and **Value filters** to the grid. This makes it easy for you to fetch the desired entries from the data in your grid.

Use **Filter by Condition** to apply conditions to narrow down your search, and **Filter by Value** to precisely locate data corresponding to the desired column value. **Ascending** and **Descending** buttons enable you to sort the entries in a particular column in ascending and descending order respectively.

The following image shows how the FlexGrid appears after applying **Filter by Value** to different columns. The example uses **Sale.cs** model added in the Quick Start section. Here, you create filters for different columns in the grid, and specify conditions or provide exact values to refine your search. Click **Apply** to fetch the filtered values.



The following image shows how the FlexGrid appears on applying **Filter by Condition**.



The following code examples demonstrate how to enable Filtering in the FlexGrid:

In Code

FilterController.cs

C# copyCode

```
public ActionResult Filter()
{
    return View(Sales.GetData(15));
}
```

Filter.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>

@(Html.C1().FlexGrid<Sale>()
    .Id("filteringGrid")
    .AutoGenerateColumns(false)
    .Bind(Model)
    .IsReadOnly(true)
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("ID"));
        columns.Add(column => column.Binding("Start"));
        columns.Add(column => column.Binding("End").Format("t"));
        columns.Add(column => column.Binding("Country"));
        columns.Add(column => column.Binding("Product"));
        columns.Add(column => column.Binding("Color"));
    })
    .Filterable(f => f.DefaultFilterType(C1.Web.Mvc.FilterType.Both))
    .CssClass("grid")
)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@using C1.Web.Mvc
@using C1.Web.Mvc.Grid

@model IEnumerable<Sale>

<c1-flex-grid id="filteringGrid" auto-generate-columns="false" is-read-only="true"
              selection-mode="SelectionMode.Row" allow-sorting="true" class="grid">
    <c1-items-source page-size="25" read-action-url="@Url.Action("Filter_Bind")">
</c1-items-source>
```

```
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="End" format="t"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Color"></c1-flex-grid-column>

    <c1-flex-grid-filter default-filter-type="FilterType.Both"></c1-flex-grid-filter>
</c1-flex-grid>
```

Virtual Scrolling

FlexGrid provides support for virtual scrolling while working with voluminous data. You can easily bind the FlexGrid with large data sets using models or other data sources, and experience a smooth scrolling without any flicker or delay. To make a grid work in virtual scrolling mode, the DisableServerRead property should be set to false(default). The value of InitialItemsCount property should be set to a number greater than 0.

The following image shows how the FlexGrid appears on binding it to large data set. The example uses Sale.cs model added in the QuickStart section.

ID	Start	End	Country	Product	Color	Amount	Amount2	Discount	Activ
1	1/25/2015	1/25/2015	Canada	Widget	White	1,722.93	3,354.07	0.03	
2	2/25/2015	2/25/2015	Canada	Gadget	Red	4,521.50	-1,837.39	0.10	
3	3/25/2015	3/25/2015	UK	Widget	Green	-724.43	-1,414.05	0.20	
4	4/25/2015	4/25/2015	US	Widget	Black	-1,623.23	4,445.76	0.20	
5	5/25/2015	5/25/2015	Canada	Widget	Black	-1,801.06	2,689.10	0.19	
6	6/25/2015	6/25/2015	Korea	Gadget	Black	-3,762.91	1,643.74	0.18	
7	7/25/2015	7/25/2015	China	Gadget	Green	4,191.53	2,849.26	0.14	
8	8/25/2015	8/25/2015	Japan	Gadget	Black	1,167.17	-77.58	0.11	
9	9/25/2015	9/25/2015	China	Gadget	Black	1,026.92	-1,191.80	0.25	
10	10/25/2015	10/25/2015	Japan	Widget	Red	4,470.48	2,944.65	0.03	
11	11/25/2015	11/25/2015	German	Gadget	Green	-4,400.30	-4,500.45	0.02	
12	12/25/2015	12/25/2015	Canada	Widget	Black	-1,856.35	4,483.26	0.16	
13	1/25/2015	1/25/2015	German	Gadget	Green	4,255.27	1,631.06	0.10	
14	2/25/2015	2/25/2015	US	Widget	Green	-4,451.57	4,170.97	0.19	
15	3/25/2015	3/25/2015	Japan	Gadget	Green	-258.83	-3,215.07	0.17	
16	4/25/2015	4/25/2015	Korea	Widget	White	1,347.56	2,000.48	0.22	

The following code examples demonstrate how to enable Virtual Scrolling in the FlexGrid:

In Code

VirtualScrollingController.cs

C# copyCode

```
public ActionResult VirtualScrolling()
{
    return View(Sales.GetData(1000));
}
```

VirtualScrolling.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models

@(Html.C1().FlexGrid<Sale>()
    .Bind(bl => bl.InitialItemsCount(100).Bind(Model))
    .Width(1000)
    .Height(500)
    .CssClass("grid")
)
```

Tag Helpers

HTML copyCode

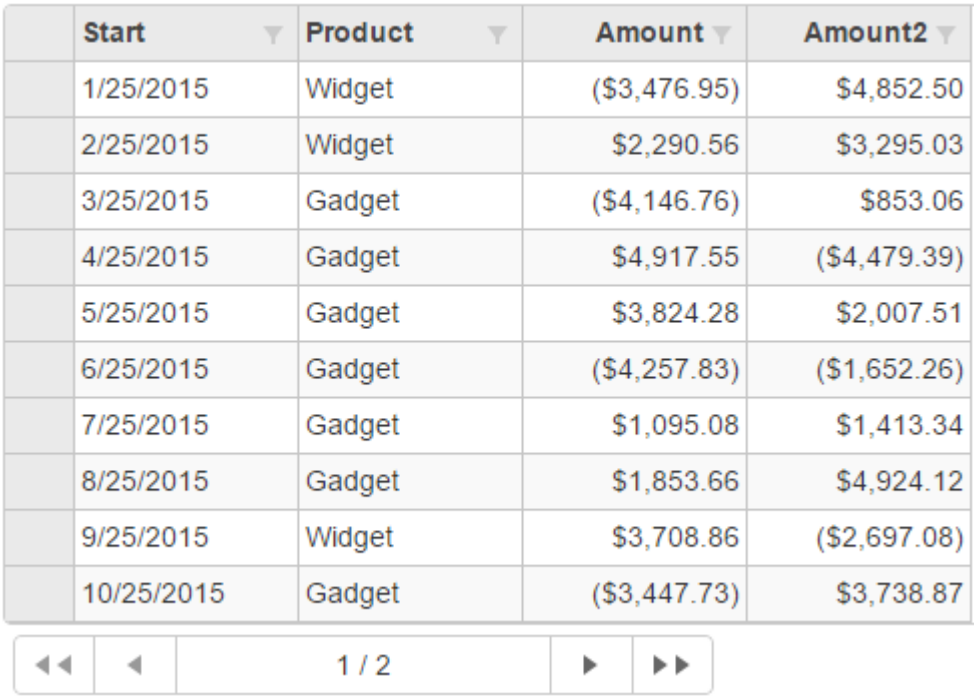
```
<c1-flex-grid is-read-only="true" class="grid"
height="500px" width="1000px" >
    <c1-items-source initial-items-count="100"
read-action-url="@Url.Action("VirtualScrolling_Bind")" >
</c1-items-source>
</c1-flex-grid>
```

Disable Server Reading

FlexGrid allows you to disable server side sorting, paging, filtering or scrolling by setting the value of DisableServerRead property to True. By default, its value is set to False. On setting the property value to true for any operation, all the current items are transferred to the client side and the server side events get disabled. You can use carry out client side operations directly without making any network calls.

ASP.NET Core doesn't provide support for CallBack. When using the ActionUrl, you can enable or disable the DisableServerRead in the ASP.NET Core FlexGrid control. In case a user binds the model directly using the bind property in ASP.NET Core, the DisableServerRead property is automatically set to true, transferring all the data from server-side to client-side.

The following image shows how the FlexGrid appears on setting the **DisableServerRead** property to set the page size to **10**. The example uses Sale.cs model added in the QuickStart section.



	Start	Product	Amount	Amount2
	1/25/2015	Widget	($3,476.95)	$4,852.50
	2/25/2015	Widget	$2,290.56	$3,295.03
	3/25/2015	Gadget	($4,146.76)	$853.06
	4/25/2015	Gadget	$4,917.55	($4,479.39)
	5/25/2015	Gadget	$3,824.28	$2,007.51
	6/25/2015	Gadget	($4,257.83)	($1,652.26)
	7/25/2015	Gadget	$1,095.08	$1,413.34
	8/25/2015	Gadget	$1,853.66	$4,924.12
	9/25/2015	Widget	$3,708.86	($2,697.08)
	10/25/2015	Gadget	($3,447.73)	$3,738.87

The following code examples demonstrate how to disable server reading in the FlexGrid. The sample uses the `_Pager.cshtml` page added in the Paging topic.

In Code

DisableServerReadController.cs

C# copyCode

```
public ActionResult DisableServerRead(FormCollection collection)
{
    IValueProvider data = collection;
    if (CallbackManager.CurrentIsCallback)
    {
        var request =
CallbackManager.GetCurrentCallbackData<CollectionViewRequest<object>>();
        if (request != null && request.ExtraRequestData != null)
        {
            var extraData = request.ExtraRequestData.Cast<DictionaryEntry>()
                .ToDictionary(kvp => (string)kvp.Key, kvp => kvp.Value.ToString());
            data = new DictionaryValueProvider<string>(extraData,
CultureInfo.CurrentCulture);
        }
    }
    return View(Sales.GetData(20));
}
```

DisableServerRead.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models

@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
      .Columns(columns =>
        {
        columns.Add(column => column.Binding("Start"));
        columns.Add(column => column.Binding("Product").AllowResizing(false));
        columns.Add(column => column.Binding("Amount").Format("c"));
        columns.Add(column => column.Binding("Amount2").Format("c"));
    })
    .CssStyle("height", "auto")
    .Id("dsrPagingGrid")
    .CssClass("grid")
    .IsReadOnly(true)
    .Filterable<Sale>()
    .Bind(b => b.DisableServerRead(true)
    .PageSize(10)
    .Bind(Model))
)

@Html.Partial("_Pager", "dsrPagingGrid")
```

Tag Helpers

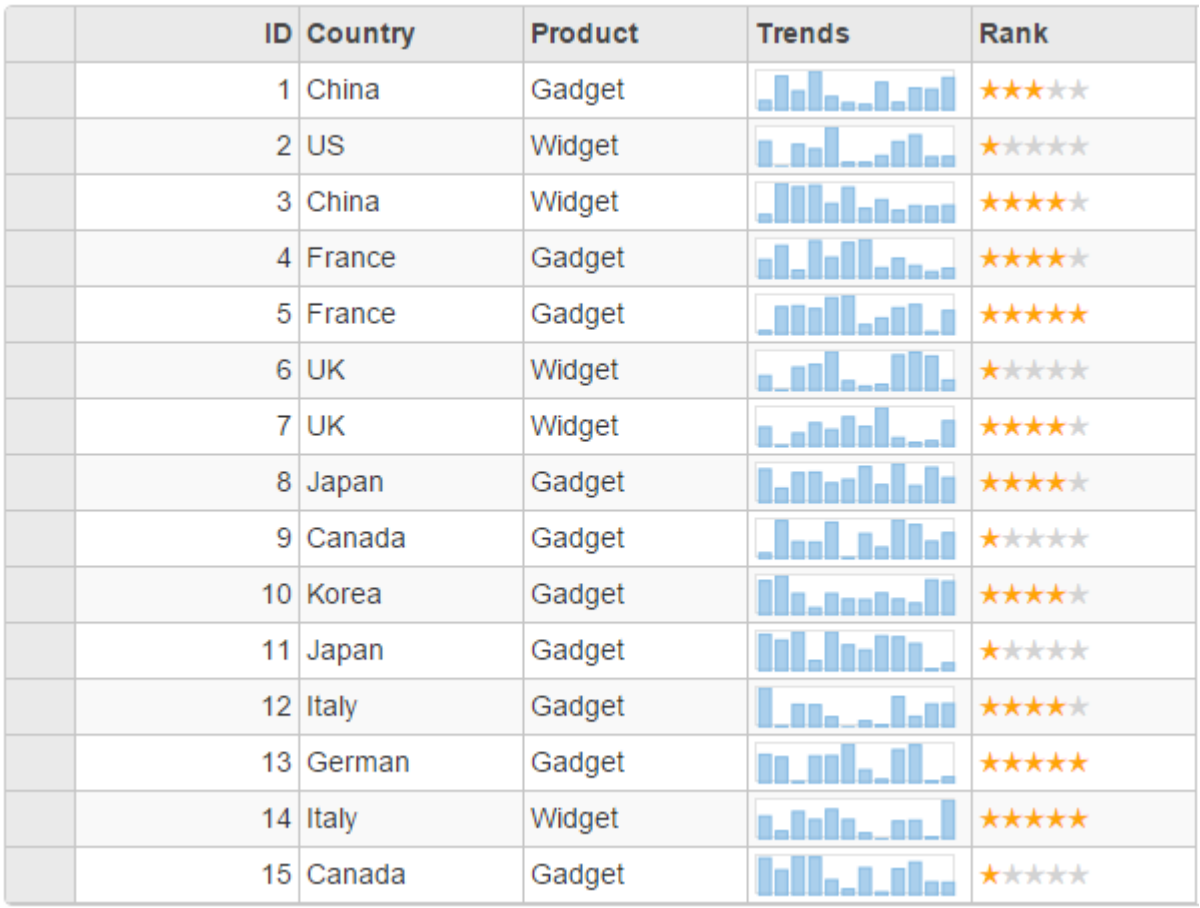
HTML copyCode

```
@using TagFlexGrid.Models

<c1-flex-grid id="dsrPagingGrid" auto-generate-columns="false" height="300px"
              style="height:auto" class="grid" is-read-only="true" >
    <c1-flex-grid-filter></c1-flex-grid-filter>
    <c1-items-source disable-server-read="true"
                     page-size="10"
                     read-action-url="@Url.Action("DisableServerRead_Bind")"></c1-
items-source>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount2" format="c"></c1-flex-grid-column>
</c1-flex-grid>
@Html.Partial("_Pager", "dsrPagingGrid")
```

Custom Cell Template

FlexGrid has an itemFormatter property that gives you complete control over the contents of the cells. MVC Edition provides CellTemplate to let users customize the cell content in FlexGrid control. To define a cell template for a column, add the HTML to display in each cell to the column definition.

The following image shows how the FlexGrid appears on setting **FlexChart** as a cell template to represent Trends, and **Stars** as another template for representing Rank. The example uses Sale.cs model added in the QuickStart section.



	ID	Country	Product	Trends	Rank
	1	China	Gadget		★★★☆☆
	2	US	Widget		★☆☆☆☆
	3	China	Widget		★★★★☆
	4	France	Gadget		★★★★☆
	5	France	Gadget		★★★★★
	6	UK	Widget		★☆☆☆☆
	7	UK	Widget		★★★★☆
	8	Japan	Gadget		★★★★☆
	9	Canada	Gadget		★☆☆☆☆
	10	Korea	Gadget		★★★★☆
	11	Japan	Gadget		★☆☆☆☆
	12	Italy	Gadget		★★★★☆
	13	German	Gadget		★★★★★
	14	Italy	Widget		★★★★★
	15	Canada	Gadget		★☆☆☆☆

The following code examples demonstrate how to set the custom cell templates in a column of the FlexGrid:

In Code

CustomCellsController.cs

C# copyCode

```
public ActionResult CustomCells()
{
    return View(Sales.GetData(15));
}
```

CustomCells.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@model IEnumerable<Sale>

<script id="template1" type="text/html">
```

```
    @(Html.C1().FlexChart()
        .Width(100).Height(20).CssStyle("padding", "0")
        .TemplateBind("ItemsSource", "Trends")
        .BindingX("Month")
        .Series(s => s.Add().Binding("Data"))
        .AxisX(C1.Web.Mvc.Chart.Position.None)
        .AxisY(C1.Web.Mvc.Chart.Position.None)
        .ToTemplate()
    )
</script>

<style>
    .star-highlighted {
        color: orange;
    }

    .star-dimmed {
        color: lightgray;
    }
</style>

<script type="text/javascript">
    function rankFormatter(panel, r, c, cell) {
        if (panel.columns[c].binding === "Rank") {
            cell.style.textAlign = "";
            if (panel.cellType === wijmo.grid.CellType.Cell) {
                cell.innerHTML = buildRank(panel.getCellData(r, c));
            }
        }
    }

    function buildRank(rank) {
        var markup = "", j, starType;
        for (j = 0; j < 5; j++) {
            starType = j < rank ? "star star-highlighted" : "star star-dimmed";
            markup += "<span class='" + starType + "'>\u2605</span>";
        }
        return markup;
    }
</script>

@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .IsReadOnly(true)
    .Bind(Model)
    .CssClass("grid")
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("ID"));
        columns.Add(column => column.Binding("Country"));
        columns.Add(column => column.Binding("Product"));
```

```
        columns.Add(column => column.Header("Trends").CellTemplate(b =>
b.Template("template1")));
        columns.Add(column => column.Binding("Rank"));
    })
    .ItemFormatter("rankFormatter")
)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@using C1.Web.Mvc.Grid
@using C1.Web.Mvc.Fluent
@model IEnumerable<Sale>

<style>
    .star-highlighted {
        color: orange;
    }

    .star-dimmed {
        color: lightgray;
    }
</style>

<script type="text/javascript">
    function rankFormatter(panel, r, c, cell) {
        if (panel.columns[c].binding === "Rank") {
            cell.style.textAlign = "";
            if (panel.cellType === wijmo.grid.CellType.Cell) {
                cell.innerHTML = buildRank(panel.getCellData(r, c));
            }
        }
    }

    function buildRank(rank) {
        var markup = "", j, starType;
        for (j = 0; j < 5; j++) {
            starType = j < rank ? "star star-highlighted" : "star star-dimmed";
            markup += "<span class='" + starType + "'>\u2605</span>";
        }
        return markup;
    }
</script>
<c1-flex-grid id="customCellFlexGrid" auto-generate-columns="false" is-read-
only="true"
              class="grid" item-formatter="rankFormatter">
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
```

```
    <c1-flex-grid-column binding="Trends">
        <c1-flex-grid-cell-template>
            <c1-flex-chart width="100px" height="20px" style="padding:0px"
                           binding-x="Month" template-bindings="@(new
{ItemsSource="Trends"})">
                <c1-flex-chart-axis c1-property="AxisX"
position="C1.Web.Mvc.Chart.Position.None"></c1-flex-chart-axis>
                <c1-flex-chart-axis c1-property="AxisY"
position="C1.Web.Mvc.Chart.Position.None"></c1-flex-chart-axis>
                <c1-flex-chart-series binding="Data"></c1-flex-chart-series>
            </c1-flex-chart>
        </c1-flex-grid-cell-template>
    </c1-flex-grid-column>
    <c1-flex-grid-column binding="Rank"></c1-flex-grid-column>
    <c1-items-source read-action-url="@Url.Action("CustomCells_Bind")"></c1-items-
source>
</c1-flex-grid>
```

Right To Left Rendering

FlexGrid has the in-built functionality of rendering the content from the right to the left of the page required in languages such as Arabic and Hebrew. While HTML supports this feature with the 'dir' attribute, FlexGrid performs this automatically on setting 'dir' to 'rtl' on any element without setting any properties on the control. The same can also be achieved by using the 'direction' attribute of CSS.

The '**dir**' attribute value is inherited, so if you set this attribute on the body tag, the entire page including the grid will be rendered from right to left.

Active	Product	Country	End	Start	ID
☐	Widget	UK	1/25/2015	1/25/2015	1
☐	Widget	UK	2/25/2015	2/25/2015	2
☐	Gadget	Japan	3/25/2015	3/25/2015	3
☐	Gadget	Canada	4/25/2015	4/25/2015	4
☐	Gadget	Korea	5/25/2015	5/25/2015	5
☐	Gadget	Japan	6/25/2015	6/25/2015	6
☐	Gadget	Italy	7/25/2015	7/25/2015	7
☐	Gadget	German	8/25/2015	8/25/2015	8
☐	Widget	Italy	9/25/2015	9/25/2015	9
☐	Gadget	Canada	10/25/2015	10/25/2015	10
☐	Gadget	German	11/25/2015	11/25/2015	11
☐	Widget	China	12/25/2015	12/25/2015	12
☐	Widget	Korea	1/25/2015	1/25/2015	13
☐	Widget	Italy	2/25/2015	2/25/2015	14
☐	Gadget	Korea	3/25/2015	3/25/2015	15
☐	Widget	Canada	4/25/2015	4/25/2015	16
☐	Gadget	US	5/25/2015	5/25/2015	17
☐	Widget	Canada	6/25/2015	6/25/2015	18
☐	Gadget	Japan	7/25/2015	7/25/2015	19
☐	Widget	US	8/25/2015	8/25/2015	20

The following code examples demonstrate how to set the 'dir' attribute to show right to left rendering in the FlexGrid:

In Code

RightToLeftController.cs

C# copyCode

```
public ActionResult RightToLeft()
{
    return View(Sales.GetData(10));
}
```

RightToLeft.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models

@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
```

```
    .Bind(bl => bl.Bind(Model))
    .Width(620)
    .CssClass("grid")
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ID"));
        bl.Add(cb => cb.Binding("Start"));
        bl.Add(cb => cb.Binding("End"));
        bl.Add(cb => cb.Binding("Country"));
        bl.Add(cb => cb.Binding("Product"));
    })
    .HtmlAttribute("dir", "rtl")
)
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models

<c1-flex-grid auto-generate-columns="false" is-read-only="true" class="grid"
              dir="rtl" width="620">
    <c1-items-source read-action-url="@Url.Action("RightToLeft_Bind")"></c1-items-
source>
    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start"></c1-flex-grid-column>
    <c1-flex-grid-column binding="End"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
</c1-flex-grid>
```

Selection Modes

By default, FlexGrid allows you to select a range of cells with the mouse or keyboard, similar to Excel. The SelectionMode property allows you to change that so that you can select a Row, a Range of Rows, Non-Contiguous Rows (like in a List-Box), a Single Cell, a Range of Cells or disable selection altogether.

To specify the selection behaviour in FlexGrid, set the SelectionMode to:

- **Cell**: The SelectionMode when set to `Cell`, lets a user select only a single cell at a time.
- **CellRange**: The SelectionMode when set to `CellRange`, lets a user select contiguous blocks of cells.
- **ListBox**: The SelectionMode when set to `ListBox`, lets a user select non-contiguous rows.
- **None**: The SelectionMode when set to `None`, doesn't allow user to select cells with the mouse or keyboard.
- **Row**: The SelectionMode when set to `Row`, lets a user select a single row at a time.
- **RowRange**: The SelectionMode when set to `RowRange`, lets a user select contiguous rows.

The following image shows how the FlexGrid appears on setting SelectionMode to **Row**. The example uses Sale.cs model added in the QuickStart section.

	ID	Start	End	Country	Product	Color
	1	1/25/2015	1/25/2015	German	Gadget	Green
	2	2/25/2015	2/25/2015	Canada	Gadget	Green
	3	3/25/2015	3/25/2015	Japan	Gadget	Red
	4	4/25/2015	4/25/2015	German	Gadget	Red
	5	5/25/2015	5/25/2015	German	Gadget	Black
	6	6/25/2015	6/25/2015	Canada	Gadget	Black
	7	7/25/2015	7/25/2015	China	Widget	Red
	8	8/25/2015	8/25/2015	US	Widget	White
	9	9/25/2015	9/25/2015	Korea	Gadget	Black
	10	10/25/2015	10/25/2015	US	Widget	White

The following code examples demonstrate how to set the selection mode in FlexGrid.

In Code

SelectionMode.cshtml

HTML Helpers

Razor

```
.SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
```

Tag Helpers

HTML

```
selection-mode="@((SelectionMode)Enum.Parse(typeof(SelectionMode),
optionsModel.Options["Selection"].CurrentValue))"
```

Data Map

FlexGrid supports data mapping, which provides automatic look up capabilities to the grid. It is helpful when your grid displays data from a database, but you are willing to change some values, visible to the customer, without altering the underlying data structure. For example, you might want to display product names in Product column instead of ProductID or color name in Color column instead of Hexadecimal code in Color column of your grid.

You can also use the DropDownCssClass property, that allows you to get or set the CSS class name to the dropdowns in the column. This property is applicable on the dropdown buttons only if the column has a Data Map and the editable mode is set.

The following image shows how a FlexGrid appears after assigning DataMap (using DisplayMemberPath and SelectedValuePath properties) to Product and Color columns of the FlexGrid control.

	ID	Start	End	Country	Product	Color	Amount
	1	Jan 25 15	00:00	German	Gadget	Green	$581.61
	2	Feb 25 15	01:01	Canada	Widget	Green	$4,919.02
	3	Mar 25 15	02:02	Japan	Widget	Red	$2,159.73
	4	Apr 25 15	03:03	German	Gadget	Red	$1,248.66
	5	May 25 15	04:04	German	Doohickey	Black	$4,051.76
	6	Jun 25 15	05:05	Canada	Gadget	Black	($3,131.28)
	7	Jul 25 15	06:06	China	Widget	Red	$698.62
	8	Aug 25 15	07:07	US	Widget	White	$3,464.15
	9	Sep 25 15	08:08	Korea	Gadget	Black	($2,363.16)
	10	Oct 25 15	09:09	US	Widget	White	($2,836.94)
	11	Nov 25 15	10:10	France	Widget	Green	$877.93
	12	Dec 25 15	11:11	Korea	Widget	Red	($3,788.14)
	13	Jan 25 15	12:12	German	Gadget	Red	($2,446.92)
	14	Feb 25 15	13:13	German	Widget	Black	($4,374.97)
	15	Mar 25 15	14:14	Japan	Gadget	Green	$1,089.32

If you do not assign DataMap to Product and Color columns in the grid, these columns would display corresponding product ids and Hexadecimal color codes from the datasource.

The following code examples demonstrate how to create data mapping in a FlexGrid. The example uses **Sale.cs** model added in the QuickStart section, and **CustomerSale.cs** model.

In Code

CustomerSale.cs

C# copyCode

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

namespace MvcFlexGrid.Models
{
    public class CustomerSale
    {
        public static List<string> COUNTRIES = new List<string> { "US", "UK",
"Canada", "Japan", "China", "France", "German", "Italy", "Korea", "Australia" };
        public static List<NamedProduct> PRODUCTS = new List<NamedProduct> {
            new NamedProduct {Id = "1", Name = "Widget"},
            new NamedProduct {Id = "2", Name = "Gadget"},
            new NamedProduct {Id = "3", Name = "Doohickey"}
        };

        public static List<NamedColor> COLORS = new List<NamedColor> {
            new NamedColor {Name = "Black", Value = "#000000"},
            new NamedColor {Name = "White", Value = "#FFFFFF"},
```

```
            new NamedColor {Name = "Red", Value = "#FF0000"},
            new NamedColor {Name = "Green", Value = "#00FF00"},
            new NamedColor {Name = "Blue", Value = "#0000FF"}
        };

        public static IEnumerable<Sale> GetData(int total)
        {
            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)].Id;
                var color = COLORS[rand.Next(0, COLORS.Count - 1)].Value;
                var startDate = new DateTime(dt.Year, i % 12 + 1, 25);
                var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60, i
% 60);

                return new Sale
                {
                    ID = i + 1,
                    Start = startDate,
                    End = endDate,
                    Country = country,
                    Product = product,
                    Color = color,
                    Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Discount = Math.Round(rand.NextDouble() / 4, 2),
                    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;
        }

        public class NamedProduct
        {
            public string Id { get; set; }

            public string Name { get; set; }
        }
        public class NamedColor
        {
            public string Name { get; set; }
            public string Value { get; set; }
        }
    }
}
```

DataMapController.cs

C# copyCode

```
public partial class DataMapController : Controller
{
    public static List<Sale> SALES = CustomerSale.GetData(15).ToList();
    public ActionResult Index()
    {
        ViewBag.Products = CustomerSale.PRODUCTS;
        ViewBag.Colors = CustomerSale.COLORS;
        return View(SALES);
    }
    public ActionResult ProductsUpdate([C1JsonRequest]CollectionViewEditRequest<Sale>
requestData)
    {
        return this.C1Json(CollectionViewHelper.Edit<Sale>(requestData, sale =>
        {
            string error = string.Empty;
            bool success = true;
            var fSale = SALES.Find(item => item.ID == sale.ID);
            fSale.Active = sale.Active;
            fSale.Amount = sale.Amount;
            fSale.Color = sale.Color;
            fSale.Country = sale.Country;
            fSale.Discount = sale.Discount;
            fSale.End = sale.End;
            fSale.Amount2 = sale.Amount2;
            fSale.Start = sale.Start;
            fSale.Product = sale.Product;
            return new CollectionViewItemResult<Sale>
            {
                Error = error,
                Success = success && ModelState.IsValid,
                Data = fSale
            };
        }, () => SALES));
    }
}
```

Make sure to include the following references to the controller.

C#

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using <ApplicationName>.Models;
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
```

DataMap.cshtml

HTML Helpers

Razor copyCode

```
@using <ApplicationName>.Models
@model IEnumerable<Sale>
@{
    var products = ViewBag.Products;
    var colors = ViewBag.Colors;
}

@(Html.C1().FlexGrid<Sale>()
    .Id("ovFlexGrid")
    .AutoGenerateColumns(false)
    .Bind(bl =>
bl.InitialItemsCount(15).Bind(Model).Update(Url.Action("ProductsUpdate")))
    .CssClass("grid")
    .IsReadOnly(false)
    .Columns(bl =>
    {
        bl.Add(cb => cb.Binding("ID"));
        bl.Add(cb => cb.Binding("Start").Format("MMM d yy"));
        bl.Add(cb => cb.Binding("End").Format("HH:mm"));
        bl.Add(cb => cb.Binding("Country"));
        bl.Add(cb => cb.Binding("Product")
            .DataMap(dm => dm.DisplayMemberPath("Name")
                .SelectedValuePath("Id")
                .Bind(products)));
        bl.Add(cb => cb.Binding("Color")
            .DataMap(dm => dm.DisplayMemberPath("Name")
                .SelectedValuePath("Value")
                .Bind(colors)));
        bl.Add(cb => cb.Binding("Amount").Format("c"));
    })
)
```

Tag Helpers

HTML copyCode

```
@model IEnumerable<Sale>
@{
    var products = ViewBag.Products;
    var colors = ViewBag.Colors;
}

<c1-flex-grid id="ovFlexGrid" auto-generate-columns="false" class="grid" is-read-
only="false">
    <c1-flex-grid-column binding="ID" is-visible="true"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start" format="MMM d yy"></c1-flex-grid-column>
```

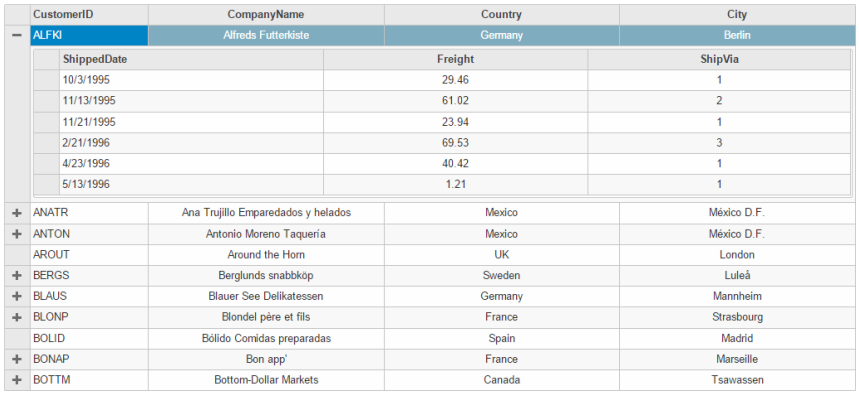
```
    <c1-flex-grid-column binding="End" format="HH:mm"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product">
        <c1-data-map display-member-path="Name" selected-value-path="Id">
            <c1-items-source source-collection="products"></c1-items-source>
        </c1-data-map>
    </c1-flex-grid-column>
    <c1-flex-grid-column binding="Color">
        <c1-data-map display-member-path="Name" selected-value-path="Value">
            <c1-items-source source-collection="colors"></c1-items-source>
        </c1-data-map>
    </c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-items-source source-collection="Model" initial-items-count="15" update-
action-url="ProductsUpdate" ></c1-items-source>
</c1-flex-grid>
```

Detail Row

FlexGrid provides support for detail row functionality, which adds a templated, expandable row to the control. You can add details section to any row in FlexGrid, which enables you to group data in a template that will be visible optionally. This allows end users to view additional data related to a row by simply selecting the row.

Built-in expand/collapse buttons are also provided to control the visibility of data within the expandable row. You can add nested grid within a Detail Row to create a hierarchical grid interface.

The following image shows how a FlexGrid with Detail Row appears. The example uses Customer data from local NorthWind database file **C1NWind.mdf**.



	CustomerID	CompanyName	Country	City
−	ALFKI	Alfreds Futterkiste	Germany	Berlin

ShippedDate	Freight	ShipVia
10/3/1995	29.46	1
11/13/1995	61.02	2
11/21/1995	23.94	1
2/21/1996	69.53	3
4/23/1996	40.42	1
5/13/1996	1.21	1

+	ANATR	Ana Trujillo Emparedados y helados	Mexico	México D.F.
+	ANTON	Antonio Moreno Taquería	Mexico	México D.F.
	AROUT	Around the Horn	UK	London
+	BERGS	Berglunds snabbköp	Sweden	Luleå
+	BLAUS	Blauer See Delikatessen	Germany	Mannheim
+	BLONP	Blondel père et fils	France	Strasbourg
	BOLID	Bólido Comidas preparadas	Spain	Madrid
+	BONAP	Bon app'	France	Marseille
+	BOTTM	Bottom-Dollar Markets	Canada	Tsawassen

The following code examples demonstrate how to enable Detail Row in a FlexGrid:

In Code

DetailRowController.cs

C# copyCode

```
public partial class FlexGridController : Controller
{
    public ActionResult DetailRow()
    {
        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);
    }
}
```

DetailRow.cshtml

HTML Helpers

Razor copyCode

```
@model IEnumerable<CustomerWithOrders>
<script>
    function hasDetail(row) {
        return row.dataItem.Country.length > 5;
    }
</script>
<script id="detailTemplate" type="text/template">
    @(Html.C1().FlexGrid()
        .Height("200px")
        .AutoGenerateColumns(false)
        .IsReadOnly(true)
        .TemplateBind("ItemsSource", "Orders")
        .Columns(columns =>
        {
            columns.Add(column => column.Binding("ShippedDate").Width("*"));
            columns.Add(column => column.Binding("Freight").Width("*").Align("Center"));
            columns.Add(column => column.Binding("ShipVia").Width("*").Align("Center"));
        })
        .ToTemplate()
    )
</script>

@(Html.C1().FlexGrid()
    .ShowDetailRow(d =>
d.DetailRowTemplateId("detailTemplate").RowHasDetail("hasDetail").DetailVisibilityMode(C1.Web.Mvc.Grid.DetailVisibilityMode.ExpandMulti))
    .Id("detailRowFlexGrid")
    .AutoGenerateColumns(false)
    .IsReadOnly(true)
    .Bind(Model)
    .Columns(columns =>
    {
        columns.Add(column => column.Binding("CustomerID").Width("*"));
        columns.Add(column => column.Binding("CompanyName").Width("2*").Align("Center"));
        columns.Add(column => column.Binding("Country").Width("2*").Align("Center"));
        columns.Add(column => column.Binding("City").Width("2*").Align("Center"));
    })
)
```

Tag Helpers

HTML copyCode

```
@model IEnumerable<CustomerWithOrders>

<script type="text/javascript">
    function hasDetail(row) {
        return row.dataItem.Country.length > 5;
    }
</script>

<c1-flex-grid id="detailRowFlexGrid" auto-generate-columns="false" is-read-only="true">
    <c1-flex-grid-column binding="CustomerID" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="CompanyName" width="2*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Country" width="2*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="City" width="2*"></c1-flex-grid-column>
    <c1-flex-grid-detail detail-visibility-mode="DetailVisibilityMode.ExpandSingle" row-has-detail="hasDetail">
        <c1-flex-grid auto-generate-columns="false" is-read-only="true" height="200px" template-bindings="@(new {ItemsSource="Orders"})">
                <c1-flex-grid-column binding="ShippedDate" width="*"></c1-flex-grid-column>
                <c1-flex-grid-column binding="Freight" width="*"></c1-flex-grid-column>
                <c1-flex-grid-column binding="ShipVia" width="*"></c1-flex-grid-column>
            </c1-flex-grid>
    </c1-flex-grid-detail>
    <c1-items-source source-collection="Model"></c1-items-source>
</c1-flex-grid>
```

Note that public class **CustomerWithOrders** inherits **Customer** class, and can be defined in the model named CustomerWithOrders, as shown below:

Example Title copyCode

```
public class CustomerWithOrders : Customer
{
    public List<Order> Orders { get; set; }
}
```

Star Sizing

You can use **XAML-style** star sizing to implement flexible layouts with the FlexGrid. The sample below uses Star sizing specified in the width property of the object.

This grid has three columns. The first is **80 pixels** wide and can be resized between **40** and **160 pixels**. The other two have widths of **2*** and *, and cannot be resized using the mouse. The width proportions of the columns remain unchanged even on resizing the first column or the whole grid.

The following image shows how the FlexGrid appears on applying star sizing on the columns using the **Width** property. The example uses Sale.cs model added in the QuickStart section.

	Start	Product	Country
	1/25/2015	Widget	France
	2/25/2015	Widget	Korea
	3/25/2015	Gadget	German
	4/25/2015	Widget	German
	5/25/2015	Gadget	Japan
	6/25/2015	Gadget	China
	7/25/2015	Widget	US
	8/25/2015	Widget	China
	9/25/2015	Gadget	France
	10/25/2015	Gadget	France

The following code examples demonstrate how to apply star sizing to the columns of the FlexGrid:

In Code

StarSizingController.cs

C# copyCode

```
public ActionResult StarSizing()
{
    return View(Sales.GetData(10));
}
```

StarSizing.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>


@(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .CssClass("grid")
    .Columns(columns =>
    {
        columns.Add(column =>
```

```
column.Binding("Start").Width("80").MaxWidth(160).MinWidth(40));
        columns.Add(column =>
column.Binding("Product").Width("2*").AllowResizing(false));
        columns.Add(column => column.Binding("Country").Format("c") .Width("*"));
    })
    .Bind(Model)
)
```

Tag Helpers

HTML copyCode

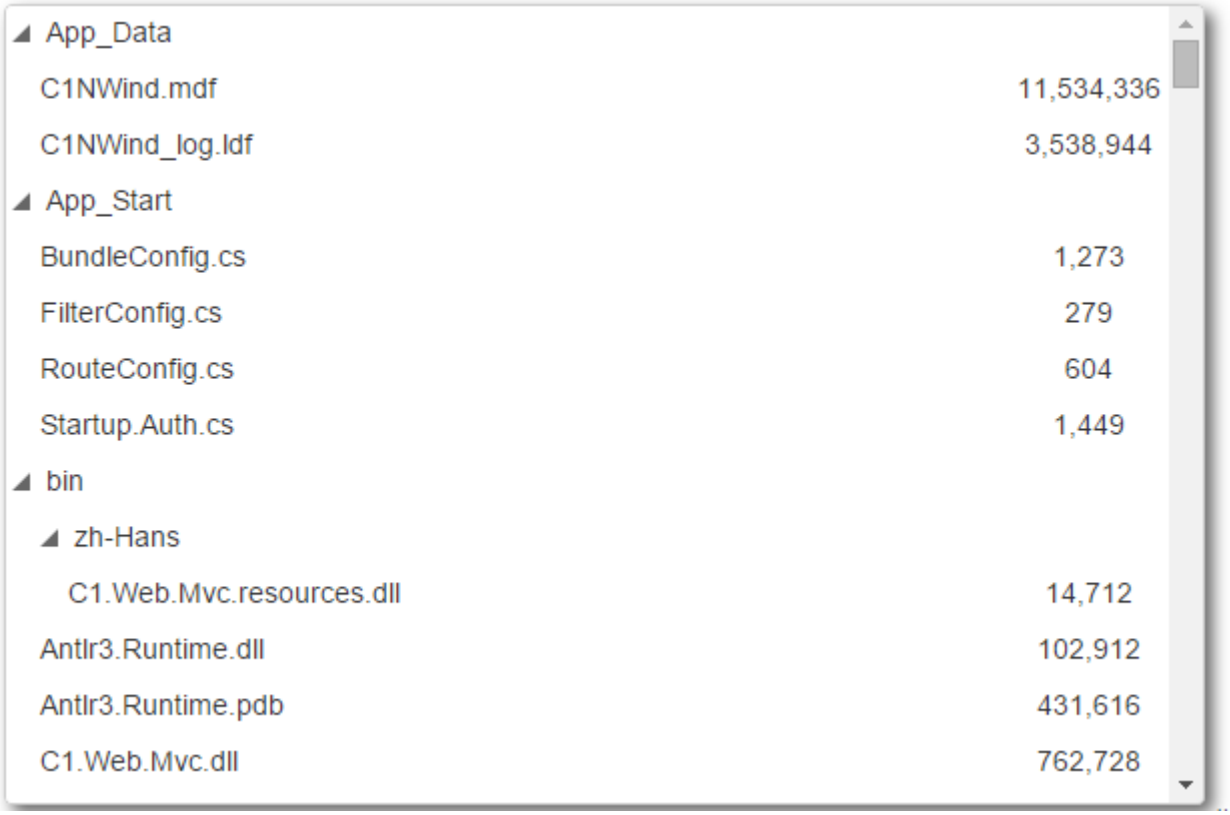
```
@using TagFlexGrid.Models
@using C1.Web.Mvc.Grid
@model IEnumerable<Sale>



<c1-flex-grid auto-generate-columns="false" class="grid" is-read-only="true">
    <c1-flex-grid-column binding="Start" width="80" max-width="160" min-width="40">
</c1-flex-grid-column>
    <c1-flex-grid-column binding="Product" width="2*" allow-resizing="false"></c1-
flex-grid-column>
    <c1-flex-grid-column binding="Country" format="c" width="*"></c1-flex-grid-
column>
    <c1-items-source read-action-url="@Url.Action("StarSizing_Bind")"></c1-items-
source>
</c1-flex-grid>
```

Tree View

FlexGrid supports hierarchical data, that is items that have lists of subitems. To use FlexGrid as a Tree-view with hierarchical data sources, set the ChildItemsPath property to the name of the data element that contains the child elements. The FlexGrid automatically scans the data and builds the tree.

The example below uses a sample folder structure of ASP.NET MVC Edition project. You can display the tree view structure of any folder by providing its path in the ChildItemsPath property.

The following image shows how the FlexGrid appears after setting the **ChildItemsPath** property.



The following code examples demonstrate how to enable TreeView in the FlexGrid:

In Code

Add a Model

1. Add a new class to the folder **Models** (for example: `TreeItem.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the tree view structure of a sample folder.

TreeItem.cs

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

namespace MVCFlexGrid.Models
{
    public interface ITreeItem
    {
        string Header { get; set; }
        IList<ITreeItem> Children { get; }
    }
    public class Folder : ITreeItem
    {
        public string Header { get; set; }
        public IList<ITreeItem> Children { get; private set; }

        public Folder(string name)
```

```
        {
            Header = name;
            Children = new List<ITreeItem>();
        }
        public static Folder Create(string path)
        {
            var folder = new Folder(System.IO.Path.GetFileName(path));
            System.IO.Directory.GetDirectories(path).ToList().ForEach(d =>
folder.Children.Add(Folder.Create(d)));
            System.IO.Directory.GetFiles(path).ToList().ForEach(f =>
folder.Children.Add(File.Create(f)));
            return folder;
        }
    }
    public class File : ITreeItem
    {
        public string Header { get; set; }
        public DateTime DateModified { get; set; }
        public long Size { get; set; }
        public IList<ITreeItem> Children { get { return null; } }

        public File(string name)
        {
            Header = name;
        }
        public static File Create(string path)
        {
            var file = new File(System.IO.Path.GetFileName(path));
            var info = new System.IO.FileInfo(path);
            file.DateModified = info.LastWriteTime;
            file.Size = info.Length;
            return file;
        }
    }
}
```

Back to Top

Add a controller and a view

TreeViewController.cs

C# copyCode

```
public ActionResult TreeView()
{
    var list = MVCFlexGrid.Models.Folder.Create(Server.MapPath("~")).Children;
    return View(list);
}
```

TreeView.cshtml

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Grid
@model IEnumerable<MVCFlexGrid.Models.ITreeItem>

<style>
    .wj-flexgrid {
        height: 400px;
        background-color: white;
        box-shadow: 4px 4px 10px 0px rgba(50, 50, 50, 0.75);
        margin-bottom: 12px;
    }

    .custom-flex-grid .wj-header.wj-cell {
        color: #fff;
        background-color: #000;
        border-bottom: solid 1px #404040;
        border-right: solid 1px #404040;
        font-weight: bold;
    }

    .custom-flex-grid .wj-cell {
        background-color: #fff;
        border: none;
    }

    .custom-flex-grid .wj-alt:not(.wj-state-selected):not(.wj-state-multi-selected) {
        background-color: #fff;
    }

    .custom-flex-grid .wj-state-selected {
        background: #000;
        color: #fff;
    }

    .custom-flex-grid .wj-state-multi-selected {
        background: #222;
        color: #fff;
    }
</style>

@(Html.C1().FlexGrid().CssClass("custom-flex-grid")
    .Bind(Model)
    .Width(600)
    .ChildItemsPath("Children")
    .AutoGenerateColumns(false)
    .Columns(columns =>
    {
        columns.Add().Binding("Header").Width("*");
```

```
        columns.Add().Binding("Size").Width("80").Align("center");
    })
    .AllowResizing(AllowResizing.None)
    .HeadersVisibility(HeadersVisibility.None)
    .SelectionMode(SelectionMode.ListBox)
)
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Grid
@using TagFlexGrid.Models
@using Microsoft.Framework.Runtime

@model IEnumerable<TagFlexGrid.Models.ITreeItem>
@inject IApplicationEnvironment appEnvironment
@{
    var list = TagFlexGrid.Models.Folder
.Create(appEnvironment.ApplicationBasePath).Children;
}
<style>
    .custom-flex-grid {
        height: 400px;
        background-color: white;
        box-shadow: 4px 4px 10px 0px rgba(50, 50, 50, 0.75);
        margin-bottom: 12px;
    }

    .custom-flex-grid .wj-cell {
        background-color: #fff;
        border: none;
        font-size: 10pt;
    }

    .custom-flex-grid .wj-state-selected {
        background: #000;
        color: #fff;
    }

    .custom-flex-grid .wj-state-multi-selected {
        background: #222;
        color: #fff;
    }
</style>

<label>The file structure of this sample project</label>
<c1-flex-grid id="grid" class="custom-flex-grid" width="500px"
    child-items-path="Children" is-read-only="true" auto-generate-columns="false"
    allow-resizing="AllowResizing.None" headers-visibility="HeadersVisibility.None"
    selection-mode="SelectionMode.ListBox">
```

```
    <c1-items-source source-collection="list"></c1-items-source>
    <c1-flex-grid-column binding="Header" width="*"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Size" width="80" align="center">
</c1-flex-grid-column>
</c1-flex-grid>

<script>
    c1.mvc.Utils.documentReady(function () {
        wijmo.Control.getControl("#grid").rows.defaultSize = 25;
    });
</script>
```

FlexGrid ASP.NET Core Tags

FlexGrid control supports the following ASP.NET Core Tags:

FlexGrid class

- allow-add-new
- allow-delete
- allow-dragging

- allow-merging
- allow-resizing
- allow-sorting
- auto-clipboard
- auto-generate-columns
- auto-size-mode
- c1-flex-grid-cell-template
- child-items-path
- ColumnHeadersTemplate
- c1-flex-grid-column
- class
- style
- defer-resizing
- frozen-columns
- frozen-rows
- group-by
- group-header-format
- headers-visibility
- height
- id
- is-read-only
- item-formatter
- c1-items-source
- auto-sized-column
- auto-sized-row
- auto-sizing-column
- auto-sizing-row
- beginning-edit
- cell-edit-ended
- cell-edit-ending
- copied
- copying
- deleting-row
- dragged-column
- dragged-row
- dragging-column
- dragging-row
- format-item
- group-collapsed-changed
- groupCollapsedChanging
- items-source-changed
- loaded-rows
- loading-rows
- pasted
- pasting
- prepare-cell-for-edit
- resized-column
- resized-row
- resizing-column
- resizing-row
- row-added
- row-edit-ended
- row-edit-ending
- scroll-position-changed

- selection-changed
- selection-changing
- sorted-column
- sorting-column
- order-by
- RowHeadersTemplate
- scroll-position
- selection
- selection-mode
- show-groups
- show-sort
- sort-row-index
- template-bindings
- TopLeftCellsTemplate
- tree-indent
- width

CellTemplate

- c1-property
- cell-type
- id
- is-editing

Columns

- aggregate
- align
- allow-dragging
- allow-merging
- allow-resizing
- allow-sorting
- binding
- CellTemplate
- class
- c1-property
- data-type
- format
- header
- input-type
- is-content-html
- is-read-only
- is-selected
- mask
- max-width
- min-width
- name
- required
- sort-member-path
- visible
- width
- word-wrap

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- c1-property-group-description
- initial-items-count
- query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

- client-converter
- property-name

SortDescriptions

- ascending
- property

GroupPanel

- hide-grouped-columns
- placeholder
- max-groups
- selector

GridFilter

- column-filter-types
- default-filter-type
- show-filter-icons
- show-sort-buttons

DetailRow

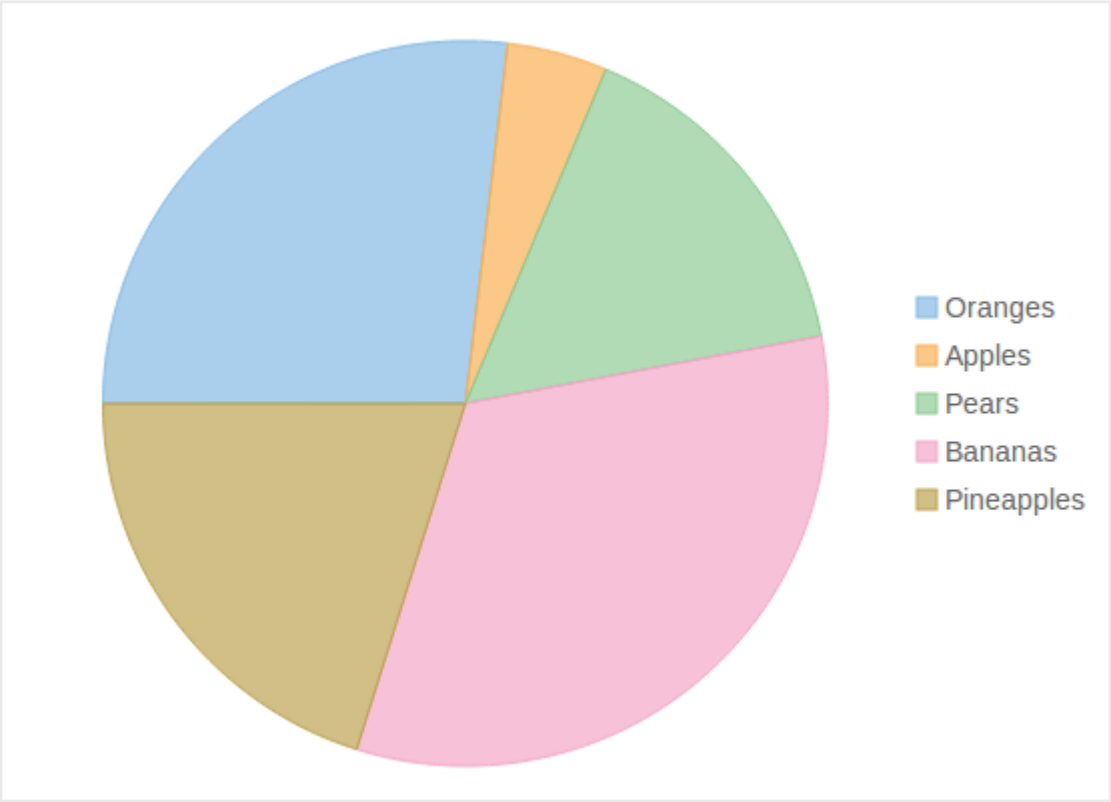
- detail-visibility-mode
- row-has-detail
- create-detail-cell
- dispose-detail-cell
- max-height

DataMap

- display-member-path
- selected-value-path
- c1-items-source

FlexPie

The FlexPie control allows you to create customized pie charts that represent a series as slices of a pie, wherein the arc length of each slice depicts the value represented by that slice. These charts are commonly used to display proportional data such as percentage cover. The multi-colored slices make pie charts easy to understand and usually the value represented by each slice is displayed with the help of labels.



Key Features

- **Header and Footer**: Use simple properties to set a title and footer text.
- **Legend**: Change position of the legend as needed.
- **Selection**: Change the selection mode and customize the selected pie slice appearance.
- **Exploding and Donut Pie Charts**: Use simple properties to convert it into an exploding pie chart or a donut pie chart.

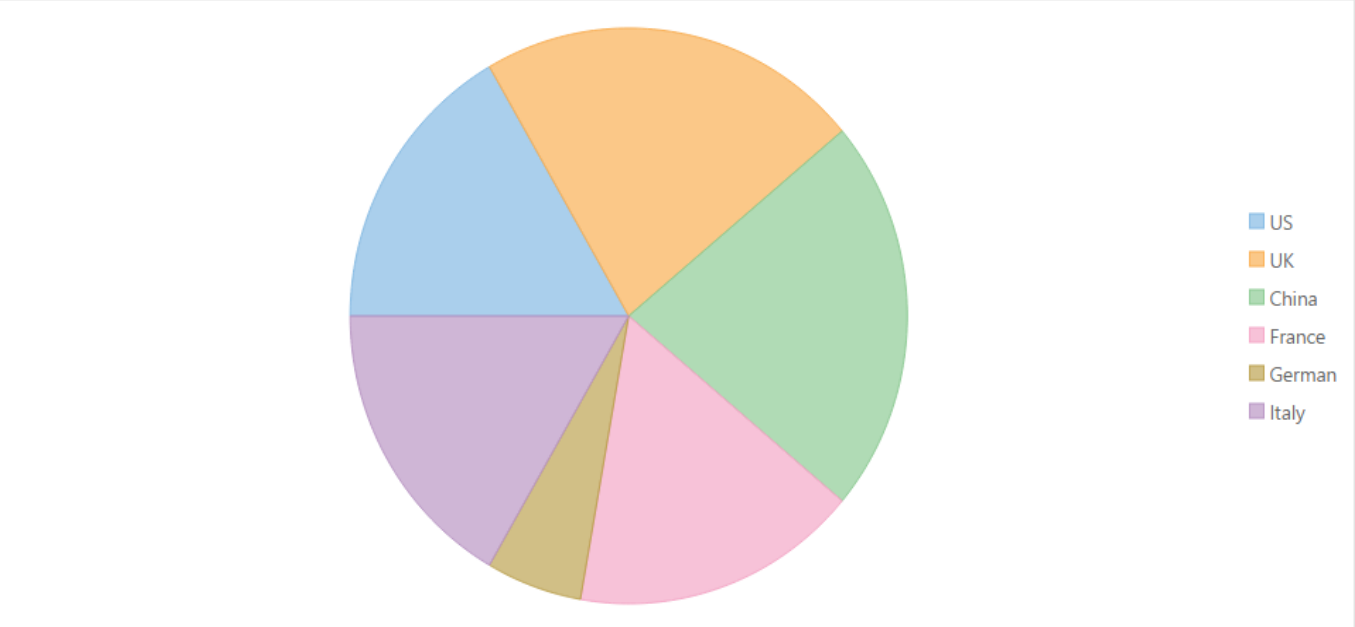
Quick Start: Add data to FlexPie

This section describes how to add a FlexPie control to your MVC web application and add data to it.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for FlexPie**
- **Step 3: Add a FlexPie control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for FlexPie

1. Add a new class to the folder **Models** (for example: `FlexPieDataSource.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes that serve as a datasource for the FlexPie control.

C#

C# copyCode

```
public class FlexPieDataSource
 {
   public string Country { get; set; }
   public int Sales { get; set; }

 public static IEnumerable<FlexPieDataSource> GetData()
  {
    var countries = new[] { "US", "UK",   "China", "France", "German", "Italy"
};
    var rand = new Random(0);
    List<FlexPieDataSource> list = new List<FlexPieDataSource>();
     for (int i = 0; i < 6; i++)
      {
        var sales = rand.Next(1, 5);
        list.Add(new FlexPieDataSource {  Sales = sales, Country = countries[i]
});
            }
            return list;
```

```
        }
    }
```

VB

VB

```
Public Class FlexPieDataSource
    Public Property Country() As String
        Get
            Return m_Country
        End Get
        Set
            m_Country = Value
        End Set
    End Property
    Private m_Country As String
    Public Property Sales() As Integer
        Get
            Return m_Sales
        End Get
        Set
            m_Sales = Value
        End Set
    End Property
    Private m_Sales As Integer

    Public Shared Function GetData() As IEnumerable(Of FlexPieDataSource)
        Dim countries = New String() {"US", "UK", "China", "France", "German",
"Italy"}
        Dim rand = New Random(0)
        Dim list As New List(Of FlexPieDataSource)()
        For i As Integer = 0 To 5
            Dim sales = rand.[Next](1, 5)
            list.Add(New FlexPieDataSource() With {
                Key.Sales = sales,
                Key.Country = countries(i)
            })
        Next
        Return list
    End Function
End Class
```

Back to Top

Step 3: Add a FlexPie control

Complete the following steps to initialize a FlexPie control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

```
C#
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method `Index()` with the following method.

C#

```
C#                                                                copyCode
public ActionResult QuickStart()
{
    return View(FlexPieDataSource.GetData());
}
```

VB

```
VB
Public Function QuickStart() As ActionResult
Return View(FlexPieDataSource.GetData())
End Function
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller `QuickStartController` to open it.
2. Place the cursor inside the method `QuickStart()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **QuickStart** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.
6. Instantiate a FlexPie control in the view `QuickStart` as shown below.

HTML Helpers

```
Index.cshtml                                                      copyCode
@using MvcApplication1.Models
@model IEnumerable<FlexPieDataSource>
@(Html.C1().FlexPie<FlexPieDataSource>()
.Bind("Country", "Sales", Model)
)
```

```
Index.vbhtml
@ModelType IEnumerable(Of FlexPieDataSource)
```

```
@(Html.C1().FlexPie(Of FlexPieDataSource) _
.Bind("Country", "Sales", Model) _
.Height("300px"))
```

Tag Helpers

Index.cshtml copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales">
    <c1-items-source source-collection="Model"></c1-items-source>
</c1-flex-pie>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**QuickStart/QuickStart**) in the address bar of the browser to see the view.

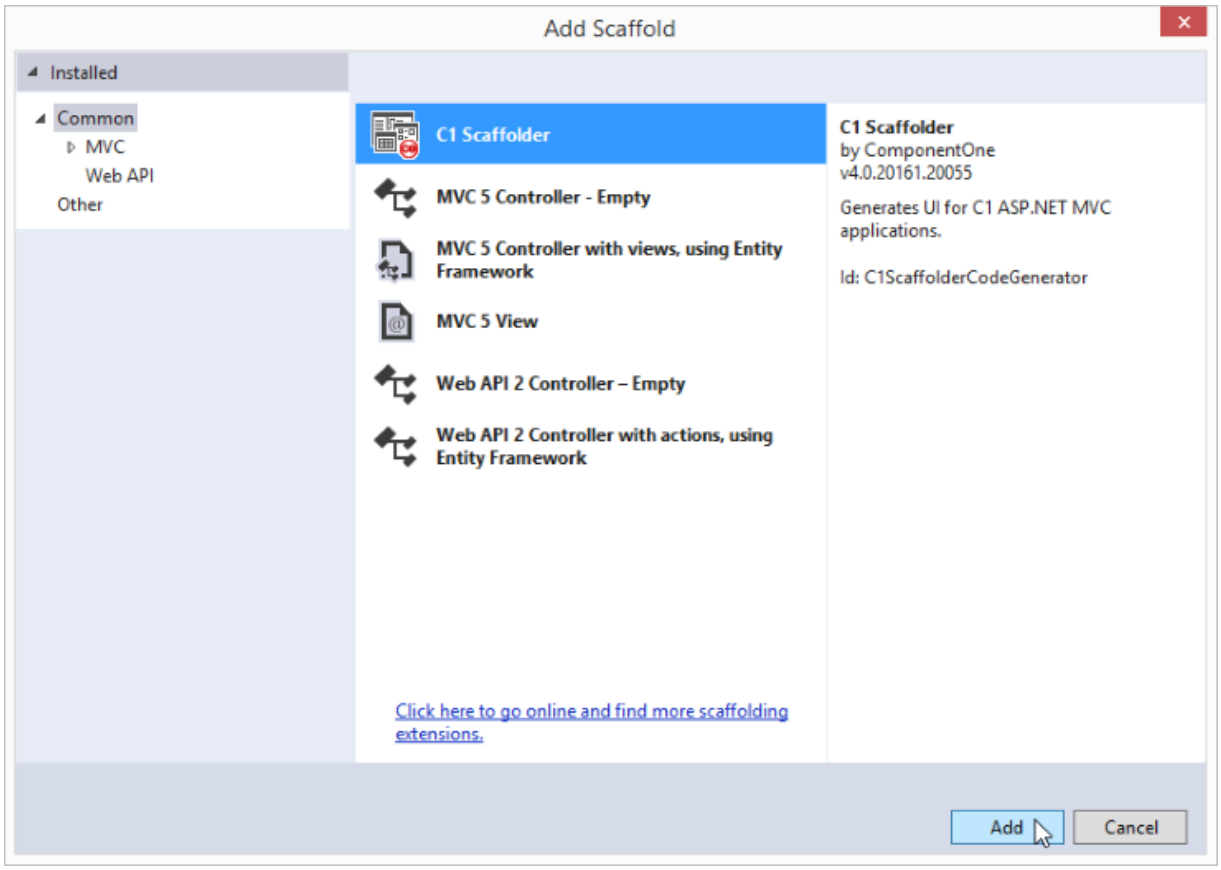
Back to Top

Features

Scaffolding in FlexPie Control

The steps to scaffold **ComponentOne Flex Pie** control for ASP.NET MVC are as follows:

1. Configure the datasource. Refer the topic Data Source Configuration to see configuring a datasource in an application.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane.



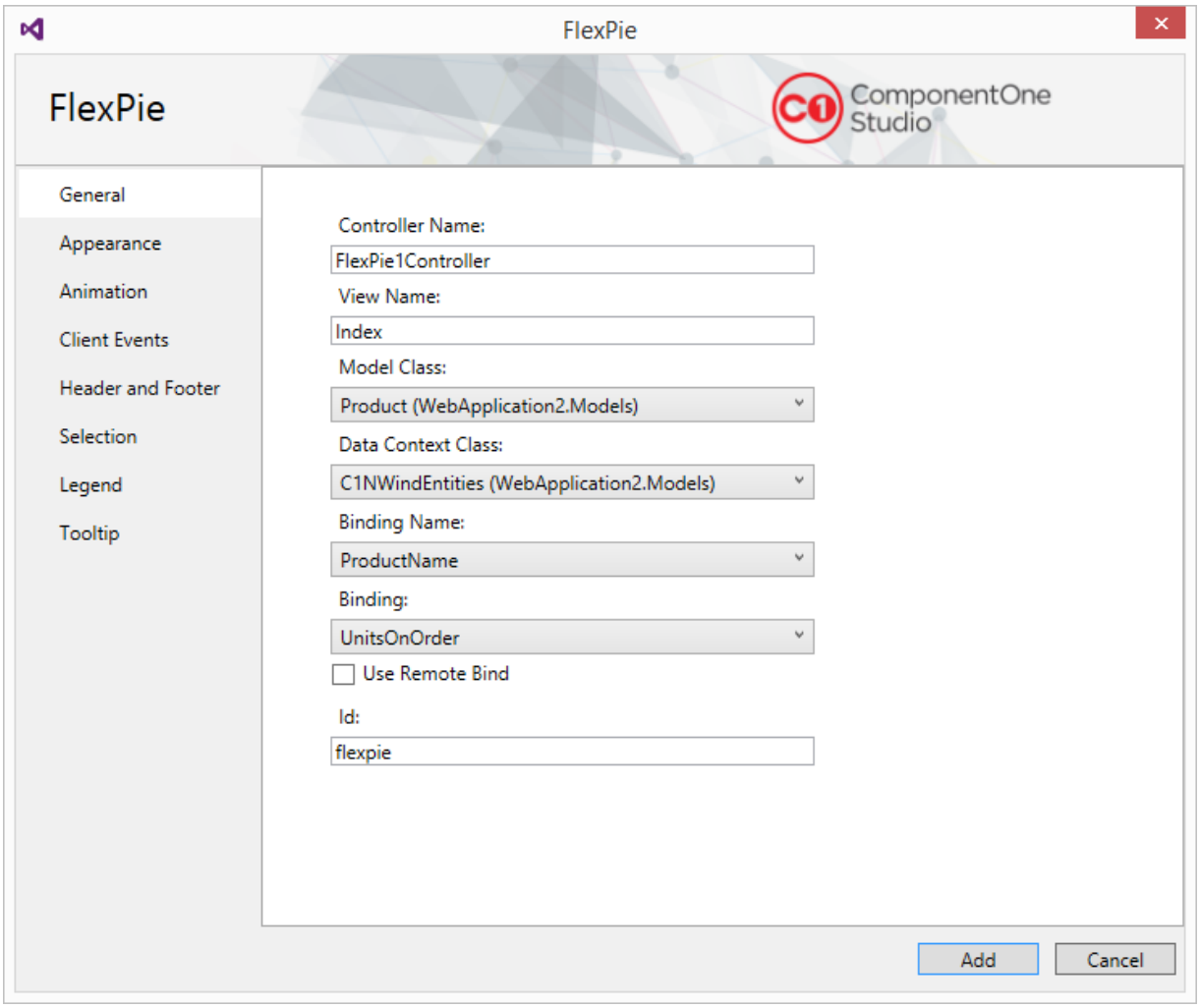
You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

4. Click **Add**.
5. In the **Add Scaffold** dialog, select FlexPie control, and the click **Next**.

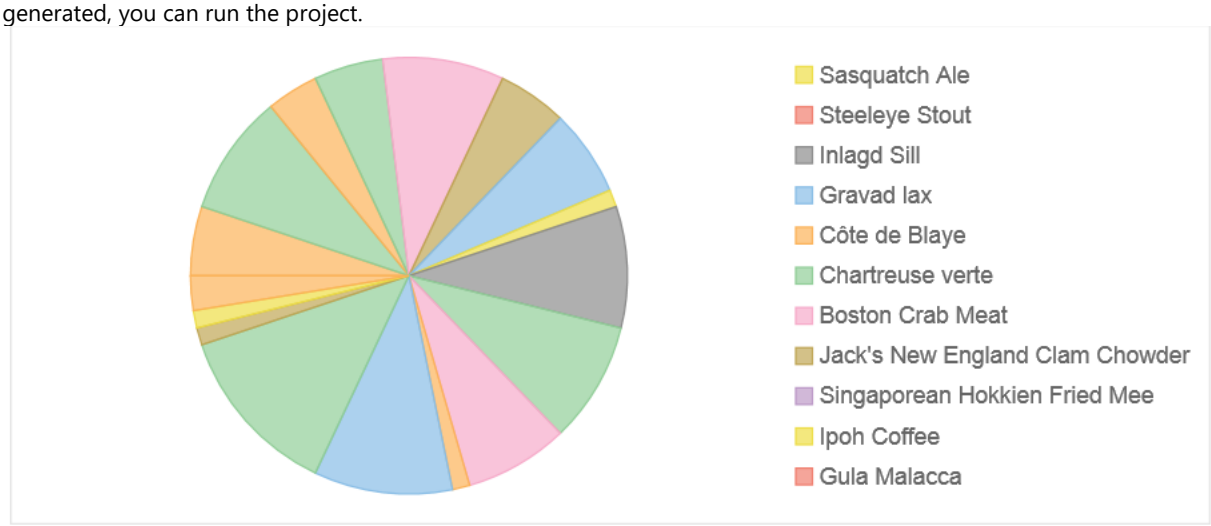
The **C1 ASP.NET MVC FlexPie** wizard appears with the General tab selected by default.

6. In the General tab, specify the model details as follows:

    1. Enter the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in **Step 1**. In our case, we select Product to populate data for the FlexChart.
    3. Select **Data Context Class** from the drop-down list. In our case, we select C1NWindEntities.
    4. In the **Binding Name** drop-down, select a column to bind it with your FlexPie. In our case, we have selected **ProductName**.
    5. In the **Binding** drop-down, select a column to bind it with your FlexPie. In our case, we have selected **UnitsOnOrder**.



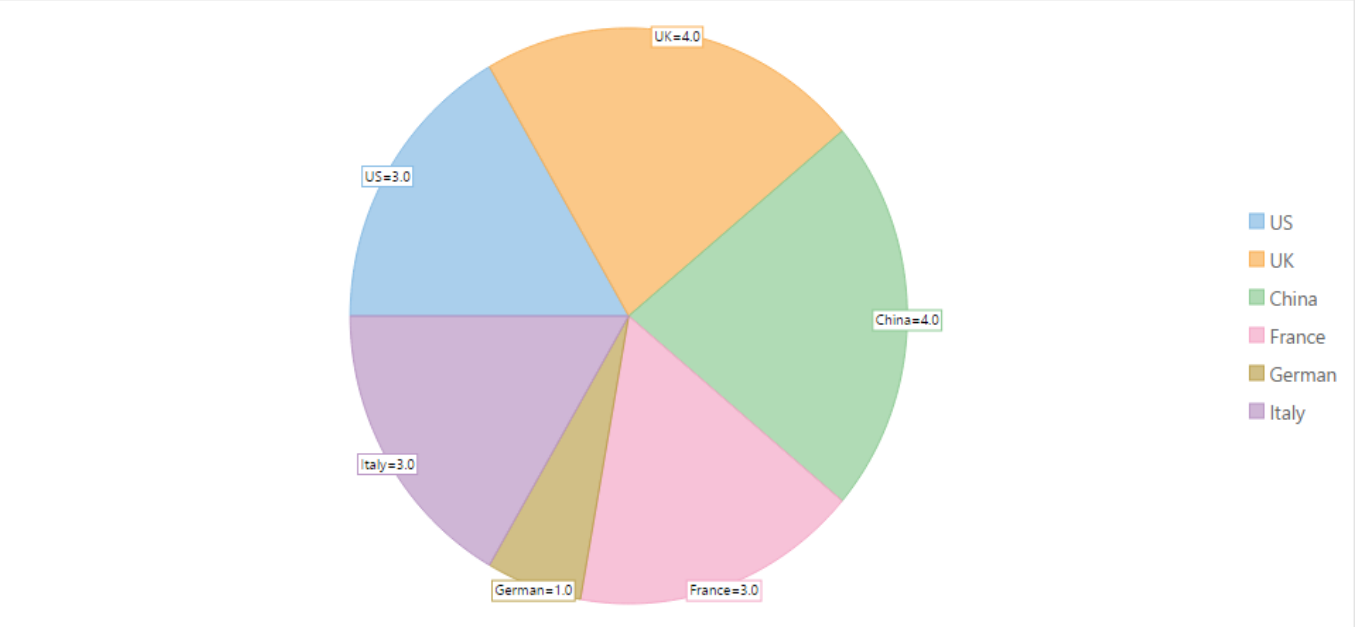
7. Click **Add**. You will see that the Controller and View for the selected model are added to your project. Once the code is generated, you can run the project.



Customize FlexPie

The `ItemFormatter` property lets you customize the FlexPie using a JavaScript function. This topic demonstrates how to add labels over the pie slices with the help of this property.

The image below shows the how the labels appear over the pie slices.



The following code examples demonstrate how to add custom labels over the pie slices.

In Code

To accomplish this, you must first write the script to add labels or similar custom content over FlexPie.

Javascript copyCode

```
<script>
    function formatItem(label, hitTestInfo, defaultFormatter)
    {
        var fsz = label.fontSize;
        label.fontSize = '10';
        defaultFormatter();
        var point = hitTestInfo.point.clone();
        var text = hitTestInfo.name + '=' + hitTestInfo.value.toFixed(1);
        var sz = label.measureString(text);
        var fill = label.fill;
        label.fill = 'white';
        label.drawRect(point.x - 2 - sz.width / 2, point.y - sz.height + 10, sz.width
+ 4, sz.height);
        label.fill = fill;
        point.x -= sz.width / 2;
        point.y += 9;
        label.drawString(text, point);
        label.fontSize = fsz;
    }
</script>
```

Set the `ItemFormatter` property as shown in the code below to add the labels onto the FlexPie. The following example uses the sample created in the Quick Start topic.

HTML Helpers

Razor copyCode

```
.ItemFormatter("formatItem")
```

Tag Helpers

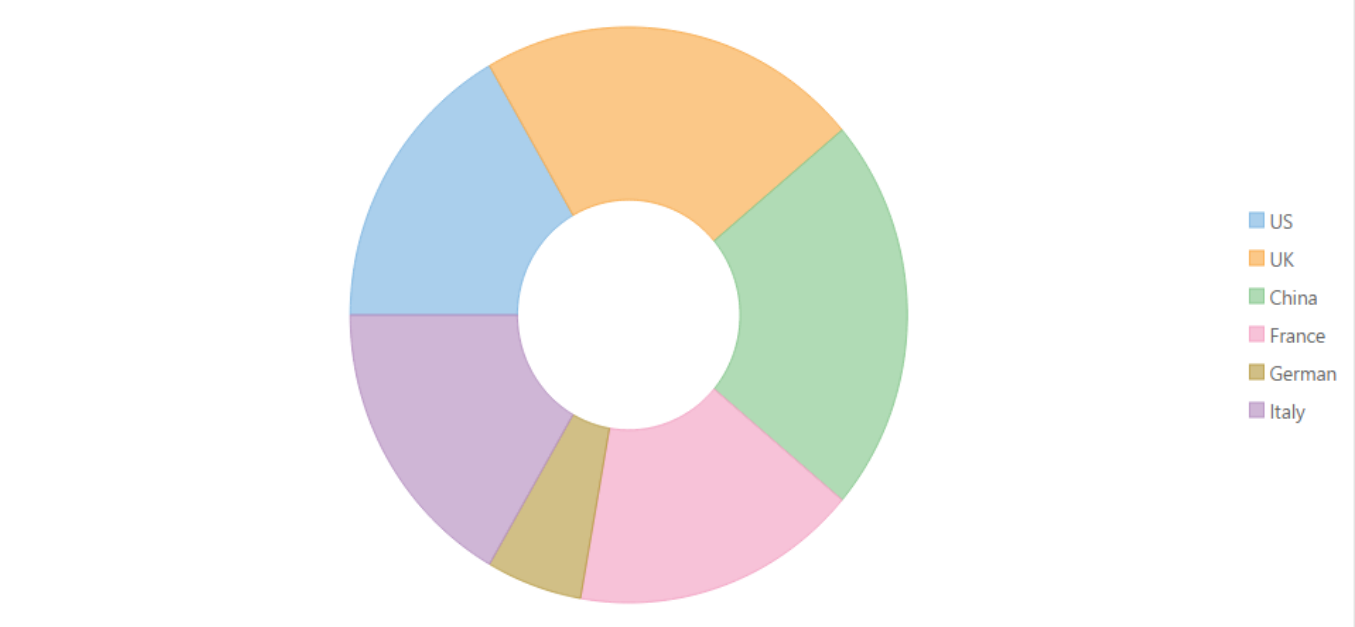
HTML copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales" item-formatter="formatItem">
</c1-flex-pie>
```

Donut Pie Chart

The `InnerRadius` property can be used to leave a blank inner space in the FlexPie, creating a Donut Pie Chart. The blank space can be used to display additional data.

The following image shows a donut FlexPie.



The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.InnerRadius(0.4f)
```

Tag Helpers

HTML copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales" inner-radius="0.4f">
</c1-flex-pie>
```

Exploded Pie Chart

The `Offset` property can be used to push the pie slices away from the center of the FlexPie, producing an exploded pie chart. This property accepts a decimal value to determine how far the pie slices should be pushed from the center.

The image below shows an exploded FlexPie.

The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor	copyCode
`.Offset(0.5f)`	

Tag Helpers

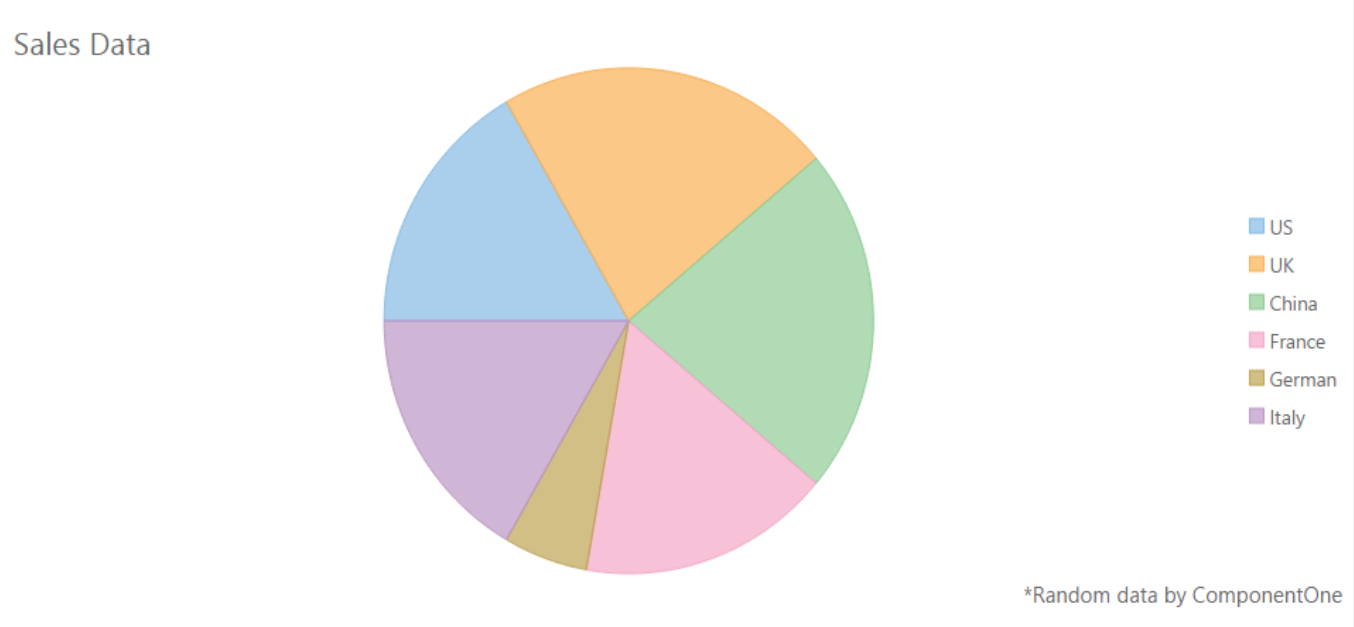
HTML	copyCode
`<c1-flex-pie binding-name="Country" binding="Sales" offset="0.5f">` `</c1-flex-pie>`	

Header and Footer

You can add a title to the FlexPie control by setting its `Header` property. Besides a title, you may also set a footer for the chart by setting the `Footer` property. You can also style the header and footer text with the help of

`HeaderStyle` and `FooterStyle` properties.

The image below shows how the FlexPie appears, after these properties have been set.



The following code example demonstrates how to set these properties. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

C# copyCode

```
.Header("Sales Data")
.HeaderStyle(style => style.FontSize("30").Halign("left"))
.Footer("*Random data by ComponentOne")
.FooterStyle(style => style.FontSize("15").Halign("right"))
```

Tag Helpers

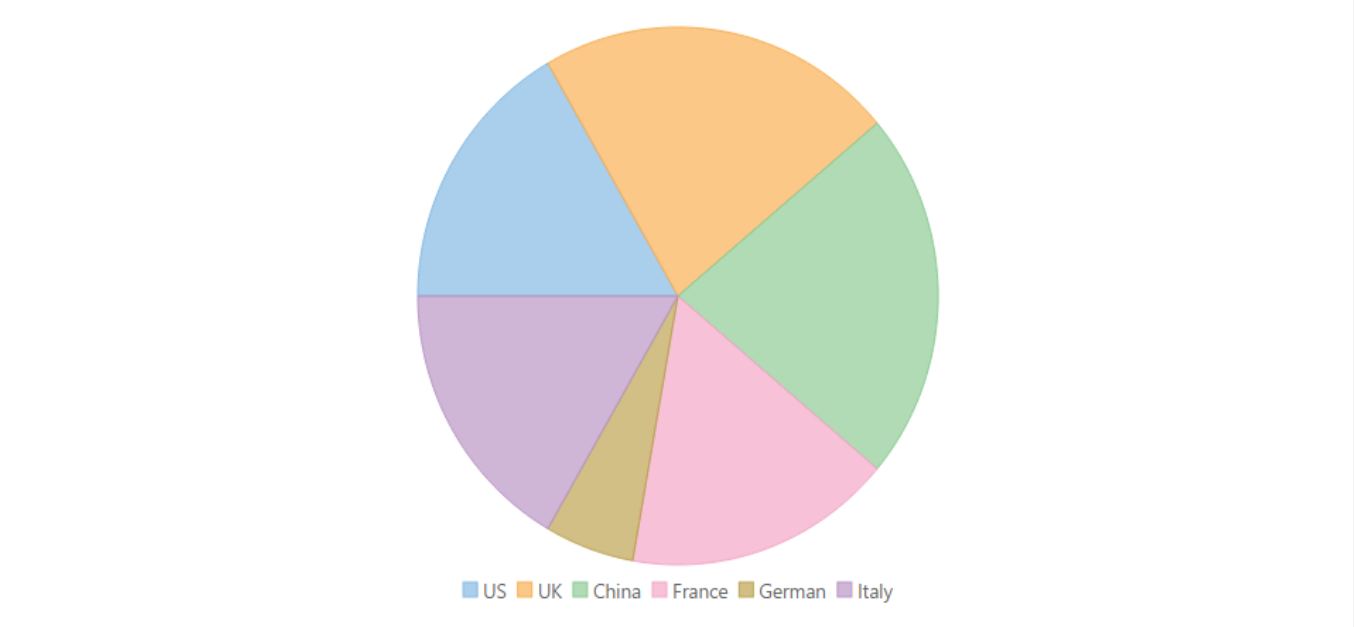
HTML copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales" header="Sales Data"
footer="*Random data by ComponentOne">
    <c1-flex-chart-title-style c1-property="FooterStyle" halign="right"></c1-flex-
chart-title-style>
    <c1-flex-chart-title-style c1-property="HeaderStyle" halign="left"></c1-flex-
chart-title-style>
    <c1-items-source source-collection="Model"></c1-items-source>
</c1-flex-pie>
```

Legend

You can specify the position where you want to display the legend in the `Legend` property of the FlexPie.

The image below shows how the FlexPie appears after these properties have been set.



The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.Legend(C1.Web.Mvc.Chart.Position.Bottom)
```

Tag Helpers

HTML copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales" legend-
position="Position.Bottom">
    <c1-items-source source-collection="Model"></c1-items-source>
</c1-flex-pie>
```

Selection

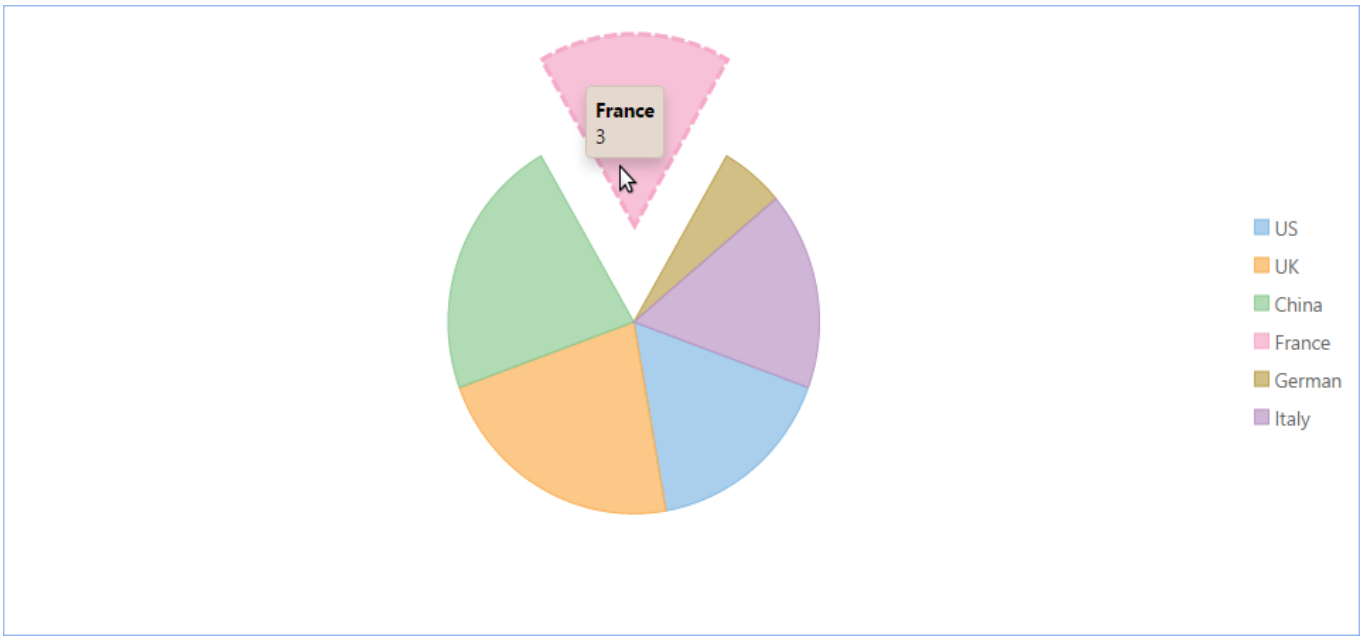
You can choose what element of the FlexPie should be selected when the user clicks on any region in a FlexPie by setting the `SelectionMode` property. This property provides three options:

- **None**: Does not select any element.
- **Point**: Highlights the pie slice that the user clicks.
- **Series**: Highlights the entire pie.

When the `SelectionMode` is set to **Point**, you can even change the position of the selected pie slice by setting the

`SelectedItemPosition` property and move the selected pie slice away from the center of the FlexPie by setting the `SelectedItemOffset` property.

The images below show how the FlexPie appears after these properties have been set.



The following code example demonstrates how to set this property. This example uses the sample created in the Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
.SelectionMode(C1.Web.Mvc.Chart.SelectionMode.Point)
.SelectedItemPosition(C1.Web.Mvc.Chart.Position.Top)
.SelectedItemOffset(0.5f)
```

Tag Helpers

HTML copyCode

```
<c1-flex-pie binding-name="Country" binding="Sales"
             selection-mode="SelectionMode.Point"
             selected-item-position="Position.Top"
             selected-item-offset="0.5f" >>
    <c1-items-source source-collection="Model"></c1-items-source>
</c1-flex-pie>
```

Theming

Enhance the appearance of the control by using pre-defined themes. The `Palette` property can be used to specify the theme to be applied over the control.

The following Palettes are currently available:

- Cocoa
- Coral
- Dark
- HighContrast
- Light
- Midnight
- Minimal
- Modern
- Organic
- Slate
- Standard

The following code example demonstrates how to set a theme in C#.

FlexPie ASP.NET Core Tags

FlexPie control supports the following ASP.NET Core Tags:

FlexPie<T> class

- binding
- binding-name
- class
- style
- DataLabel
- footer
- FooterStyle
- header
- HeaderStyle
- height
- id
- inner-radius
- is-animated
- item-formatter
- ItemsSource
- Legend-Position
- offset
- rendered
- rendering
- palette
- plot-margin
- reversed
- selected-item-offset
- selected-item-position
- selection-mode
- start-angle
- template-bindings
- Tooltip
- width

DataLabel class

- border
- content
- position

Footer TitleStyle class

- fill
- font-family
- font-size
- foreground
- halign

Header TitleStyle class

- fill
- font-family
- font-size
- foreground
- halign

Tooltip ChartTooltip class

- content
- gap
- hide-delay
- is-content-html
- show-delay
- threshold

ItemSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

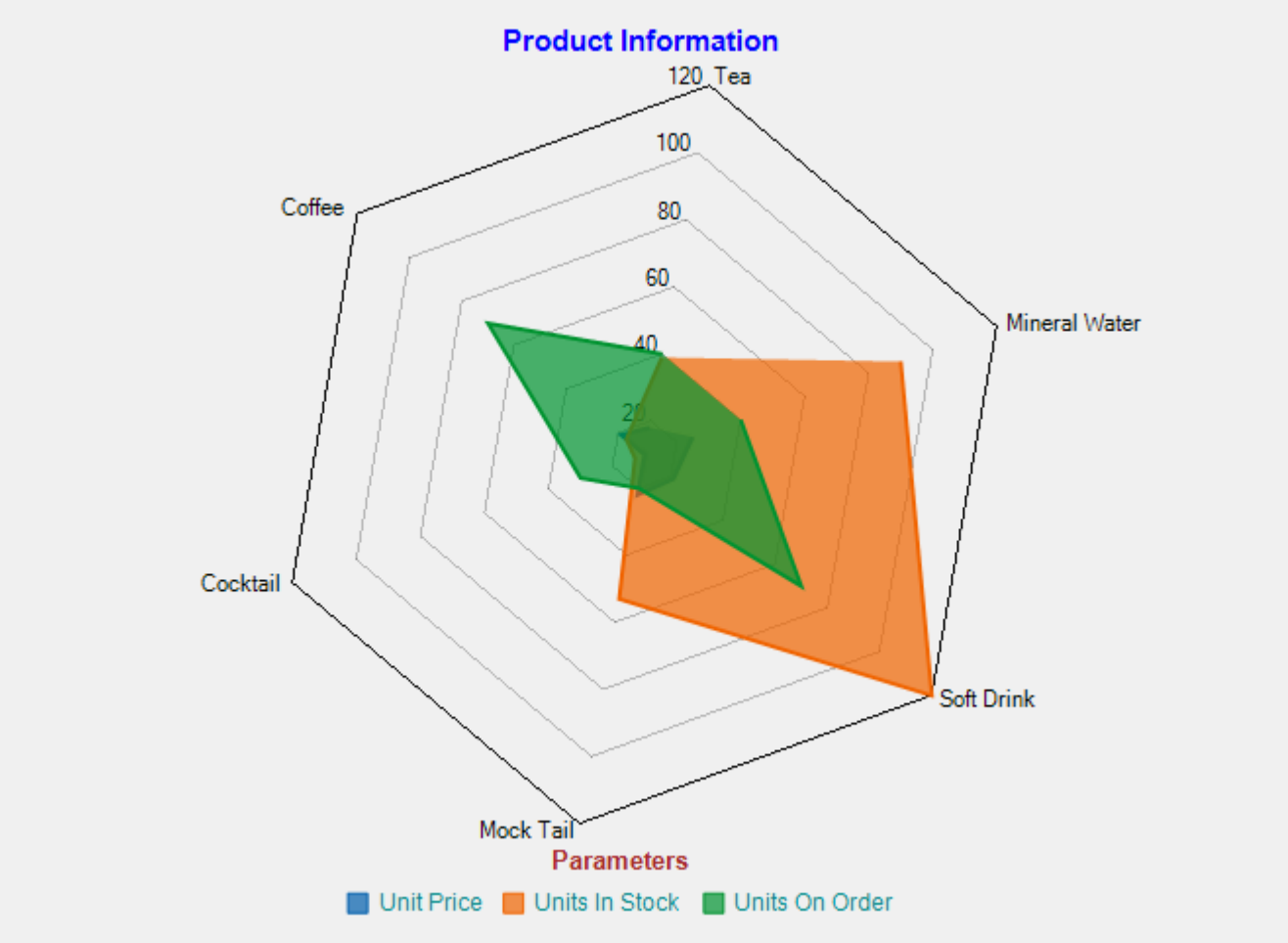
- client-converter
- property-name

SortDescriptions

- ascending
- property

FlexRadar

FlexRadar chart display multivariate data in the form of a two-dimensional chart of three or more quantitative variables. These variables are represented on axes starting from the same point. The chart plots value of each variable along a separate axis that starts from the center and ends on the outer ring. All axes are arranged radically, with equYal distances between each other, while maintaining the same scale between all axes. Each variable value is plotted along its individual axis and all the values are connected together to form a polygon. The centre of the chart represents the minimum value, and the chart edge represents the maximum value.



Common business applications of FlexRadar can include skill analysis of employees and product comparison. It is important to note that the FlexRadar control represents a polar chart when X values are numbers that specify angular values in degrees.

Key Features

- **Chart types:** FlexRadar allows you to choose from different chart types according to the data you want to visualize in the chart. The supported chart types are: Area, Column, Line, LineSymbols and Scatter.
- **Start angle:** FlexRadar allows you to set start angle of the chart(in degrees) which, by default, draws radial axes in the clockwise direction starting at the 12o'clock position.
- **Header and Footer**: FlexRadar allows you to set header and footer of the chart using which you can set chart

titles or supplement it with any other additional information regarding the chart or the data.

- **Legend**: FlexRadar allows you to display a legend to represent each series of the chart. You can also customize orientation, position, or style of the legend.

- **Reversed FlexRadar**: FlexRadar allows you to reverse the direction of plotting and display your data in anti-clockwise direction.

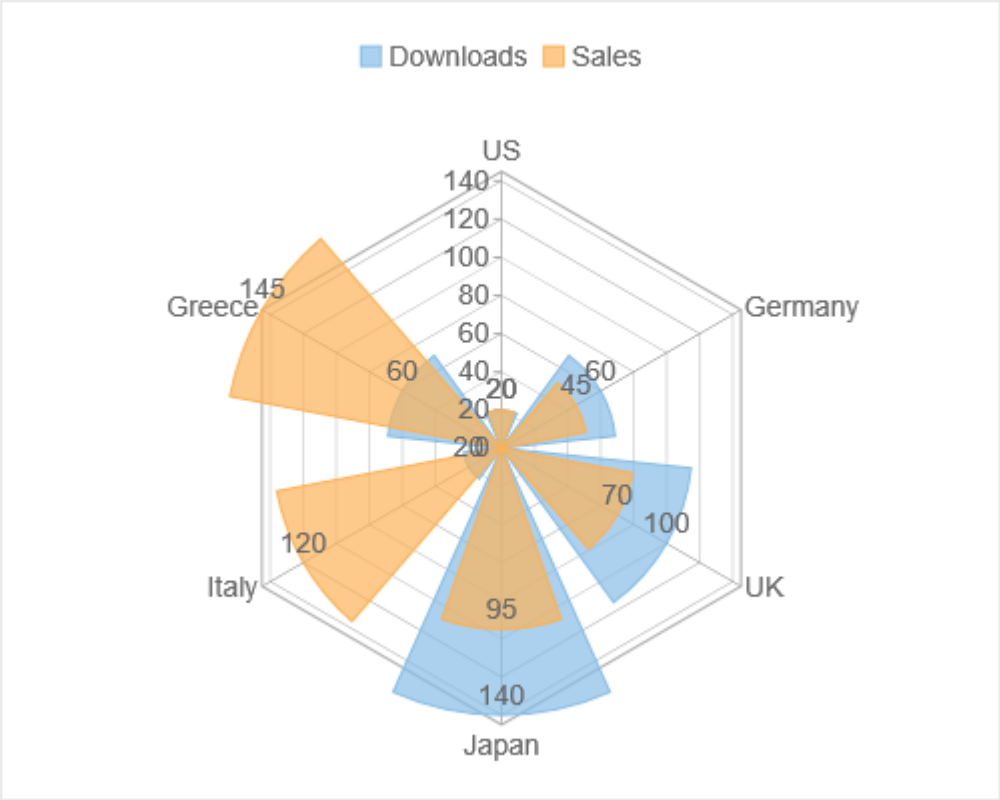
Quick Start: Add data to FlexRadar

The topic describes how to add a FlexRadar chart to your MVC web application and add data to it.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for FlexRadar**
- **Step 3: Add a FlexRadar Chart**
- **Step 4: Build and Run the Project**

The following image shows how FlexRadar chart appears after completing the steps:



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for FlexRadar

1. Add a new class to the **Models** folder (for example: `Sale.cs`). For more information on how to add a new

model, see Adding Controls.

2. Add the following code to `Sale.cs` model. We are using **`Sale`** class to represent sales data in the database. Each instance of **Sale** object will correspond to the data on the FlexChart.

C#

Sale.cs copyCode

```
using System.Collections.Generic;

namespace FlexRadarChart.Models
{
    public class Sale
    {
        public int Id { get; set; }
        public string Country { get; set; }
        public int Downloads { get; set; }
        public int Sales { get; set; }
        public static List<Sale> GetData()
        {
            var countries = "US,Germany,UK,Japan,Italy,Greece".Split(new char[] { ','
});
            List<Sale> data = new List<Sale>();
            for (var i = 0; i < countries.Length; i++)
            {
                data.Add(new Sale()
                {
                    Country = countries[i],

                    Downloads = ((i % 4) * 40) + 20,

                    Sales = ((i % 7) * 25) + 20
                });
            }
            return data;
        }
    }
}
```

Back to Top

Step 3: Add a FlexRadar chart

To add a FlexRadar chart to the application, follow these steps:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (for example: `FlexRadarController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method **Index()** with the following method.

C#

FlexRadarController.cs

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

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `FlexRadarController`.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml copyCode

```
@using FlexRadarChart.Models;
@model IEnumerable<Sale>
@using C1.Web.Mvc.Chart;

@(Html.C1().FlexRadar()
.Bind("Country", "Downloads", Model)
.ChartType(RadarChartType.Column)
.DataLabel(label =>{
label.Content("{y}");})
.Series(ser =>
{
    ser.Add().Name("Downloads");

    ser.Add().Binding("Sales").Name("Sales");
})
.Legend(Position.Top)
.Width("500px")
.Height("400px"))
```

Tag Helpers

Index.cshtml copyCode

```
@model IEnumerable<Sale>
@using C1.Web.Mvc.Chart

<c1-flex-radar binding="Downloads" binding-x="Country" height="400px"
width="500px"
chart-type="Column" legend-position="Top">
<c1-items-source source-collection="Model" />
<c1-flex-radar-series name="Downloads" />
<c1-flex-radar-series name="Sales" binding="Sales" />
</c1-flex-radar>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**FlexRadar/Index**) in the address bar of the browser to see the view.

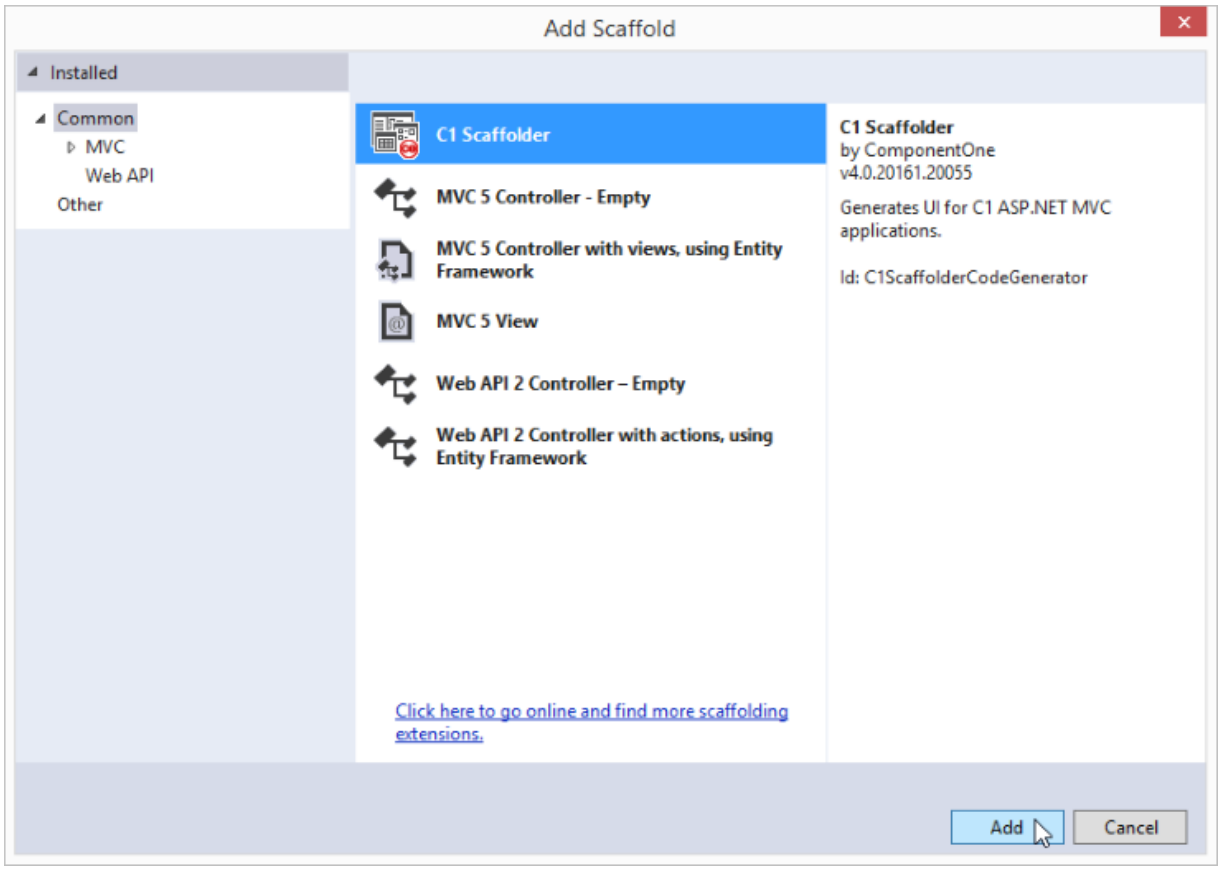
Back to Top

Features

Scaffolding

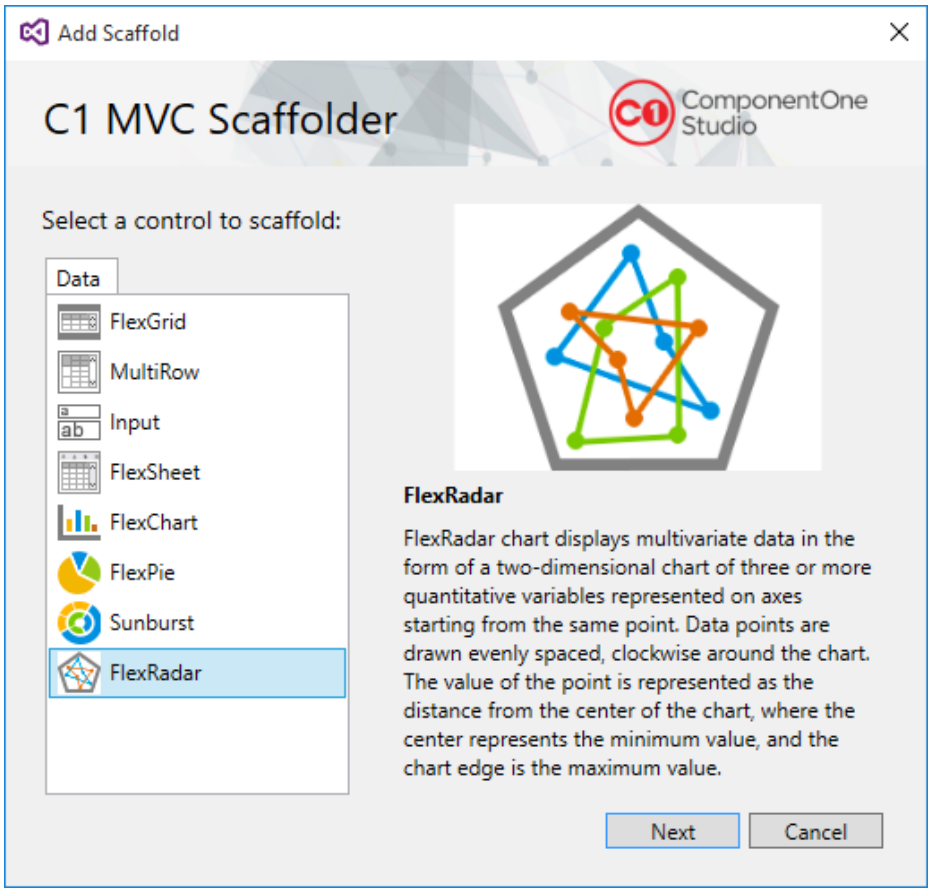
The steps to scaffold **ComponentOne FlexRadar** control for ASP.NET MVC are as follows:

1. Configure a datasource in your application. For more information on configuring a datasource, see Data Source Configuration topic.
2. In the **Solution Explorer**, right-click the project name and select **Add|New Scaffolded Item**. The Add Scaffold wizard appears.
3. In the **Add Scaffold** wizard, select **Common** and then select **C1 Scaffolder** from the right pane.



You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

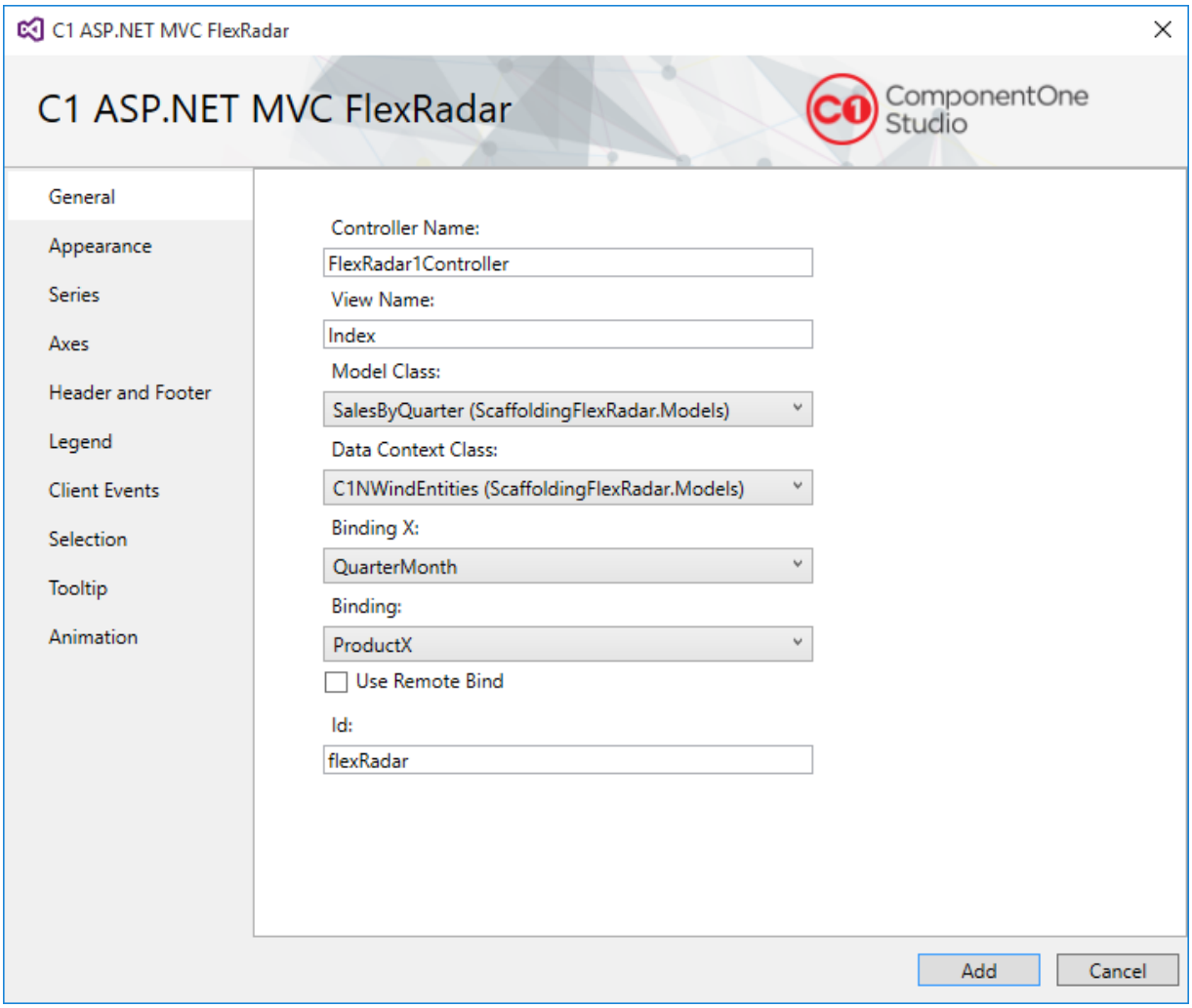
4. Click **Add**.
5. In the **Add Scaffold** dialog, select FlexRadar control, and the click **Next**.



The **C1 ASP.NET MVC FlexRadar** wizard appears with the General tab selected by default.

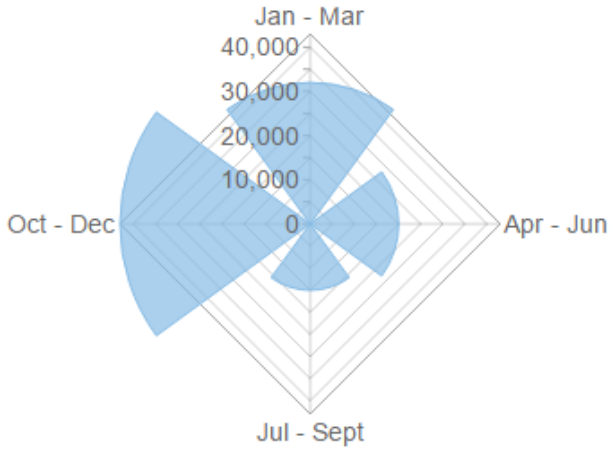
6. In the General tab, specify the model details as follows:

    1. Enter the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in **Step 1**. In our case, we select **SalesByQuarter** to populate data in FlexRadar.
    3. Select **Data Context Class** from the drop-down list. In our case, we select **C1NWindEntities**.
    4. In the **BindingX** drop-down, select a column to bind it with your FlexRadar. In our case, we have selected **QuarterMonth**.
    5. In the **Binding** drop-down, select a column to bind it with your FlexRadar. In our case, we have selected **ProductX**.



7. Click **Add**. You will see that the Controller and View for the selected model are added to your project. Once the code is generated, you can run the project using the **F5** button.

Output

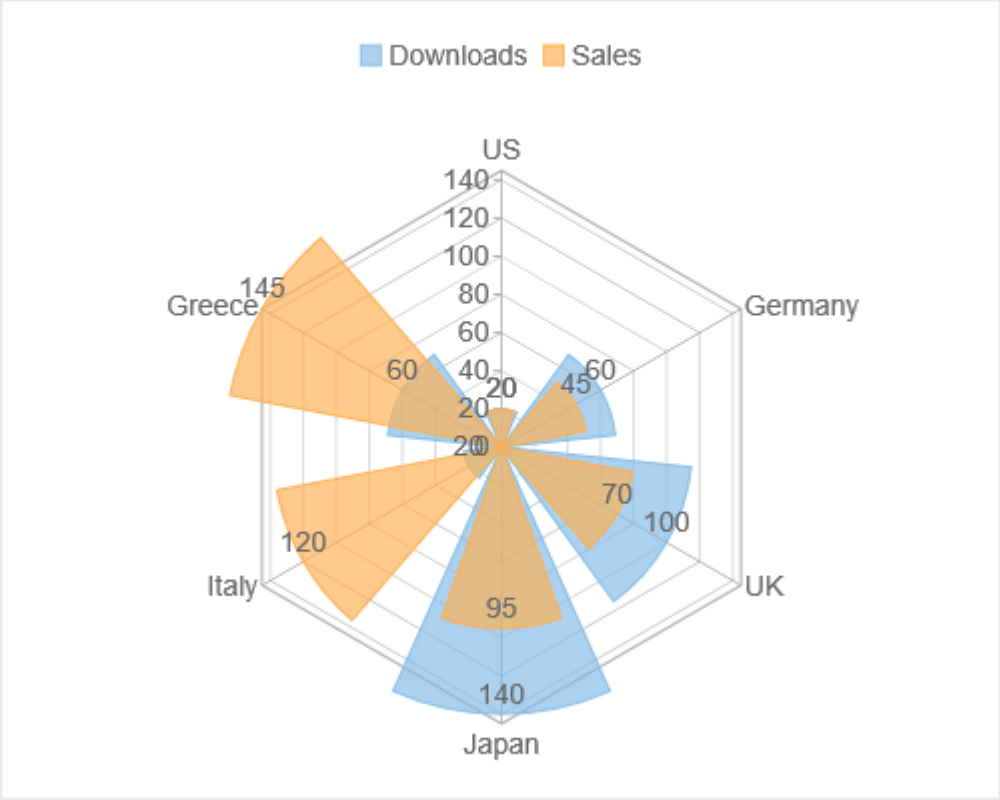


Legend

You can specify the position where you want to display the legend using the **Legend** property of the FlexRadar chart.

Legend helps in displaying the series of a chart with a predefined symbol and name of the series.

The image below shows how the FlexRadar chart appears after you set the Chart.Position property to **Top**.



The following code example demonstrates how to set the Position property. This example uses the sample created in the Quick Start topic.

In Code

HTML Helpers

Razor
`.Legend(C1.Web.Mvc.Chart.Position.Top)`

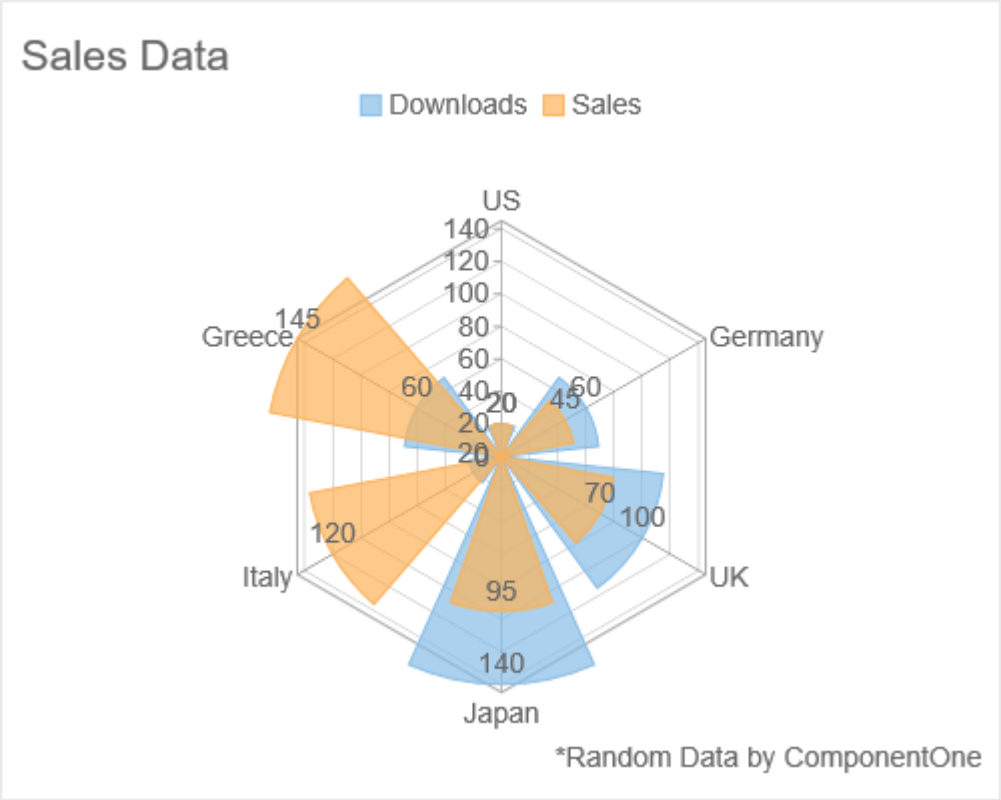
Tag Helpers

HTML
`<c1-flex-radar legend-position="Position.Top" legend-toggle="true">`

Header and Footer

You can add a title to the FlexRadar control by setting its Header property. Besides a title, you may also set a footer for the chart by setting the Footer property. You can also style the header and footer text with the help of HeaderStyle and FooterStyle properties.

The image below shows how the FlexRadar appears, after these properties have been set.



HTML Helpers

Razor

```
.Header("Sales Data").HeaderStyle(hs => hs.Halign("left")).FooterStyle(fs =>
fs.Halign("right")).Footer("*Random Data by ComponentOne")
```

Tag Helpers

HTML

```
<c1-flex-radar binding="Downloads" binding-x="Country" header="Sales Data"
footer="*Random Data by ComponentOne height="400px" width="500px" chart-type="Column"
legend-position="Top"><c1-flex-radar-title-style c1-property="FooterStyle"
halign="right"></c1-flex-radar-title-style><c1-flex-radar-title-style c1-
property="HeaderStyle" halign="left"></c1-flex-radar-title-style><c1-items-source
source-collection="Model" /><c1-flex-radar-series name="Downloads" /><c1-flex-radar-
series name="Sales" binding="Sales" />
```

FlexReport

Reporting is vital to business decision making and knowledge management. ASP.NET MVC Edition introduces FlexReport for MVC, to enable viewing reports on the web.

This light-weight and fast control comes as a comprehensive tool for enterprise reporting. It enables you to build complex high performance reports which can be previewed, exported, and printed easily. Reports generated through

FlexReport can be viewed with the help of FlexViewer, a versatile report viewing tool.

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

The FlexReport for ASP.NET MVC comprises:

1. **FlexReport Designer:** A standalone designer application to create reports. The designer application gets installed by default at **C:\Program Files (x86)\ComponentOne\Apps\v4.0**. To know more about how to create Report Definition and working with FlexReport designer see FlexReportDesigner documentation.
2. **FlexViewer Control:** FlexViewer control for **ASP.NET MVC Edition** allows you to preview FlexReports in your web applications.
3. **FlexReport Web API:** REST services with which the viewer communicates to load and display reports to the client.

Configuring FlexReport Web API

You can easily view reports created through FlexReport Designer on web, with the help of FlexReport Web API and a client Html 5 report viewer control.

The FlexReport Web API plays a vital role in communicating with the report viewer control (on the client side) which loads and displays the html content exported by FlexReport.

A FlexReport WebAPI service can be created in two ways:

1. Using C1 Web API Edition
2. Using Standard Visual Studio Web API Template

The following sections demonstrate how to create and configure FlexReport Web API, by using ComponentOne template and Standard Visual Studio template.

- Using ComponentOne Web API Edition Template
- Using Standard Visual Studio Web API Template
- Using C1ReportViewer Template

> **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.
> - Explore detailed demo samples of FlexViewer at How to use FlexViewer and FlexReport Web API at Report Services for Web API Edition.

Using ComponentOne WebAPI 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: Add Report files to Web API Application**
- **Step 2: Set Report's Root Location**
- **Step 3: Deploy FlexReport Web API Service**

Step 1: 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.
   This example is using the FlexCommonTasks.flxr report definition, which is provided in **FlexViewer HowTo** sample (it is by default installed at **~\Documents\ComponentOne Samples\ASP.NET MVC\MVC\HowTo\FlexViewer**).

   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 2: 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);
var folder = GetFullRoot("Files");
app.AddDiskStorage("root", folder);
```

   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 3: Deploy FlexReport 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:

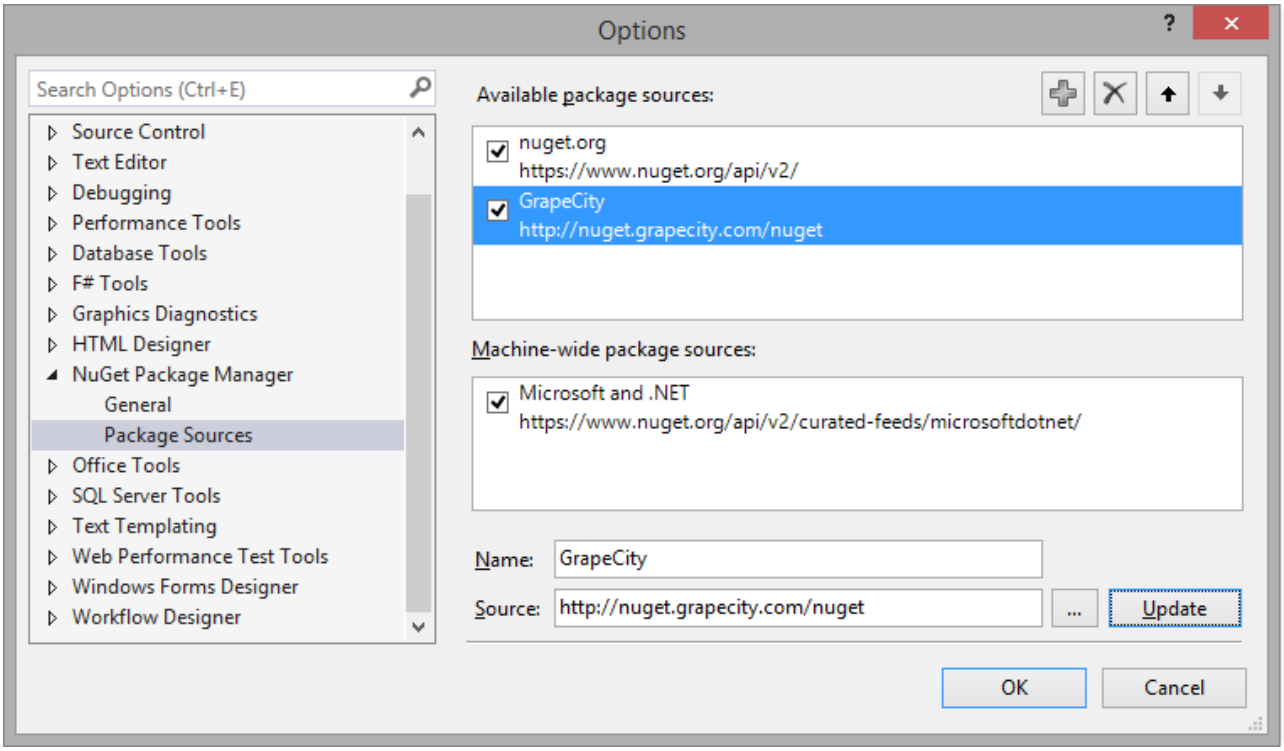
- **Step 1: Configure Web API Project**
- **Step 2: Add Report files to the Project**
- **Step 3: Set Report's Root Location and Use C1 Web APIs**
- **Step 4: Deploy FlexReport Web API Service**

Step 1: Configure Web API Project

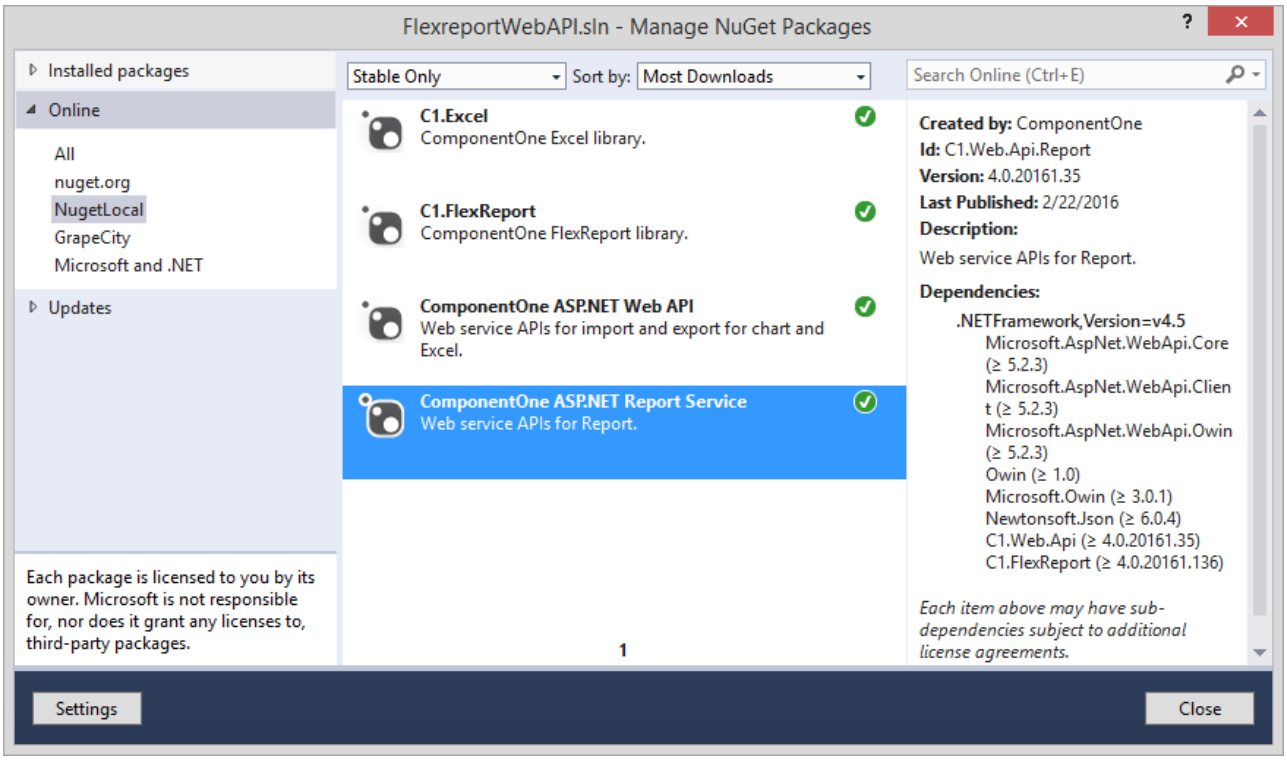
Complete the following steps to configure Web API project:

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

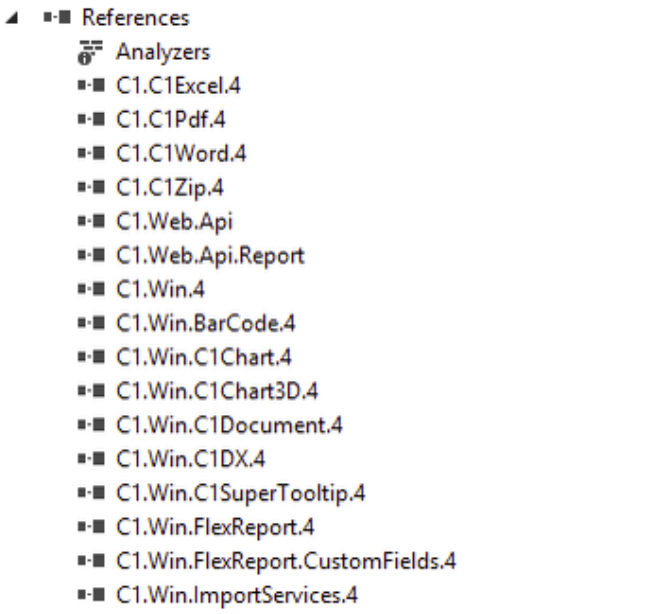
   If you have installed MVC Edition, the GrapeCity NuGet source path gets already set inside Visual Studio.



   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
```

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" />
```

Back to Top

Step 4: Deploy FlexReport Web API Service

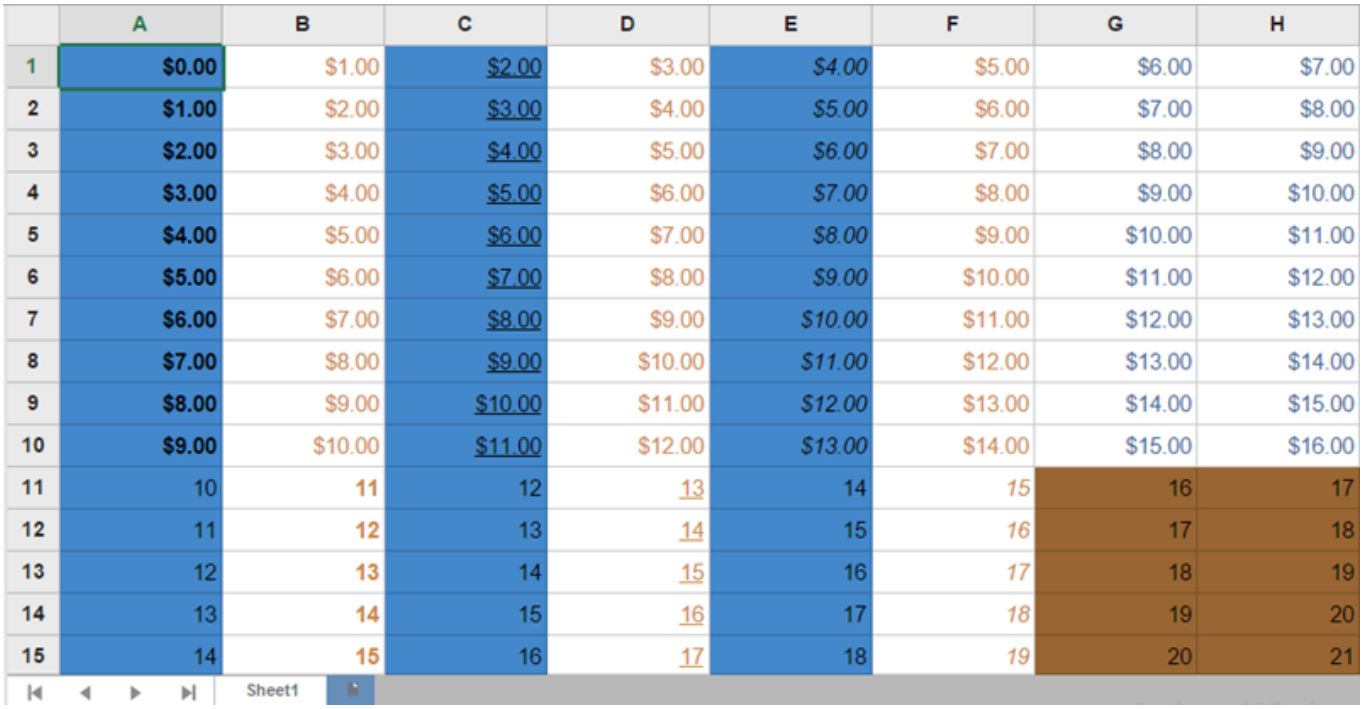
1. Compile the Project.
2. Deploy the project to IIS. The Web API URL, for service hosted on local IIS, will be http://localhost/FlexReportwebAPI/api/report.

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.

FlexSheet

Provide an Excel-like experience to your ASP.NET web application users with FlexSheet for MVC, a fast and lightweight control which extends the features of the FlexGrid. The control introduces basic spreadsheet functionality to your MVC application, enabling you to work with formulas and load or export excel files. It extends server side operations for remote data binding and loading of excel files from server. Moreover, it gives your end users the flexibility to display, edit and save the excel files on the client side.

Easily create (unbound or bound) FlexSheet applications and explore endless possibilities for effectively viewing, editing and analyzing your business data.



Key Features

- **Built-in Formula Support:** Work with nearly 100 built-in formulas supported by FlexSheet's calculation engine, together with the advantage of auto-completion. Our formula operations are aligned with Microsoft Excel, including the operations for aggregation, mathematics, logic and text.
- **Multiple Sheets:** Add multiple separated worksheets to your application, which are represented by tabs that flip between pages.
- **Cell Style:** Customize your FlexSheet cells by applying styles to each cell. You can easily set desired data format, font, horizontal alignment, fore color, fill color, etc. to the cells in FlexSheet.
- **Merge Cells:** Easily merge two or more adjacent cells of your FlexSheet control. The merged cells will contain data from upper-left most cell only. FlexSheet enables you to merge any group of adjacent cells, unlike FlexGrid which allows content-driven merging only.
- **Undo/Redo:** Undo/redo multiple operations of editing cells, inserting/removing rows or columns, applying cell styles, merging cells, resizing rows/columns, dragging and dropping rows/columns, etc.

- **Drag and Drop:** Drag and drop rows/columns to move or copy current rows/columns to another rows/columns.
- **Column Selection:** Select an entire column by simply clicking the column header.

Quick Start: Load Excel to FlexSheet

The FlexSheet control in your application can be unbound, bound to a data source, and can even display data loaded from an excel file. It offers you a flexible spreadsheet experience, allowing you to load existing excel file or workbook in FlexSheet, modify its data and even save it as an excel or a workbook file remotely or on the client side.

This section discusses how to use FlexSheet control in your application, and load an excel file or workbook instance in it on the server side.

This can be accomplished in the following steps.

- **Step 1: License your application**
- **Step 2: Add the relevant references to your application**
- **Step 3: Configure the application to use FlexSheet control**
- **Step 4: Register Resources**
- **Step 5: Add jszip.js library to the application**
- **Step 6: Add an excel file or a workbook to server**
- **Step 7: Add a FlexSheet Control**
- **Step 8: Build and Run the Project**

> **Note:** The **ComponentOne template** for ASP.NET MVC Edition automatically registers the required resources, and adds the relevant references and packages to your application. Therefore, you need not follow the Steps 1 to 5 above if your application is created using ComponentOne template.

The following image shows workbook data loaded from the server in the FlexSheet control.

	A	B	C	D	E	F	G	H	I	J	K	L
1										*For Office Use Only*		
2		**Expense Report**										
3												
4		PURPOSE:	On business			Attachment:	Yes			PAY PERIOD:	From	3/1/2015
5											To	4/1/2015
6		EMPLOYEE IMFORMATION:										
7		Name	Robert King			Position	Sales Repre...			SSN	A12345	
8		Department	Sales			Manager	Andrew Fuller			Employee ID	E123456	
9												
10		**Date**	**Account**	**Description**	**Hotel**	**Transport**	**Fuel**	**Meals**	**Phone**	**Entertainment**	**Misc**	**Total**
11		3/1/2015	12345678	Visit VIP customers.	$191.37	$64.01	$111.47	$69.93	$64.13	$199.19	$70.91	$771.01
12		3/3/2015	12345678	Visit VIP customers.	$174.44	$164.69	$188.21	$26.18	$45.83	$137.87	$107.89	$845.11
13		3/7/2015	12345678	Visit VIP customers.	$86.65	$70.11	$153.70	$182.37	$42.53	$180.13	$183.11	$898.60
14		3/11/2015	12345678	Visit VIP customers.	$157.00	$4.29	$173.18	$64.98	$104.66	$76.08	$74.66	$654.85
15		3/18/2015	12345678	Visit VIP customers.	$111.14	$61.48	$2.01	$185.03	$32.02	$183.17	$88.18	$663.03
16		3/21/2015	12345678	Visit VIP customers.	$193.97	$107.80	$163.20	$74.09	$149.50	$178.78	$60.38	$927.72
17		3/27/2015	12345678	Visit VIP customers.	$3.73	$128.74	$183.55	$46.56	$129.63	$173.46	$34.47	$700.14
18		**Total**			**$918.30**	**$601.12**	**$975.32**	**$649.14**	**$568.30**	**$1,128.68**	**$619.60**	**$5,460.46**
19											Subtotal	$4,460.46
20											Cash Advan...	$1,000.00
21		APPROVED:				NOTES:					Total	$5,460.46

Expense Report | Income Report

Step 1: License your application

Complete the following steps to license your application for using FlexSheet control.

ASP.NET MVC

1. In the **Solution Explorer**, double-click the project name (for example, **MVCFlexSheet**) and expand the node **Properties.**
2. Double-click the **licenses.licx** file to open it.

3. In the **licenses.licx** file, add the following:

licenses.licx

```
C1.Web.Mvc.LicenseDetector, C1.Web.Mvc
C1.Web.Mvc.Sheet.LicenseDetector, C1.Web.Mvc.FlexSheet
```

ASP.NET Core MVC

To generate the runtime license for applications created in Visual Studio, use **license generator Add-in** available in **Tools** menu in Visual Studio.

However, to generate license through C1 website:

1. Create a **License.cs** in your project (by right-clicking the **Project Name** in Solution Explorer and selecting **Add New | Class**).
2. Add the license key generated on C1 website in this class.
3. From the Solution Explorer, open **Startup.cs** file and assign the key to it.

For more information on how to add license to your application, refer to Licensing.

Back to Top

Step 2: Add the required references to your application

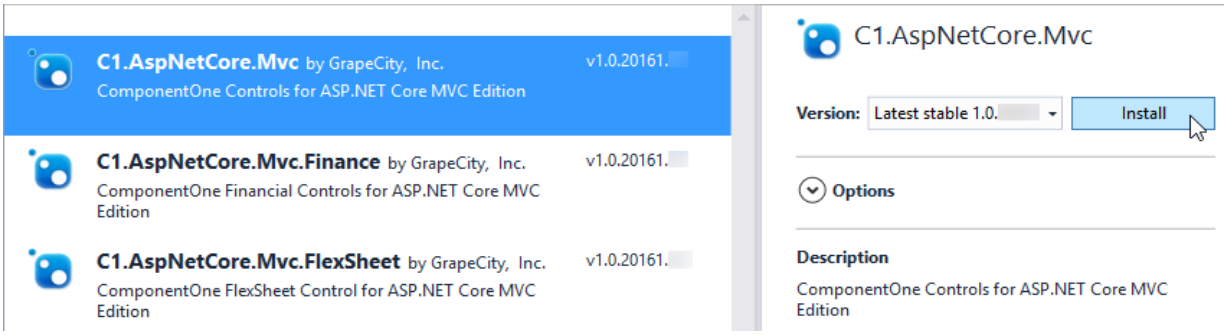
ASP.NET

Complete the following steps to add the ASP.NET MVC Edition references and FlexSheet references to your project.

1. In the **Solution Explorer**, right click **References** and select **Add Reference.**
2. Browse to the location- **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin.**
3. Select **C1.Web.Mvc.dll** and **C1.Web.Mvc.FlexSheet.dll**, and click **Add**.
4. Set the **Copy Local** property of the **C1.Web.Mvc.dll** and **C1.Web.Mvc.FlexSheet.dll** to **True**.

ASP.NET Core

1. Add the ASP.NET MVC Edition references to the project. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**. In **NuGet Package Manager**, select **GrapeCity** as the Package source. Search for **C1.AspNetCore.Mvc** package, and click **Install**. Refer to Configuring NuGet Package Sources for information on manually configuring GrapeCity NuGet source.



> **Note:** "C1.AspNetCore.Mvc" gets added under the "dependencies" within project.json file of your project once you restore the packages.

2. To work with **FlexSheet** control in your application, add **C1.AspNetCore.Mvc.FlexSheet** package. Once you restore the packages, "C1.AspNetCore.Mvc.FlexSheet" gets added under the "dependencies" in project.json file.

Back to Top

Step 3: Configure the application to use FlexSheet control

Complete the following steps to configure your application to use Financial Charts.

ASP.NET

1. From the **Solution Explorer**, expand the folder **Views** and double click the `web.config` file to open it.
2. Add the following markups in <namespaces></namespaces> tags, within the `<system.web.webPages.razor>` `</system.web.webPages.razor>` tags.

HTML

```
<add namespace="C1.Web.Mvc" />
<add namespace="C1.Web.Mvc.Fluent" />
<add namespace="C1.Web.Mvc.Sheet" />
<add namespace="C1.Web.Mvc.Sheet.Fluent" />
```

ASP.NET Core

1. From the **Solution Explorer**, expand the folder **Views** and double click the _ViewImports.cshtml file to open it.
2. Add the following references to use FlexSheet controls in your ASP.NET Core application.

_ViewImports

```
@addTagHelper *, C1.AspNetCore.Mvc

@addTagHelper *, C1.AspNetCore.Mvc.FlexSheet
```

Back to Top

Step 4: Register Resources

Complete the following steps to register the required resources for using FlexSheet control.

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click `_Layout.cshtml` to open it.
3. Add the following code between the `<head></head>` tags.

ASP.NET MVC

_Layout.cshtml

```
@Html.C1().Styles()@Html.C1().Scripts().Basic().FlexSheet()
```

ASP.NET Core MVC

_Layout.cshtml

```
<c1-styles />    <c1-scripts>        <c1-flex-sheet-scripts />    </c1-scripts>
```

For more information on how to register resources, refer to Registering Resources.

Back to Top

Step 5: Add jszip.js library to the application or use its CDN link

Loading Excel files in FlexSheet and saving FlexSheet data to Excel files have dependency on **jszip.min.js** file. Therefore, you need to add it in your application, and provide reference to it in the respective view or in <head> section of **_Layout.cshtml** file. Complete the following steps to add the **jszip.min.js** file to your application.

1. Download the **jszip.min.js** file from the CDN link: http://cdnjs.cloudflare.com/ajax/libs/jszip/2.2.1/jszip.min.js
2. Place the jszip.min.js file in the **Scripts** folder of your application.
3. Right-click the **Scripts** folder in Solution Explorer, and select **Add | Existing Item...** from the options.

4. In the **Add Existing Item** dialog, browse to the location of **jszip.min.js** in your project.
5. Select the **jszip.min.js** file and click **Add** to include it in your project.

Back to Top

Step 6: Add an excel file or a workbook to server

1. Make sure that the desired excel or workbook file to be loaded is placed on server. In this case, we have placed the file in the **Content** folder of our application.

   > Note that, this example considers Visual Studio IIS (Internet Information Services) of your machine as the Server and browser as the client.

2. Right-click your project name in the Solution Explorer, and select **Add** | **Add Existing Files** from the options, to add the excel file in your project.

Back to Top

Step 7: Add a FlexSheet Control to your Application

Complete the following steps to initialize a FlexSheet control.

Add a new Controller

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: `Default1Controller`).
   3. Click **Add**.
4. Include the MVC references as shown below.

```
C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

5. Replace the method **Index()** by the following code.

   ServerLoadController.cs

C#

```
C#                                                                copyCode
// GET: ServerLoad
public ActionResult Index()
{
    return View();
}
```

VB

```
VB
' GET: ServerLoad
Public Function Index() As ActionResult
        Return View()
End Function
```

Add a View for the controller:

1. Within the Controller, which was added in the above step, place the cursor inside the method `Index()`.
2. Right click and select **Add View**. The **Add View** dialog appears.
3. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
4. Click **Add**. A view has been added for the controller.
5. In the Solution Explorer, double click `Index.cshtml` to open it.
6. Replace the default code of the **Views\Index.cshtml** file with the one given below to initialize the **FlexSheet** control.

HTML Helpers

Index.cshtml copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<div>

@(Html.C1().FlexSheet().Load("~/Content/ExcelFiles/WorkBook.xlsx").Width(1500).Height(800)
    )
</div>
```

Index.vbhtml

```
@Imports C1.Web.Mvc
@Imports C1.Web.Mvc.FlexSheet.Fluent
@Imports C1.Web.Mvc.FlexSheet
@Imports C1.Web.Mvc.Grid
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
    @(Html.C1().FlexSheet() _
                .Load("~/Content/ExcelFiles/Product.xlsx") _
                .Width("1500px") _
                .Height("800px")
    )
```

Tag Helpers

HTML copyCode

```
<script src=
"http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>

<div>
    <c1-flex-sheet class="flexSheet" file-path="~/FilestoLoad
/Workbook.xlsx" height="800px" width="1500px"></c1-flex-sheet>
</div>
```

Back to Top

Step 8: Build and Run the Project

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

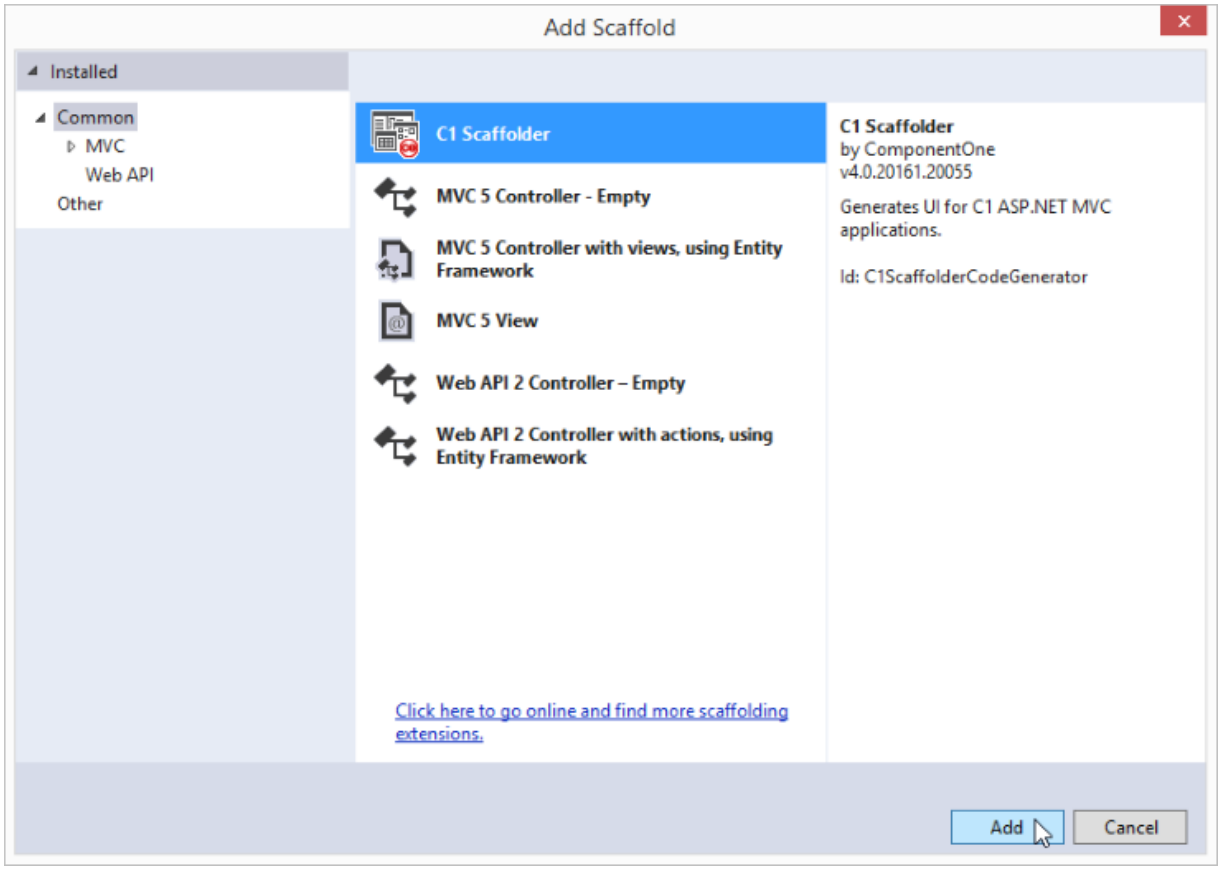
Back to Top

Features

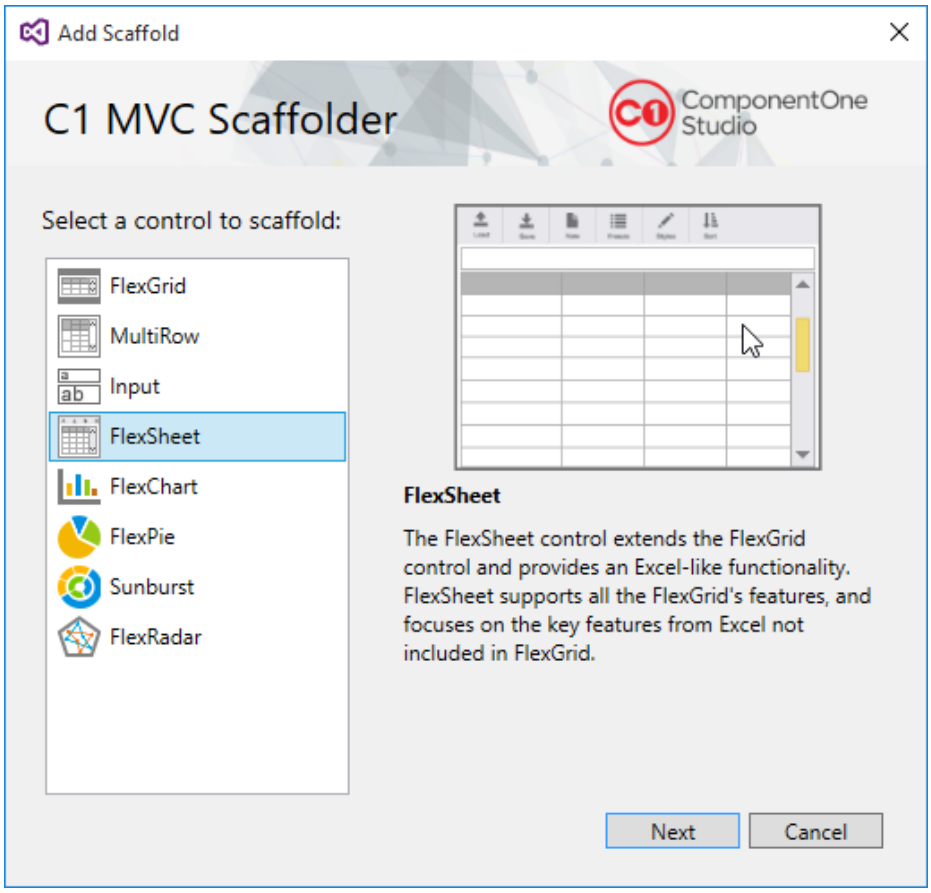
Scaffolding

The steps to scaffold **ComponentOne FlexSheet** control for ASP.NET MVC are as follows:

1. Configure the datasource.To configure data source in MVC application, see Data Source Configuration.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane. You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

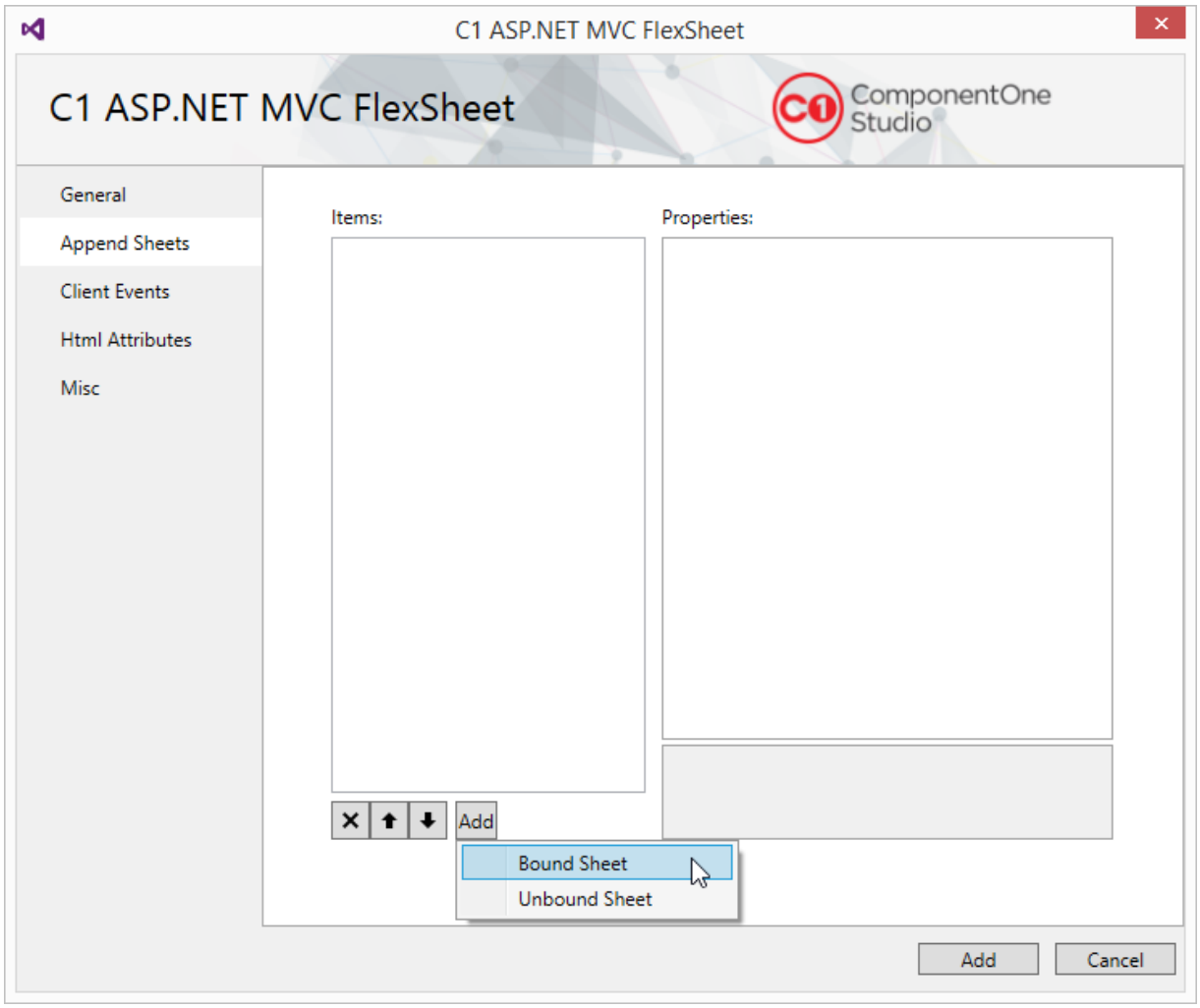


4. Click **Add**.
5. In the **Add Scaffold** dialog, select FlexSheet control, and the click **Next**.

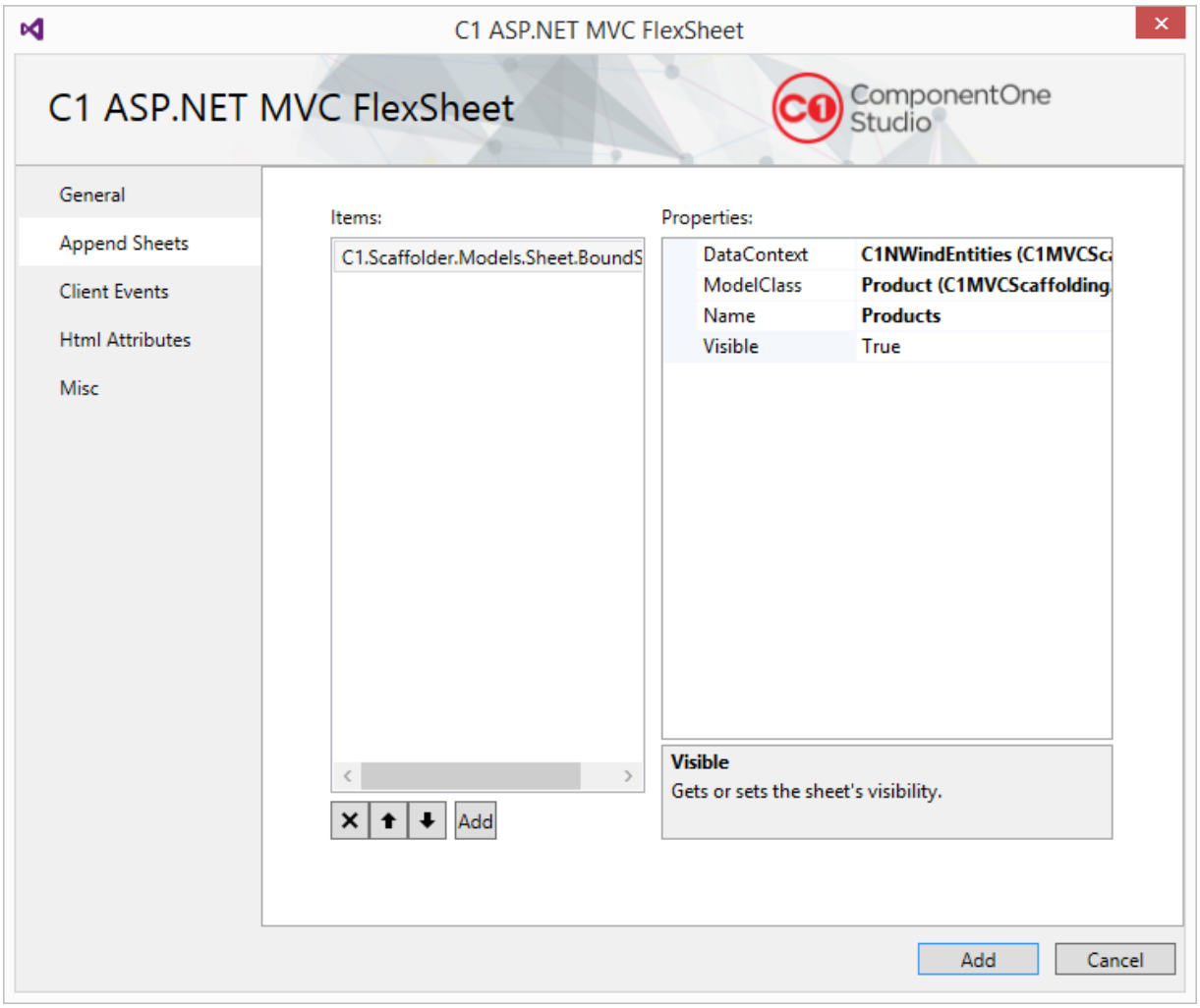


The **C1 ASP.NET MVC FlexSheet** wizard appears with the General tab selected by default.

6. Go to the **Append Sheets** tab, to add a sheet in the FlexSheet control. You can add, delete, or move sheets upward or downward in the sequence.
7. To add a new sheet, click **Add**, and then select **Bound Sheet**.



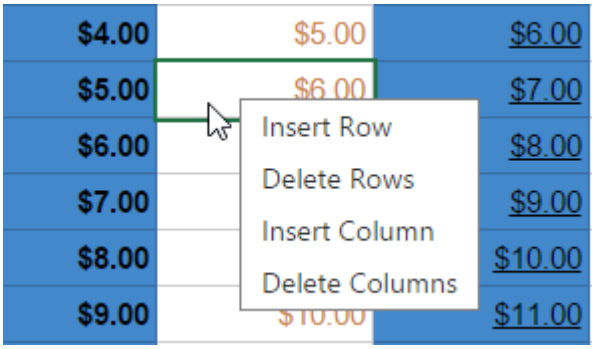
8. Specify the following sheet properties
    1. Select **Data Context Class** from the drop-down list. In our case, we select C1NwindEntities.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in **Step 1**. In our case, we select Product to populate list of products in the FlexSheet.
    3. Enter a **Name** for the FlexSheet. In our we case, its **Products**.



9. Go to **Client Events** tab, and check **AutoSizedColumn** and **AutoSizedRow** check boxes.
10. Go to **Misc** tab, and check the **Show Formula Bar** check box to include the Formula Bar in your FlexSheet. You can also set different options from the Misc tab. For example, Allow Dragging, AllowMerging, Selection Mode, and more.
11. Click **Add**. You will see that the Controller and View for the selected model are added to your project.

Context Menu

The FlexSheet control provides familiar features as supported by Microsoft Excel, and additional capabilities to enhance user experience. Context Menu in FlexSheet is one of these excel inspired features. Triggered by right-click, it enables you to insert or remove rows or columns in your sheet.



Upon right-clicking any cell or row/column header in FlexSheet, Context Menu pops up which allows you to perform following operations on your sheet.

Context Menu Operations	Description
Insert Row	Adds a new row above the active row
Delete Rows	Deletes the active row
Insert Column	Adds a new column before the active column
Delete Columns	Deletes the active column

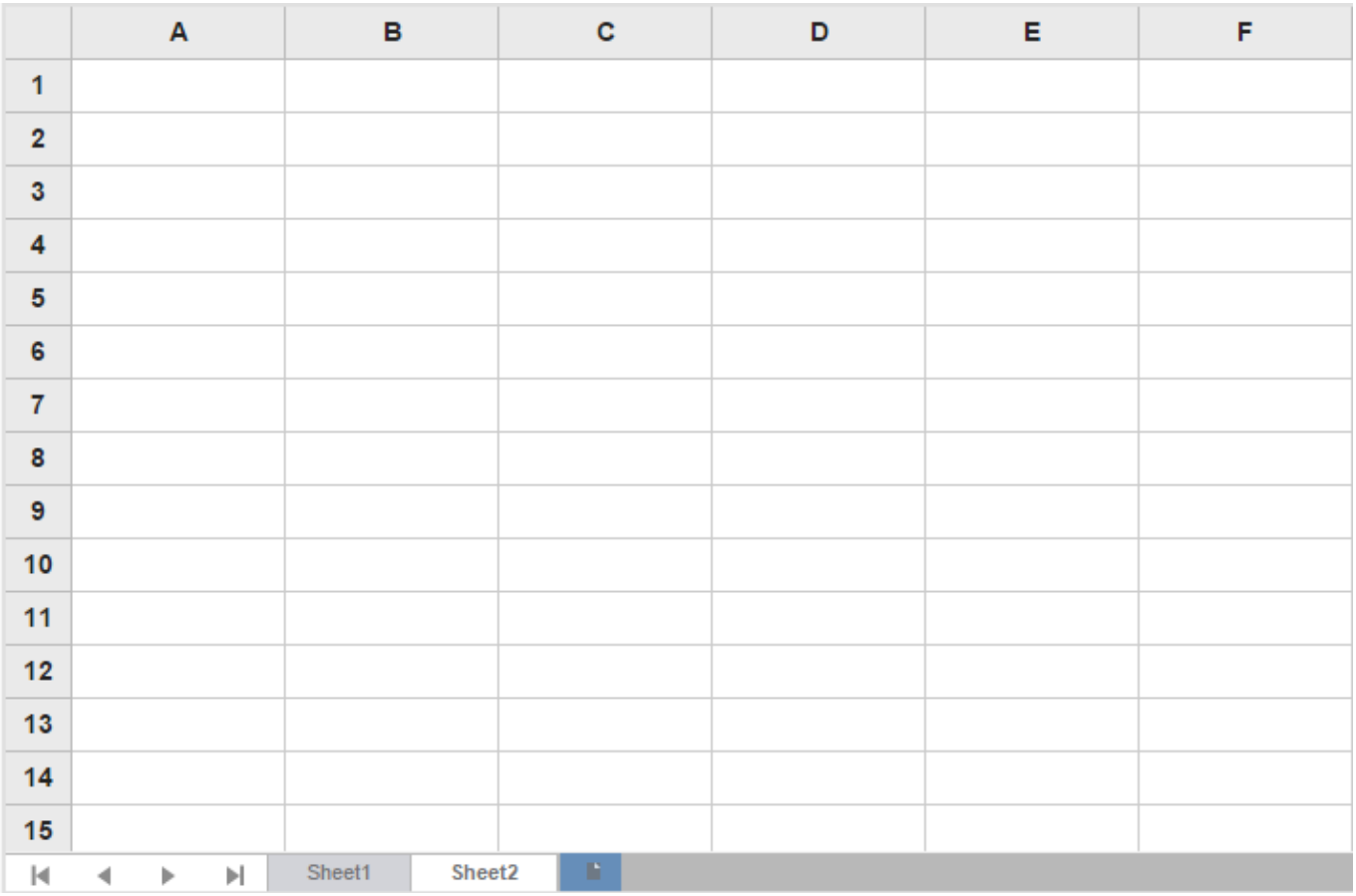
Back to Top

Unbound Sheets

The FlexSheet control in your application can be unbound, or bound to a data source. You can create unbound FlexSheet in an MVC application through code using builder method **AddUnboundSheet**. Sheet name along with the desired row count and column count are specified as method parameters.

End users can populate data manually in each cell of an unbound FlexSheet, or set cell values programmatically. Such a sheet can be used as an application form, where an applicant is asked to provide data in various fields.

The following image shows an MVC FlexSheet in unbound mode.



The following code examples demonstrate how to add unbound FlexSheet to an MVC application:

In Code

UnboundController.cs

C# copyCode

```
// GET: Unbound
public ActionResult Index()
{
    return View();
}
```

Unbound.cshtml

HTML Helpers

Razor copyCode

```
<div>
    @(Html.C1().FlexSheet().CssClass("flexSheet")
    .AddUnboundSheet("Sheet1", 25, 12)
    .AddUnboundSheet("Sheet2", 25, 12))
</div>
```

Tag Helpers

HTML copyCode

```
<div>
    <c1-flex-sheet class="flexSheet">
        <c1-unbound-sheet column-count="12" row-count="25"></c1-unbound-sheet>
        <c1-unbound-sheet column-count="12" row-count="25"></c1-unbound-sheet>
    </c1-flex-sheet>
</div>
```

Back to Top

Setting Data on Client-side

FlexSheet supports conditional formatting on the client side, that allows you to change the formatting of a cell or a range of cells depending on the value of the cell(s) or formula applied on the cell(s). Applying conditional formatting to FlexSheet control empowers you to boost its capabilities, thereby enabling you to do more than what you could do using its built-in functionality.

For example, while maintaining expenses and sales record for your retail store, you can easily track when your actual expenses are shooting over the budgeted expenses. The below example demonstrates this scenario.

The following image shows FlexSheet control displaying expenses details for a retail store, with conditional formatting applied on cells having values greater than 1000.
<image>

The following code examples demonstrate how to set data in FlexSheet on the client side.

In Code

SetDataController.cs

C# copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

SetData.cshtml

HTML Helpers

Razor copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Tag Helpers

HTML copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Back to Top

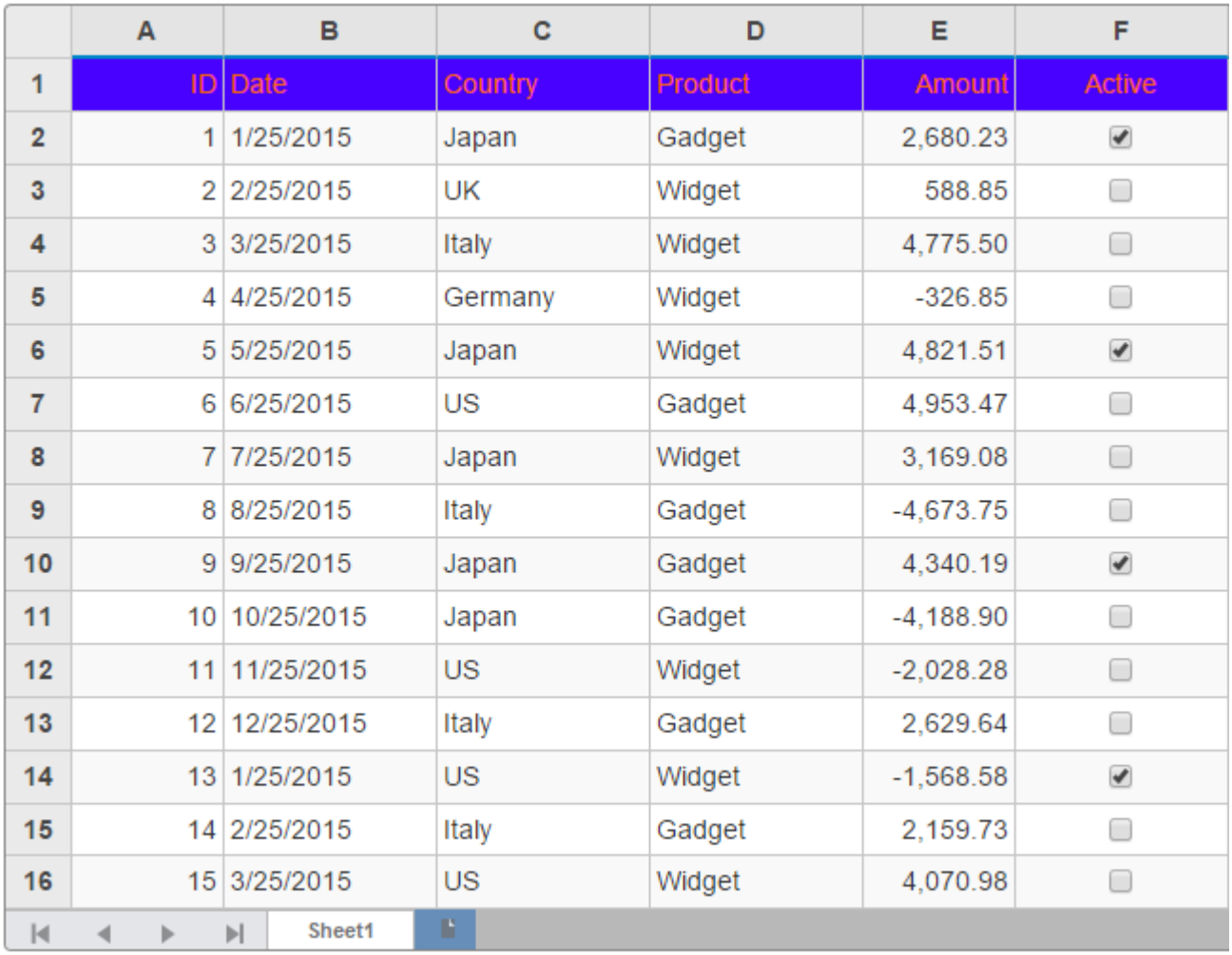
Data Binding

FlexSheet supports model binding and remote data binding where you can retrieve data directly using C1JsonRequest. In the bound mode, columns can be defined and FlexSheet is bound to a data source just like FlexGrid.

In remote binding, your sheet is bound to a collection by passing **Action URL** method, using **Bind** property. Remote data binding specifies remote data URLs, which include server, table and columns. The returned arrays serve as data sources for CollectionView objects. The CollectionViewHelper class aids collections to have editing, filtering, grouping, and sorting services.

This topic demonstrates how to retrieve data from an existing data source. This is useful for developing data-intensive applications and scenarios for representing data as dashboards.

The following image shows how the FlexSheet appears in bound mode to fetch data from the model Sale.cs.



	A	B	C	D	E	F
1	ID	Date	Country	Product	Amount	Active
2	1	1/25/2015	Japan	Gadget	2,680.23	☑
3	2	2/25/2015	UK	Widget	588.85	☐
4	3	3/25/2015	Italy	Widget	4,775.50	☐
5	4	4/25/2015	Germany	Widget	-326.85	☐
6	5	5/25/2015	Japan	Widget	4,821.51	☑
7	6	6/25/2015	US	Gadget	4,953.47	☐
8	7	7/25/2015	Japan	Widget	3,169.08	☐
9	8	8/25/2015	Italy	Gadget	-4,673.75	☐
10	9	9/25/2015	Japan	Gadget	4,340.19	☑
11	10	10/25/2015	Japan	Gadget	-4,188.90	☐
12	11	11/25/2015	US	Widget	-2,028.28	☐
13	12	12/25/2015	Italy	Gadget	2,629.64	☐
14	13	1/25/2015	US	Widget	-1,568.58	☑
15	14	2/25/2015	Italy	Gadget	2,159.73	☐
16	15	3/25/2015	US	Widget	4,070.98	☐

Sheet1

The following code examples demonstrate how to bind FlexSheet to fetch data from remote data source:

In Code

Add a Sale.cs class to the Models folder.

Model

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 double Amount { get; set; }
    public bool Active { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "Germany", "UK",
"Japan", "Italy", "Greece" };
    private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget",
"Doohickey" };
```

```
    /// <summary>
    /// Get the data.
    /// </summary>
    /// <param name="total"></param>
    /// <returns></returns>
    public static IEnumerable<Sale> GetData(int total)
    {
        var rand = new Random(0);
        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 date = new DateTime(2015, i % 12 + 1, 25);

            return new Sale
            {
                ID = i + 1,
                Date = date,
                Country = country,
                Product = product,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Active = (i % 4 == 0)
            };
        });
        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;
    }
}
```

DataBindingController.cs

C# copyCode

```
public class DataBindingController : Controller
{
    public static List<Sale> SALES = Sale.GetData(50).ToList();
```

```
    public ActionResult RemoteBind([C1JsonRequest] CollectionViewRequest<Sale>
requestData)
    {
        return this.C1Json(CollectionViewHelper.Read(requestData, Sale.GetData(50)));
    }

    public ActionResult Index()
    {
        return View(SALES);
    }
}
```

DataBinding.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<Sale>

<div>
    @(Html.C1().FlexSheet().CssClass("flexSheet").Height("700px").Width("700px")
          .AddBoundSheet(cv =>
                              cv.Bind(Url.Action("RemoteBind")))
    )
</div>
```

Tag Helpers

HTML copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<Sale>

<div>
    <c1-flex-sheet class="flexSheet" height="700px" width="700px">
        <c1-bound-sheet>
            <c1-items-source read-action-url="@Url.Action("RemoteBind")">
            </c1-items-source>
        </c1-bound-sheet>
    </c1-flex-sheet>
</div>
```

Back to Top

Local Model Binding in FlexSheet

Data Binding in FlexSheet can also be accomplished by passing a model locally in Bind property, unlike remote binding where a URL is passed.

The following code examples demonstrate how to bind FlexSheet control to fetch data from local data source:

In Code

Create a Sale.cs model class in the Models folder, as discussed **above**.

ModelBindingController.cs

C# copyCode

```
public partial class ModelBindController : Controller
{
    // GET: ModelBind
    public static List<Sale> SALES = Sale.GetData(15).ToList();

    public ActionResult ModelDBIndex()
    {
        return View(SALES);
    }
}
```

ModelBinding.cshtml

HTML Helpers

Razor copyCode

```
@model IEnumerable<Sale>
<div>
    @(Html.C1().FlexSheet().CssClass("flexSheet").Id("boundSheet")
        .AddBoundSheet(sheet =>
            sheet.Bind(cv =>
                            cv.Bind(Model).DisableServerRead(true)))
                    )
</div>
```

Tag Helpers

HTML copyCode

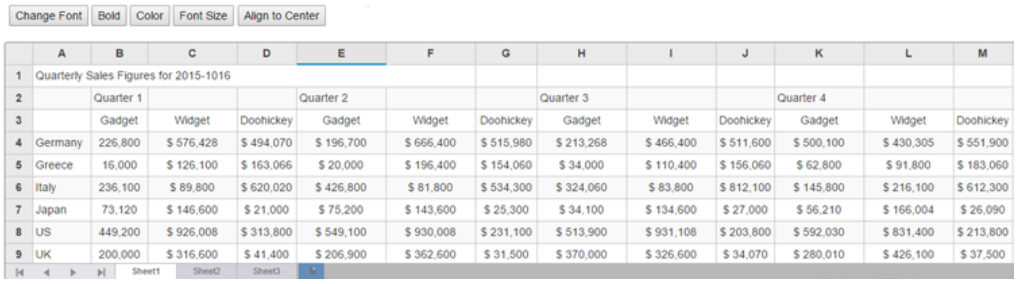
```
@model IEnumerable<sale>
    <div>
        <c1-flex-sheet class="flexSheet" id="boundSheet">
            <c1-bound-sheet>
                <c1-items-source source-collection="Model"></c1-items-source>
            </c1-bound-sheet>
        </c1-flex-sheet>
    </div>
```

Format Cells

FlexSheet enables you to format its cells and content in these cells on the client-side. You can easily apply desired fill color on each cell, and change the font's size, style, color, or weight (bold/unbold). Additionally, underline, italicize or align the cells' text in the run-time. Moreover, it gives you the ability to set the desired display format for data in each cell, to Decimal, Number, Percentage or Currency formats.

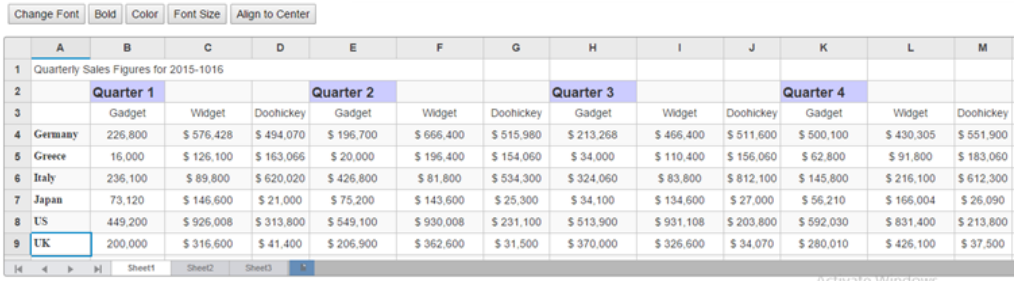
Customize your FlexSheet control to change the appearance of the displayed data, and make the data stand out in a huge worksheet. For example, you might want to categorize region wise sales figures for your organization on quarterly basis for comparison. This is achieved through **applyCellsStyle()** method. The below example demonstrates this scenario on a FlexSheet control which displays hypothetical quarterly sales data for three products in different countries for the year 2015-16.

The following image shows a FlexSheet with buttons to change the formatting of the cells and data in them.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Quarterly Sales Figures for 2015-1016												
2		Quarter 1			Quarter 2			Quarter 3			Quarter 4		
3		Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey
4	Germany	226,800	$ 576,428	$ 494,070	$ 196,700	$ 666,400	$ 515,980	$ 213,268	$ 466,400	$ 511,600	$ 500,100	$ 430,305	$ 551,900
5	Greece	16,000	$ 126,100	$ 163,066	$ 20,000	$ 196,400	$ 154,060	$ 34,000	$ 110,400	$ 156,060	$ 62,800	$ 91,800	$ 183,060
6	Italy	236,100	$ 89,800	$ 620,020	$ 426,800	$ 81,800	$ 534,300	$ 324,060	$ 83,800	$ 812,100	$ 145,800	$ 216,100	$ 612,300
7	Japan	73,120	$ 146,600	$ 21,000	$ 75,200	$ 143,600	$ 25,300	$ 34,100	$ 134,600	$ 27,000	$ 56,210	$ 166,004	$ 26,090
8	US	449,200	$ 926,008	$ 313,800	$ 549,100	$ 930,008	$ 231,100	$ 513,900	$ 931,108	$ 203,800	$ 592,030	$ 831,400	$ 213,800
9	UK	200,000	$ 316,600	$ 41,400	$ 206,900	$ 362,600	$ 31,500	$ 370,000	$ 326,600	$ 34,070	$ 280,010	$ 426,100	$ 37,500

The below image shows how the FlexSheet appears after applying formatting to its cells.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Quarterly Sales Figures for 2015-1016												
2		Quarter 1			Quarter 2			Quarter 3			Quarter 4		
3		Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey
4	Germany	226,800	$ 576,428	$ 494,070	$ 196,700	$ 666,400	$ 515,980	$ 213,268	$ 466,400	$ 511,600	$ 500,100	$ 430,305	$ 551,900
5	Greece	16,000	$ 126,100	$ 163,066	$ 20,000	$ 196,400	$ 154,060	$ 34,000	$ 110,400	$ 156,060	$ 62,800	$ 91,800	$ 183,060
6	Italy	236,100	$ 89,800	$ 620,020	$ 426,800	$ 81,800	$ 534,300	$ 324,060	$ 83,800	$ 812,100	$ 145,800	$ 216,100	$ 612,300
7	Japan	73,120	$ 146,600	$ 21,000	$ 75,200	$ 143,600	$ 25,300	$ 34,100	$ 134,600	$ 27,000	$ 56,210	$ 166,004	$ 26,090
8	US	449,200	$ 926,008	$ 313,800	$ 549,100	$ 930,008	$ 231,100	$ 513,900	$ 931,108	$ 203,800	$ 592,030	$ 831,400	$ 213,800
9	UK	200,000	$ 316,600	$ 41,400	$ 206,900	$ 362,600	$ 31,500	$ 370,000	$ 326,600	$ 34,070	$ 280,010	$ 426,100	$ 37,500

The following code examples demonstrate how to apply styles in FlexSheet.

In Code

FormatController.cs

C# copyCode

```
public class FormatController : Controller
{
    // GET: Format
    public ActionResult FormatIndex()
    {
        return View();
    }
}
```

FormatCells.cshtml

HTML Helpers

Razor copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script type="text/javascript">
    function setBold() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontWeight: 'bold' });
    }
    function changeFont() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontFamily: 'Unicode' });
    }
    function changeFontSize() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontSize: '18px' });
    }
    function setColor() {
        var flex = wijmo.Control.getControl("#fSheet");
```

```
        flex.applyCellsStyle({ backgroundColor: '#CCCCFF' });
    }

    function align() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ textAlign: 'Right' });
    }

</script>
<div>
    <input id="font" type="button" onclick="changeFont()" value="Change Font" />
    <input id="bold" type="button" onclick="setBold()" value="Bold" />
    <input id="cellColor" type="button" onclick="setColor()" value="Color" />
    <input id="fontSize" type="button" onclick="changeFontSize()" value="Font Size" />
    <input id="textAlign" type="button" onclick="align()" value="Align to Center" />
    <br /><br />

@(Html.C1().FlexSheet().Height("500px").Width("1500px").Load("~/Content/FlexSheet/QuarterlyData.xlsx").Id("fSheet")
    )
</div>
```

Tag Helpers

HTML copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script type="text/javascript">
    function setBold() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontWeight: 'bold' });
    }
    function changeFont() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontFamily: 'Unicode' });
    }
    function changeFontSize() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ fontSize: '18px' });
    }
    function setColor() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ backgroundColor: '#CCCCFF' });
    }
    function align() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.applyCellsStyle({ textAlign: 'Right' });
    }
</script>

<div>
    <input id="font" type="button" onclick="changeFont()" value="Change Font" />
    <input id="bold" type="button" onclick="setBold()" value="Bold" />
    <input id="cellColor" type="button" onclick="setColor()" value="Color" />
    <input id="fontSize" type="button" onclick="changeFontSize()" value="Font Size" />
    <input id="textAlign" type="button" onclick="align()" value="Align to Center" />
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="fSheet" Height="500" Width="1500" file-
path="~/Content/FlexSheet/QuarterlyData.xlsx">
    </c1-flex-sheet>
</div>
```

Explore the detailed demo samples for formatting cells in FlexSheet at:

- More about formatting FlexSheet cells

Back to Top

Conditional Formatting

FlexSheet supports conditional formatting on the client side, that allows you to change the formatting of a cell or a range of cells depending on the value of the cell(s) or formula applied on the cell(s). Applying conditional formatting to FlexSheet control empowers you to boost its capabilities, thereby enabling you to do more than what you could do using its built-in functionality.

For example, while maintaining expenses and sales record for your retail store, you can easily track when your actual expenses are shooting over the budgeted expenses. The below example demonstrates this scenario.

The following image shows FlexSheet control displaying expenses details for a retail store, with conditional formatting applied on cells having values greater than 1000.
<image>

The following code examples demonstrate how to add conditional formatting in FlexSheet.

In Code

ConditionalFormatController.cs

C# copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

ConditionalFormatting.cshtml

HTML Helpers

Razor copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Tag Helpers

HTML copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

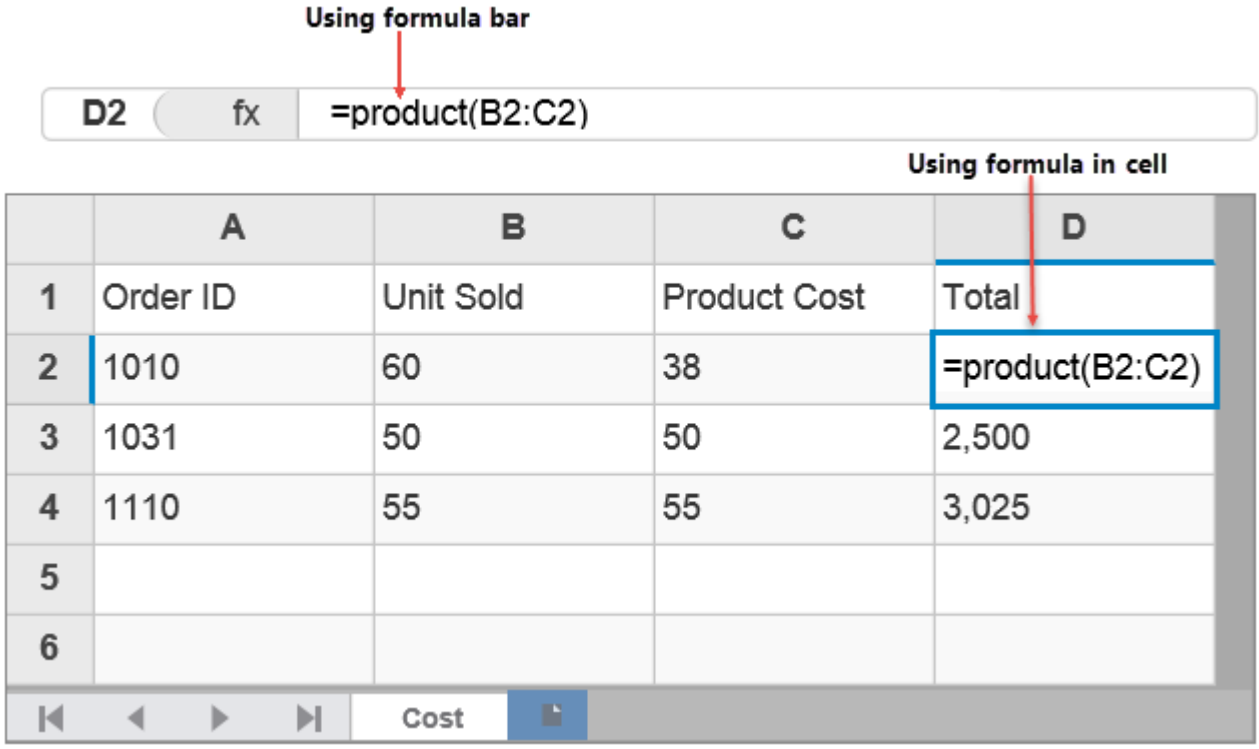
Back to Top

Formulas in FlexSheet

FlexSheet provides in-built auto-completion and formula support. You can use nearly 100 formulas to perform calculations and manipulate data in your sheet. You can use these formulas to manipulate data and perform calculations on the data in your sheet. FlexSheet provides you the flexibility to perform various mathematical operations ranging from finding totals to complex engineering problem.

Formula Bar at the top of the FlexSheet can be used to enter or copy an existing formula in to cells. It is labeled with function symbol (fx). By clicking the Formula Bar, or when you type an equal (=) symbol in a cell, the Formula Bar will activate. You can use the formulas in cells, just like excel. The formula operations in FlexSheet range from Basic

Operations to Mathematical, Logical, Text, Aggregate, Date, Lookup and Reference, and Financial.

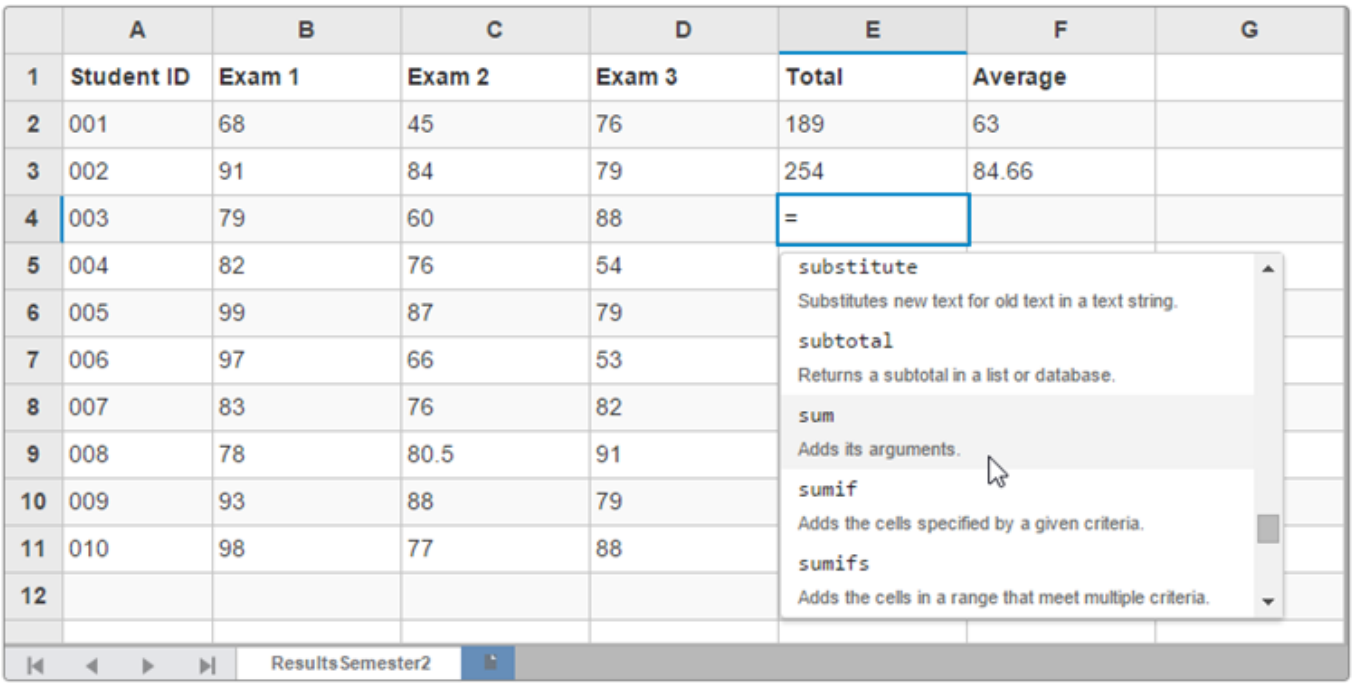


Using Formulas at Run-time

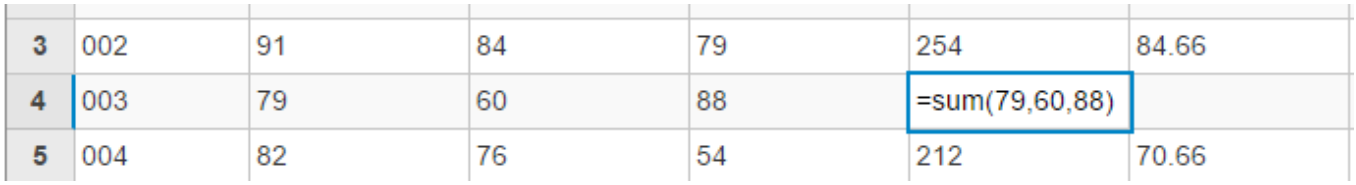
The FlexSheet control, just like excel, does more than simply displaying the entered or bound data. You can explore immense potential of built-in functions and perform complex calculations on your data, thereby utilizing FlexSheet as a comprehensive reporting and analysis tool.

This topic provides examples demonstrating how to work on data in FlexSheet control through in-built formulas. Using pre-defined formulas in FlexSheet is as simple as performing basic Context Menu operations. You simply select a cell, type an equal (=) sign through keyboard, and a popup appears which prompts you to select the desired formula from a list of pre-defined ones.

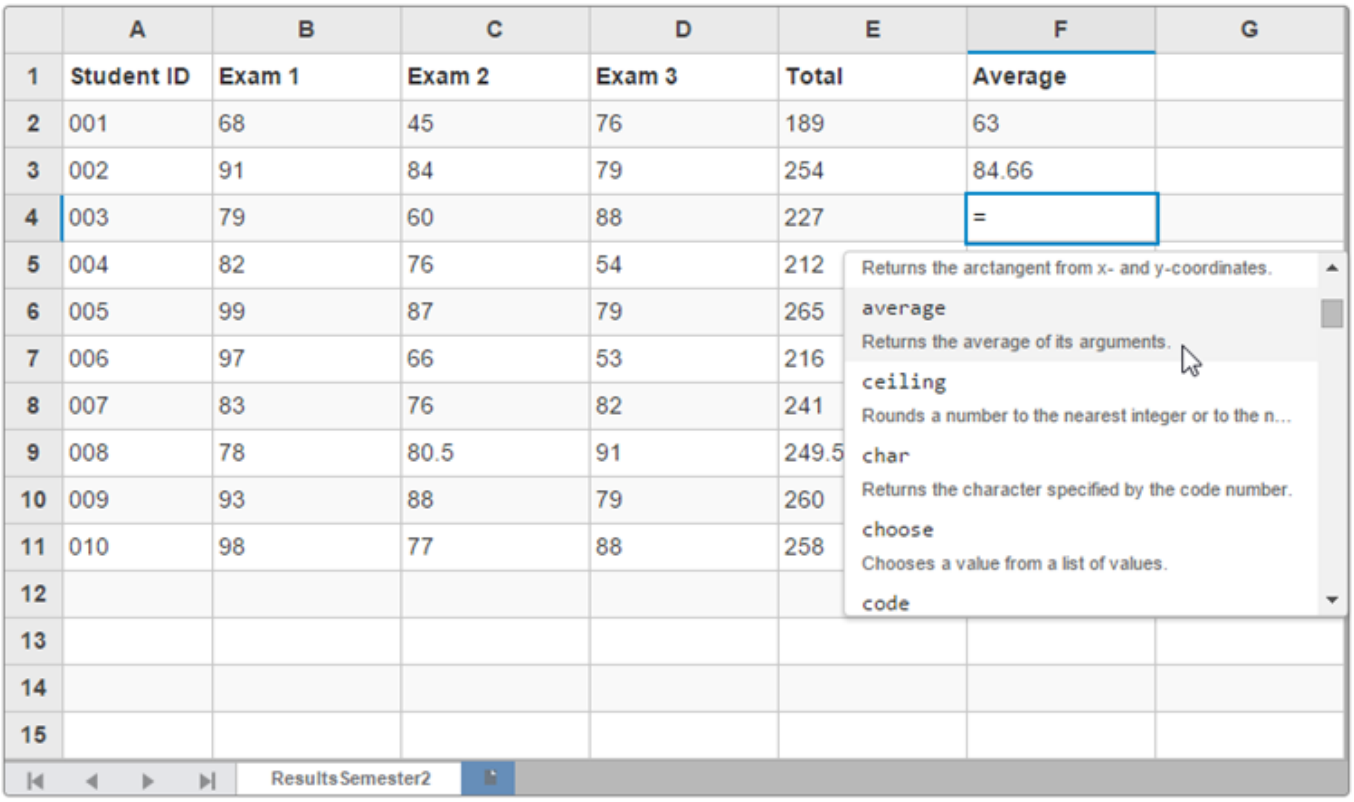
For example, the following image shows FlexSheet control displaying scores (out of 100) of 10 engineering students of a batch in three semester examinations. The examples below demonstrate how to use formulas to calculate sum and averages of their scores.



The following image shows how to use formula to calculate the sum of scores in the cell E4.



Similarly, below we average the scores of Student ID 003 in the cell F4.



	A	B	C	D	E	F
1	Student ID	Exam 1	Exam 2	Exam 3	Total	Average
2	001	68	45	76	189	63
3	002	91	84	79	254	84.66
4	003	79	60	88	227	=average(79,60,88)

In the following image, the **Ceiling** formula rounds the value in the cell C9 to a nearest integer.

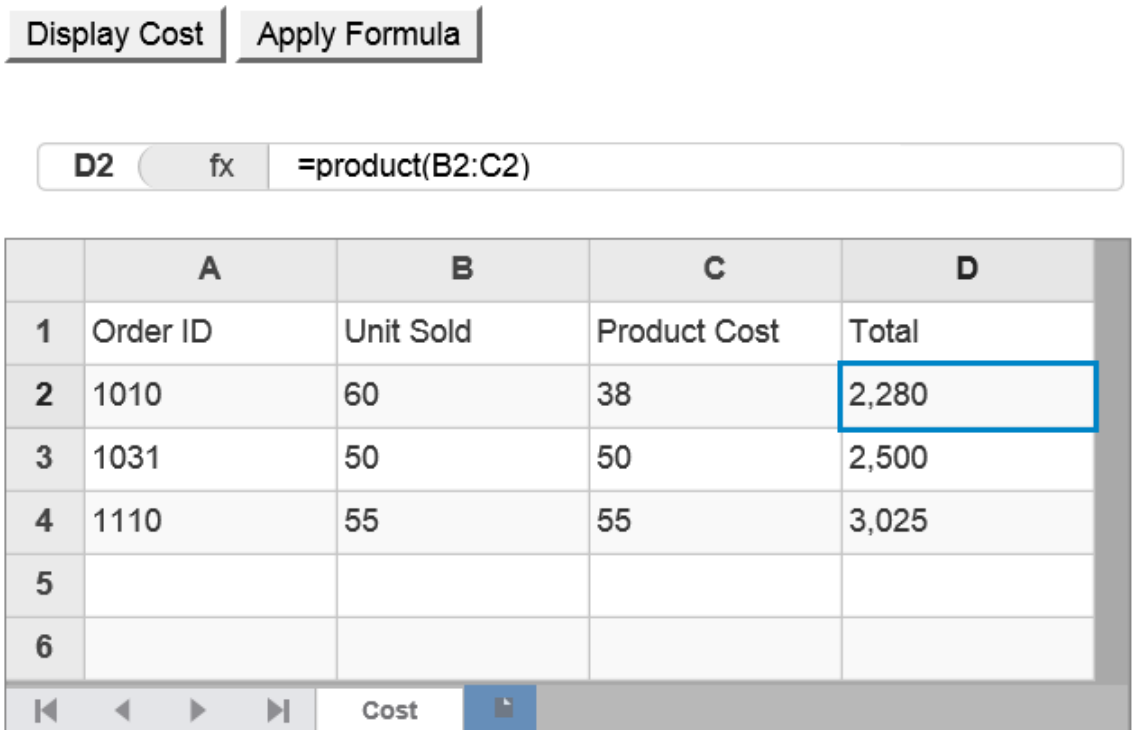
Explore the detailed demo samples for using predefined formulas in-cell at:

- Working with formulas in FlexSheet

Using Formulas through Code

FlexSheet allows adding formulas within cells programmatically, apart from using them at run-time. This topic demonstrates how to use pre-defined formulas supported in FlexSheet through code at the client side.

The following image shows a FlexSheet control which displays Unit Sold and Product Cost of a product. Here, formula is applied programmatically to calculate the total cost for each order. This is done through **setCellData()** method on the client side.



The following code examples demonstrate how to use formula programmatically in FlexSheet.

In Code

FormulaController.cs

C# copyCode

```
public class FormulaController : Controller
{
    // GET: Formula
    public ActionResult FormulaCodeIndex()
    {
        return View();
    }
}
```

FormulaIndex.cshtml

HTML Helpers

Razor copyCode

```
<script>

    function generateCostSheet() {
        var flex = wijmo.Control.getControl("#formulaSheet");
        flex.setCellData(0, 0, "Order ID");
        flex.setCellData(0, 1, "Unit Sold");
        flex.setCellData(1, 0, "1010");
        flex.setCellData(2, 0, "1031");
        flex.setCellData(3, 0, "1110");
        flex.setCellData(1, 1, "60");
        flex.setCellData(2, 1, "50");
        flex.setCellData(3, 1, "55");
        flex.setCellData(0, 2, "Product Cost");
        flex.setCellData(1, 2, "38");
        flex.setCellData(2, 2, "50");
        flex.setCellData(3, 2, "55");
        flex.setCellData(0, 3, "Total");
    }
    function setFormula() {
        var flex = wijmo.Control.getControl("#formulaSheet");
        flex.setCellData(1, 3, "=product(B2:C2)");
        flex.setCellData(2, 3, "=product(B3:C3)");
        flex.setCellData(3, 3, "=product(B4:C4)");
    }
</script>
<div>
    <input id="bold" type="button" onclick="generateCostSheet()" value="Display Cost" />
    <input id="textAlign" type="button" onclick="setFormula()" value="Apply Formula" />

@(Html.C1().FlexSheet().CssClass("flexSheet").Id("formulaSheet").Height("220px").Width("500px")
    .AddUnboundSheet("Cost", 6, 4).IsReadOnly(false).ShowFormulaBar())
</div>
```

Tag Helpers

HTML copyCode

```
<script>
```

```
    function generateCostSheet() {
        var flex = wijmo.Control.getControl("#formulaSheet");
        flex.setCellData(0, 0, "Order ID");
        flex.setCellData(0, 1, "Unit Sold");
        flex.setCellData(1, 0, "1010");
        flex.setCellData(2, 0, "1031");
        flex.setCellData(3, 0, "1110");
        flex.setCellData(1, 1, "60");
        flex.setCellData(2, 1, "50");
        flex.setCellData(3, 1, "55");
        flex.setCellData(0, 2, "Product Cost");
        flex.setCellData(1, 2, "38");
        flex.setCellData(2, 2, "50");
        flex.setCellData(3, 2, "55");
        flex.setCellData(0, 3, "Total");
    }
    function setFormula() {
        var flex = wijmo.Control.getControl("#formulaSheet");
        flex.setCellData(1, 3, "=product(B2:C2)");
        flex.setCellData(2, 3, "=product(B3:C3)");
        flex.setCellData(3, 3, "=product(B4:C4)");
    }
</script>

<div>
    <input id="bold" type="button" onclick="generateCostSheet()" value="Display Cost" />
    <input id="textAlign" type="button" onclick="setFormula()" value="Apply Formula" />
    <c1-flex-sheet class="flexSheet" id="formulaSheet" Height="500px" Width="800px" is-read-
only="false">
    <c1-unbound-sheet name="Cost" row-count="6" column-count="4"></c1-unbound-sheet>
    <c1-formula-bar/>
    </c1-flex-sheet>
</div>
```

Back to Top

In-built Formulas Supported in FlexSheet

The FlexSheet control supports following in-built formulas.

Name	Description
abs	Returns the absolute value of a number.
acos	Returns the arccosine of a number.
and	Returns TRUE if all of its arguments are TRUE.
asin	Returns the arcsine of a number.
atan	Returns the arctangent of a number.
atan2	Returns the arctangent from x- and y-coordinates.
average	Returns the average of its arguments.
ceiling	Rounds a number to the nearest integer or to the nearest multiple of significance.
char	Returns the character specified by the code number.

Name	Description
choose	Chooses a value from a list of values.
code	Returns a numeric code for the first character in a text string.
column	Returns the column number of a reference.
columns	Returns the number of columns in a reference.
concatenate	Joins several text items into one text item.
cos	Returns the cosine of a number.
count	Counts how many numbers are in the list of arguments.
counta	Counts how many values are in the list of arguments.
countblank	Counts the number of blank cells within a range.
countif	Counts the number of cells within a range that meet the given criteria.
countifs	Counts the number of cells within a range that meet multiple criteria.
date	Returns the serial number of a particular date.
datediff	Calculates the number of days, months, or years between two dates.
day	Converts a serial number to a day of the month.
dcount	Counts the cells that contain numbers in a database.
exp	Returns e raised to the power of a given number.
false	Returns the logical value FALSE.
find	Finds one text value within another (case-sensitive).
floor	Rounds a number down, toward zero.
hlookup	Looks in the top row of an array and returns the value of the indicated cell.
hour	Converts a serial number to an hour.
if	Specifies a logical test to perform.
index	Uses an index to choose a value from a reference.
left	Returns the leftmost characters from a text value.
len	Returns the number of characters in a text string.
ln	Returns the natural logarithm of a number.
lower	Converts text to lowercase.
max	Returns the maximum value in a list of arguments.
mid	Returns a specific number of characters from a text string starting at the position you specify.
min	Returns the minimum value in a list of arguments.
mod	Returns the remainder from division.
month	Converts a serial number to a month.
not	Reverses the logic of its argument.

Name	Description
now	Returns the serial number of the current date and time.
or	Returns TRUE if any argument is TRUE.
pi	Returns the value of pi.
power	Returns the result of a number raised to a power.
product	Multiplies its arguments.
proper	Capitalizes the first letter in each word of a text value.
rand	Returns a random number between 0 and 1.
rank	Returns the rank of a number in a list of numbers.
rate	Returns the interest rate per period of an annuity.
replace	Replaces characters within text.
rept	Repeats text a given number of times.
right	Returns the rightmost characters from a text value.
round	Rounds a number to a specified number of digits.
rounddown	Rounds a number down, toward zero.
roundup	Rounds a number up, away from zero.
row	Returns the row number of a reference.
rows	Returns the number of rows in a reference.
search	Finds one text value within another (not case-sensitive).
sin	Returns the sine of the given angle.
sqrt	Returns a positive square root.
stdev	Estimates standard deviation based on a sample.
stdevp	Calculates standard deviation based on the entire population.
substitute	Substitutes new text for old text in a text string.
subtotal	Returns a subtotal in a list or database.
sum	Adds its arguments.
sumif	Adds the cells specified by a given criteria.
sumifs	Adds the cells in a range that meet multiple criteria.
tan	Returns the tangent of a number.
text	Formats a number and converts it to text.
time	Returns the serial number of a particular time.
today	Returns the serial number of today's date.
trim	Removes spaces from text.
true	Returns the logical value TRUE.

Name	Description
trunc	Truncates a number to an integer.
upper	Converts text to uppercase.
value	Converts a text argument to a number.
var	Estimates variance based on a sample.
varp	Calculates variance based on the entire population.
year	Converts a serial number to a year.

Custom Function

FlexSheet supports conditional formatting on the client side, that allows you to change the formatting of a cell or a range of cells depending on the value of the cell(s) or formula applied on the cell(s). Applying conditional formatting to FlexSheet control empowers you to boost its capabilities, thereby enabling you to do more than what you could do using its built-in functionality.

For example, while maintaining expenses and sales record for your retail store, you can easily track when your actual expenses are shooting over the budgeted expenses. The below example demonstrates this scenario.

The following image shows FlexSheet control displaying expenses details for a retail store, with conditional formatting applied on cells having values greater than 1000.
<image>

The following code examples demonstrate how to add custom functions in FlexSheet.

In Code

CustomFunctionController.cs

```
C#                                                                    copyCode
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

CustomFunction.cshtml

HTML Helpers

```
Razor                                                                 copyCode
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Tag Helpers

```
HTML                                                                  copyCode
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Back to Top

Remote Loading and Saving of Excel

FlexSheet allows you to remotely load and save excel file using C1JSONRequest. This specifies remote URLs to load data from excel file or workbook in FlexSheet control, and save FlexSheet data to server as Excel file or workbook through action.

The below example loads an excel file, placed on server, in FlexSheet using **Load** method of FlexSheetHelper class.

Note that FlexSheet requires **jszip.min.js** file for remote loading and saving of excel. Therefore, you need to add this file to your application and provide reference to it in the respective view or in <head> section of _Layout.cshtml file.

The following code examples demonstrate how to remotely load data from excel file or workbook in FlexSheet control:

In Code

Include the MVC references as shown below.

C# copyCode

```
using C1.Web.Mvc.Sheet;
using System;
using System.Collections;
using System.Globalization;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Collections.Generic;
using C1.Web.Mvc.Serializition;
```

RemoteLoadController.cs

ASP.NET

C# copyCode

```
public class RemoteController : Controller
{
    // GET: /<controller>/
    public ActionResult Index()
    {
        return View();
    }
    public ActionResult RemoteLoadXlsx()
    {
        return this.C1Json(FlexSheetHelper.Load("~/Content/ExcelFiles/WorkBook.xlsx"), null, null,
JsonRequestBehavior.AllowGet);
    }
}
```

Here, the Excel workbook that is loaded remotely is placed in Content folder of the application.

ASP.NET Core

C# copyCode

```
public class RemoteController : Controller
{
    // GET: /<controller>/
    public IActionResult Index()
    {
        return View();
    }
    public ActionResult RemoteLoadXlsx()
    {
        return this.C1Json(FlexSheetHelper.Load("~/ToLoad/WorkBook.xlsx"));
    }
}
```

Here, the Excel workbook that is loaded remotely is placed in wwwroot folder of the application.

RemoteLoading.cshtml

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc.Sheet;
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>

<div>
@(Html.C1().FlexSheet().CssClass("flexSheet").Height(700).Width(700)
        .RemoteLoad(Url.Action("RemoteLoadXlsx"))
        )
</div>
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc.Sheet;
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>

<div>
    <c1-flex-sheet class="flexSheet" id="flexSheet" height="700px" width="700px"
                   load-action-url="@(Url.Action("RemoteLoadXlsx"))">
    </c1-flex-sheet>
</div>
```

Back to Top

Remote Save

In Code

RemoteSaveController.cs

C# copyCode

```
public class RemoteSaveController : Controller
{
    // GET: /<controller>/
    public static List<Sale> SALES = Sale.GetData(50).ToList();
    public ActionResult Index()
    {
        return View(SALES);
    }

    private const string FILE_PATH = "\\wwwroot\\uploadFile\\save.xlsx";
    public JsonResult RemoteSaveFile([FlexSheetRequest]FlexSheetSaveRequest request)
    {
        var success = true;
        var error = "";
        var app = GetService<IApplicationEnvironment>();
        var savePath = app.ApplicationBasePath + FILE_PATH;
        try
        {
            Stream st = request.GetFileStream();
            using (FileStream fs = new FileStream(savePath, FileMode.Create))
            {
                if (st != null)
                {
                    st.CopyTo(fs);
                }
            }
        }
        catch (Exception e)
        {
            success = false;
            error = e.ToString();
        }

        return this.C1Json(FlexSheetHelper.Save(success, error));
```

```
    }

    public FileResult DownloadFile()
    {
        var app = GetService<IApplicationEnvironment>();
        var savePath = app.ApplicationBasePath + FILE_PATH;
        var name = Path.GetFileName(FILE_PATH);
        return File(new FileStream(savePath, FileMode.Open, FileAccess.Read),
            "application/msexcel", name);
    }

    public static T GetService<T>() where T : class
    {
        var serviceProvider = CallContextServiceLocator.Locator.ServiceProvider;
        return serviceProvider.GetService(typeof(T)) as T;
    }
}
```

RemoteSaving.cshtml

HTML Helpers

Razor copyCode

```
@using C1MvcFSheetNew.Models;
@model IEnumerable<sale>

<script>
        function remoteSave() {
            var flexSheet = wijmo.Control.getControl('#flexSheet');
            flexSheet.remoteSave(c1.mvc.grid.sheet.ContentType.Xlsx);
        }

        function onFileSaved(sender, args) {
            if (args.success) {
                window.location.href = '@Url.Action("DownloadFile")';
            } else {
                alert(args.error);
            }
        }
</script>

<div>

@(Html.C1().FlexSheet().CssClass("flexSheet").Width("500px").Height("700px").RemoteSave(Url.Action("RemoteSaveFile"))
        .OnClientRemoteSaved("onFileSaved")
        .AddBoundSheet(sheet =>
            sheet.Bind(cv =>
                            cv.Bind(Model)))
    )
</div>
```

Tag Helpers

HTML copyCode

```
<script>
        function remoteSave() {
            var flexSheet = wijmo.Control.getControl('#flexSheet');
            flexSheet.remoteSave(c1.mvc.grid.sheet.ContentType.Xlsx);
        }

        function onFileSaved(sender, args) {
            if (args.success) {
                window.location.href = '@Url.Action("DownloadFile")';
            } else {
                alert(args.error);
            }
        }
```

```
</script>

<div>
    <button type="button" class="btn btn-default" onclick="remoteSave()">Save</button>
    <br /><br />

        <c1-flex-sheet class="flexSheet" id="flexSheet" width="500px" height="700px" save-action-
url="@(Url.Action("RemoteSaveFile"))"
                       remote-saved="onFileSaved">
            <c1-bound-sheet>
                <c1-items-source source-collection="Model"></c1-items-source>
            </c1-bound-sheet>
        </c1-flex-sheet>
    </div>
```

Back to Top

Client-side Loading and Saving of Excel

FlexSheet supports loading data in it from an Excel file or workbook provided by client. Moreover, it allows saving the data in FlexSheet to Excel file or workbook on client-side.

The following code examples demonstrate how to enable client loading of excel in FlexSheet. In the below example, client loading of excel occurs on button click. Load method is called when a user clicks on the button to load excel data in FlexSheet.

> **Note** that FlexSheet requires **jszip.min.js** file for loading and saving of excel on the client side. Therefore, you need to add this file to your application and provide reference to it in the respective view or in <head> section of **_Layout.cshtml** file.

In Code

ClientLoadController.cs

C# copyCode

```
public class ClientLoadController : Controller
{
    // GET: /<controller>/
    public ActionResult Index()
    {
        return View();
    }
}
```

ClientLoad.cshtml

HTML Helpers

Razor copyCode

```
@using C1MvcFSheetNew.Models;
@model IEnumerable<Sale>
```

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script>
        function load() {
            var flex = wijmo.Control.getControl("#clientLoadSheet");
            var fileInput = wijmo.getElement('#importFile');
            if (flex && fileInput.files[0]) {
                flex.load(fileInput.files[0]);
            }
        }
    </script>

    <div>
        <input type="file" class="form-control" id="importFile"
accept="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" />
        <button class="btn btn-default" onclick="load()">Load</button>
        <br /><br />
        @(Html.C1().FlexSheet().CssClass("flexSheet").Id("clientLoadSheet")
        .SelectedSheetIndex(0).Width("500px").Height("700px").AddBoundSheet(sheet =>
                sheet.Bind(cv =>
                    cv.Bind(Model).DisableServerRead(true)))
        .AddUnboundSheet("Unbound", 20, 8))

    </div>
```

Tag Helpers

HTML copyCode

```
<script>
    function load() {
        var flex = wijmo.Control.getControl("#clientLoadSheet");
        var fileInput = wijmo.getElement('#importFile');
        if (flex && fileInput.files[0]) {
            flex.load(fileInput.files[0]);
        }
    }
</script>

<div>
    <input type="file" class="form-control" id="importFile"
accept="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" />
    <button class="btn btn-default" onclick="load()">Load</button>
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="clientLoadSheet" selected-sheet-index="0"
width="500px" height="700px">
        <c1-unbound-sheet column-count="8" row-count="20" name="Unbound"></c1-
unbound-sheet>
    </c1-flex-sheet>
</div>
```

Back to Top

Saving FlexSheet Data to Excel File on Client-side

Saving the FlexSheet data to an Excel file or workbook is also supported on the client side. This can be accomplished through **save** method. The following code examples demonstrate how to enable saving FlexSheet data to Excel file on the client side. In the below example save method is invoked on button click.

In Code

ClientSaveController.cs

C# copyCode

```
public class ClientSaveController : Controller
{
    // GET: /<controller>/
    public static List<Sale> SALES = Sale.GetData(50).ToList();
    public ActionResult Index()
    {
        return View(SALES);
    }
```

ClientSaving.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<Sale>

<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script>
        function save() {
            var flex = wijmo.Control.getControl("#clientSaveSheet");
            var fileNameInput = wijmo.getElement("#fileName");
            var fileName = 'FlexSheet.xlsx';
            flex.save(fileName);
        }
    </script>

    <div>
        <button class="btn btn-default" onclick="save()">Save</button>
        <br /><br />
        @(Html.C1().FlexSheet().CssClass("flexSheet").Id("clientSaveSheet")
        .SelectedSheetIndex(0).Width("500px").Height("700px").AddBoundSheet(sheet =>
                sheet.Bind(cv =>
                    cv.Bind(Model).DisableServerRead(true)))
```

```
        .AddUnboundSheet("Unbound", 20, 8))
    </div>
```

Tag Helpers

HTML copyCode

```
@using C1MvcFSheetNew.Models;
@model IEnumerable<sale>

    <script>
        function save() {
            var flex = wijmo.Control.getControl("#clientSaveSheet");
            var fileNameInput = wijmo.getElement("#fileName");
            var fileName = 'FlexSheet.xlsx';
            flex.save(fileName);
        }


    </script>

    <div>
        <button class="btn btn-default" onclick="save()">Save</button>
        <br /><br />
        <c1-flex-sheet class="flexSheet" id="clientSaveSheet" selected-sheet-
index="0" width="500px" height="700px">
            <c1-bound-sheet>
                <c1-items-source source-collection="Model"></c1-items-source>
            </c1-bound-sheet>
            <c1-unbound-sheet column-count="8" row-count="20" name="Unbound"></c1-
unbound-sheet>
        </c1-flex-sheet>
    </div>
```

Back to Top

JSON Loading and Saving on Client-side

FlexSheet supports Json loading and saving on the client side. You can easily load data from a json string in FlexSheet control, and even save the FlexSheet data in Json string. The control uses Workbook object model to accomplish this.

The below code examples demonstrate how to use workbook object model to load Json string into FlexSheet control.

In Code

JsonLoadController.cs

C# copyCode

```
public class JsonLoadController : Controller
{
    public static List<Sale> SALES = Sale.GetData(15).ToList();
```

```
    // GET: Json
    public ActionResult JsonIndex()
    {
        return View(SALES);
    }
}
```

JsonLoading.cshtml

HTML Helpers

Razor copyCode

```
<script>
    var flex, jsonString;
    c1.mvc.Utils.documentReady(function () {
        flex = wijmo.Control.getControl('#jsonLoadSheet');
    });

    function SaveToJson() {
        var workBook = flex.saveToWorkbookOM();
        jsonString = JSON.stringify(workBook);

    }

    function loadJSON() {
        var workBook = JSON.parse(jsonString);
        flex.loadFromWorkbookOM(workBook);
    }
</script>

<div>
    <button class="btn btn-default" onclick="SaveToJson()">Save to Json</button>
    <button class="btn btn-default" onclick="loadJSON()">Load Json</button>
    <br /><br />

@(Html.C1().FlexSheet().Id("jsonLoadSheet").CssClass("flexSheet").Width(700).Height(700)
        .AddBoundSheet(sheet =>
            sheet.Bind(cv =>
                            cv.Bind(Model).DisableServerRead(true)))
    )
</div>
```

Tag Helpers

HTML copyCode

```
<script>
    var flex, jsonString;
    c1.mvc.Utils.documentReady(function () {
        flex = wijmo.Control.getControl("#jsonLoadSheet");
```

```
    });

    function SaveToJson() {
        var workBook = flex.saveToWorkbookOM();
        jsonString = JSON.stringify(workBook);

    }

    function loadJSON() {
        var workBook = JSON.parse(jsonString);
        flex.loadFromWorkbookOM(workBook);
    }
</script>

<div>
    <button class="btn btn-default" onclick="SaveToJson()">Save to Json</button>
    <button class="btn btn-default" onclick="loadJSON()">Load Json</button>
    <br /><br />
    <div>
        <c1-flex-sheet class="flexSheet" id="jsonLoadSheet" height="700px"
width="700px">
            <c1-bound-sheet>
                <c1-items-source source-collection="Model"></c1-items-source>
            </c1-bound-sheet>
        </c1-flex-sheet>
    </div>
</div>
```

Back to Top

Cell Merging

FlexSheet supports excel-like merging of cells on the client side, unlike FlexGrid which enables content-driven cell merging, FlexSheet allows you to combine adjacent cells containing any data/value into one. When you merge a group of cells, the contents/value/data from the upper-leftmost cell is preserved in the merged cell.

You can merge the selected cells of a sheet into one by invoking the **mergeRange()** method. However if your selection contains already merged cell(s), then the mergeRange method will un-merge the merged cell(s).

The following sections demonstrate how to merge cells at run-time and through code.

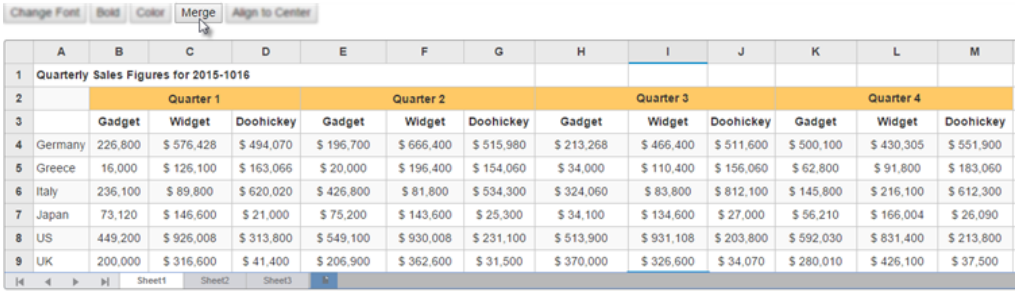
> **Note:** Note that, FlexSheet also supports FlexGrid like merging on the server side.

Merging Cells at Run-time

Merging the cells together in a worksheet allows you to present data in a comprehensible manner. When you group the contents of similar cells together by merging them, you can enhance the readability of otherwise usual data.

There could be many instances where you need to merge some of your worksheet data, for better representation. For example, you might need to display and analyze region wise Quarterly sales per product for an organization. In such a situation, merging the cells with heading for different quarters can enable better presentation. The following example discusses this scenario, by performing cell merge at the run-time on button click. Here, **mergeRange()** is invoked when you select the cells to merge and then click the Merge button.

Note that the FlexSheet, in the below example, shows data from a workbook file which is loaded from server.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Quarterly Sales Figures for 2015-1016												
2		Quarter 1			Quarter 2			Quarter 3			Quarter 4		
3		Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey	Gadget	Widget	Doohickey
4	Germany	226,800	$ 576,428	$ 494,070	$ 196,700	$ 666,400	$ 515,980	$ 213,268	$ 466,400	$ 511,600	$ 500,100	$ 430,305	$ 551,900
5	Greece	16,000	$ 126,100	$ 163,066	$ 20,000	$ 196,400	$ 154,060	$ 34,000	$ 110,400	$ 156,060	$ 62,800	$ 91,800	$ 183,060
6	Italy	236,100	$ 89,800	$ 620,020	$ 426,800	$ 81,800	$ 534,300	$ 324,060	$ 83,800	$ 812,100	$ 145,800	$ 216,100	$ 612,300
7	Japan	73,120	$ 146,600	$ 21,000	$ 75,200	$ 143,600	$ 25,300	$ 34,100	$ 134,600	$ 27,000	$ 56,210	$ 166,004	$ 26,090
8	US	449,200	$ 926,008	$ 313,800	$ 549,100	$ 930,008	$ 231,100	$ 513,900	$ 931,108	$ 203,800	$ 592,030	$ 831,400	$ 213,800
9	UK	200,000	$ 316,600	$ 41,400	$ 206,900	$ 362,600	$ 31,500	$ 370,000	$ 326,600	$ 34,070	$ 280,010	$ 426,100	$ 37,500

The following code examples demonstrate how to enable Cell Merging in the FlexSheet control at Run-time:

In Code

MergingController.cs

C# copyCode

```
public class MergeController : Controller
{
    // GET: Merge
    public ActionResult MergeIndex()
    {
        return View();
    }
}
```

Merging.cshtml

HTML Helpers

Razor copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script>
    function mergeSelection() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.mergeRange();
    }
    function setBold() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ fontWeight: 'bold' });
    }
    function changeFont() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ fontFamily: 'Unicode' });
    }
    function setColor() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ backgroundColor: '#FFC966' });
    }
    function align() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ textAlign: 'Center' });
    }
</script>

<div>
    <input id="font" type="button" onclick="changeFont()" value="Change Font" />
    <input id="bold" type="button" onclick="setBold()" value="Bold" />
    <input id="cellColor" type="button" onclick="setColor()" value="Color" />
    <input id="merge" type="button" onclick="mergeSelection()" value="Merge" />
    <input id="textAlign" type="button" onclick="align()" value="Align to Center" />
    <br /><br />

@(Html.C1().FlexSheet().Height("500px").Width("1500px").Load("~/Content/FlexSheet/QuarterlyData.xlsx").Id("mSheet")
    )
</div>
```

Tag Helpers

HTML copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script>
    function mergeSelection() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.mergeRange();
    }
    function setBold() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ fontWeight: 'bold' });
    }
    function changeFont() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ fontFamily: 'Unicode' });
    }
    function setColor() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ backgroundColor: '#FFC966' });
    }
    function align() {
        var flex = wijmo.Control.getControl("#mSheet");
        flex.applyCellsStyle({ textAlign: 'Center' });
    }

</script>

<div>
    <input id="font" type="button" onclick="changeFont()" value="Change Font" />
    <input id="bold" type="button" onclick="setBold()" value="Bold" />
    <input id="cellColor" type="button" onclick="setColor()" value="Color" />
    <input id="merge" type="button" onclick="mergeSelection()" value="Merge" />
    <input id="textAlign" type="button" onclick="align()" value="Align to Center" />
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="mSheet" height="500px" width="1500px" file-
path="~/Content/FlexSheet/QuarterlyData.xlsx">
    </c1-flex-sheet>
</div>
```

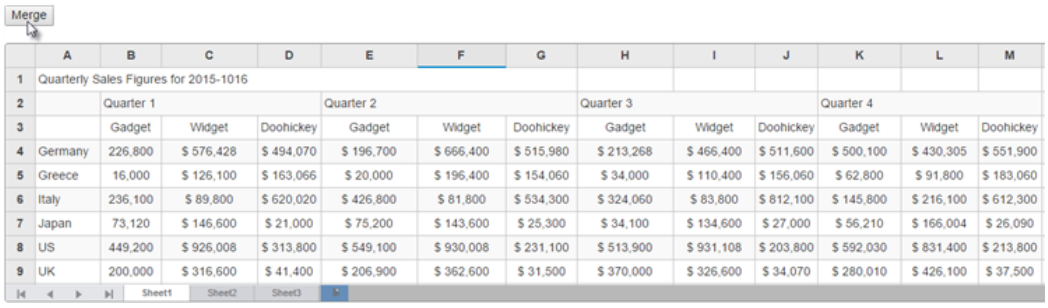
Back to Top

Merging Cells Programmatically

FlexSheet allows you to carry out merging of cells in FlexSheet control through code. This can be achieved by specifying the CellRange to merge in **mergeRange()** method. This section demonstrates this by extending the scenario in Merging Cells at Run-time.

Here, we use CellRange to specify the group of adjacent cells which are to be merged, by stating the index of first row, first column, last row, and last column in the range.

Note that the FlexSheet, in the below example, shows data from a workbook file which is loaded from server.



The following code examples demonstrate how to enable Cell Merging in the FlexSheet control through code:

In Code

MergeController.cs

C# copyCode

```
public class CellMergeController : Controller
{
    // GET: CellMerge
    public ActionResult CodeMrgeIndex()
    {
        return View();
    }
}
```

Merging.cshtml

HTML Helpers

Razor copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script>
function mergeSel() {
        //merge predefined range through code
        var flex = wijmo.Control.getControl("#mSheet");
        flex.mergeRange(new wijmo.grid.CellRange(0, 0, 0, 6));
        flex.mergeRange(new wijmo.grid.CellRange(1, 1, 1, 3));
        flex.mergeRange(new wijmo.grid.CellRange(1, 4, 1, 6));
        flex.mergeRange(new wijmo.grid.CellRange(1, 7, 1, 9));
        flex.mergeRange(new wijmo.grid.CellRange(1, 10, 1, 12));
        flex.refresh();
    }
</script>

<div>
    <input id="merge" type="button" onclick="mergeSel()" value="Merge" />
    <br /><br />

@(Html.C1().FlexSheet().Height("500").Width("1500").Load("~/Content/FlexSheet/QuarterlyData.xlsx").Id("mSheet")
    )
</div>
```

Tag Helpers

HTML copyCode

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
<script>
    function mergeSel() {
        //merge predefined range through code
        var flex = wijmo.Control.getControl("#mSheet");
        flex.mergeRange(new wijmo.grid.CellRange(0, 0, 0, 6));
        flex.mergeRange(new wijmo.grid.CellRange(1, 1, 1, 3));
        flex.mergeRange(new wijmo.grid.CellRange(1, 4, 1, 6));
        flex.mergeRange(new wijmo.grid.CellRange(1, 7, 1, 9));
        flex.mergeRange(new wijmo.grid.CellRange(1, 10, 1, 12));
        flex.refresh();
    }
</script>

<div>
    <input id="merge" type="button" onclick="mergeSel()" value="Merge" />
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="mSheet" height="500px" width="1500px" file-
path="~/Content/FlexSheet/QuarterlyData.xlsx">
    </c1-flex-sheet>
```

```
</div>
```

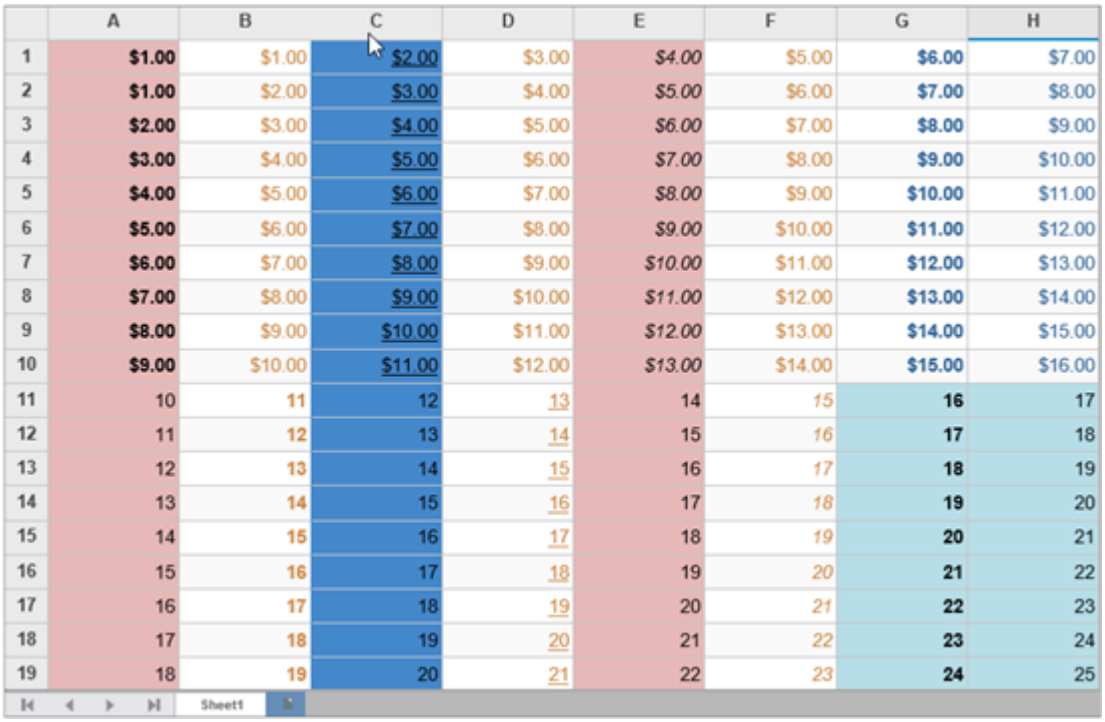
Back to Top

Drag and Drop

FlexSheet provides in-built support for drag-and-drop editing of its rows and columns. You can easily select a row or column with mouse, to drag and move its contents to a different row or column. This not only copies and moves the contents of the specific rows and columns but also their cell style.

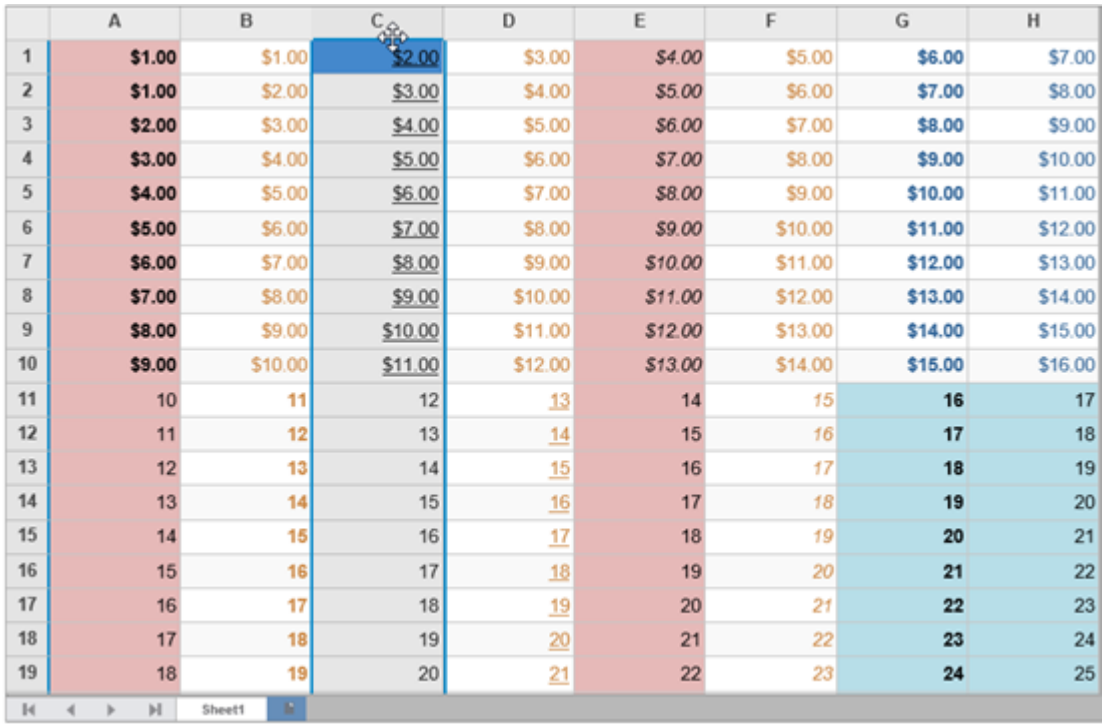
You simply need to accomplish the following steps to handle drag and drop operation:

1. Position your mouse pointer on a row/column range border so that it changes to a move pointer .



	A	B	C	D	E	F	G	H
1	$1.00	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00
2	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00
3	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00
4	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00
5	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00
6	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00
7	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00
8	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00
9	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00
10	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00	$16.00
11	10	11	12	13	14	15	16	17
12	11	12	13	14	15	16	17	18
13	12	13	14	15	16	17	18	19
14	13	14	15	16	17	18	19	20
15	14	15	16	17	18	19	20	21
16	15	16	17	18	19	20	21	22
17	16	17	18	19	20	21	22	23
18	17	18	19	20	21	22	23	24
19	18	19	20	21	22	23	24	25

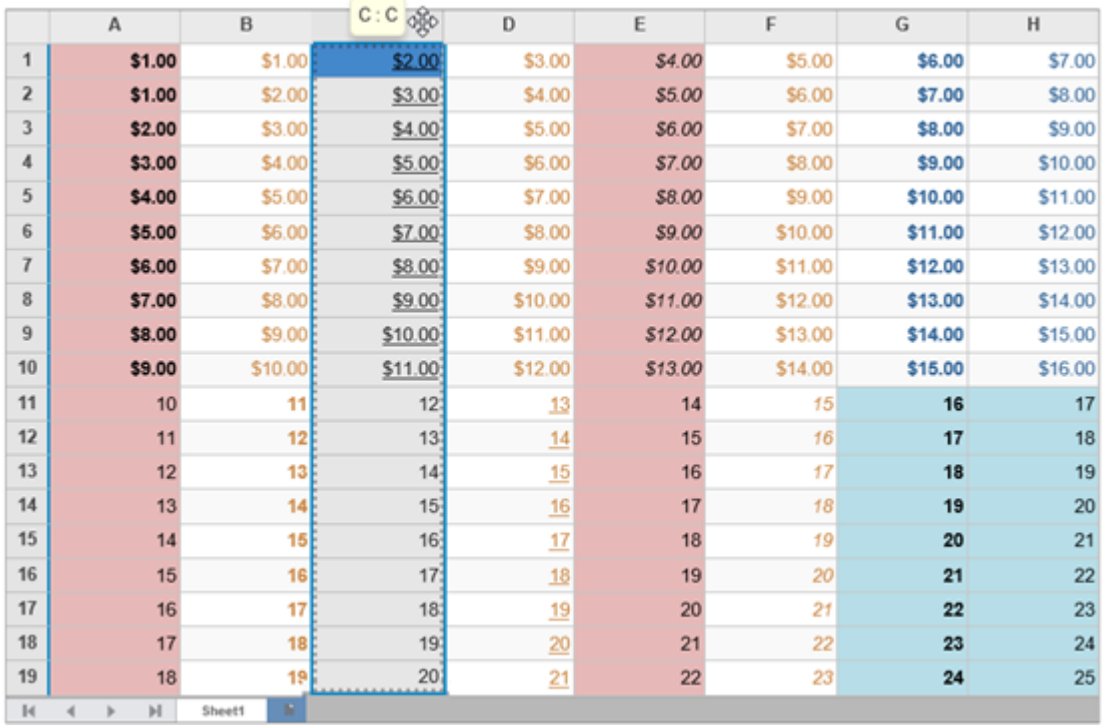
Sheet1



	A	B	C	D	E	F	G	H
1	$1.00	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00
2	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00
3	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00
4	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00
5	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00
6	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00
7	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00
8	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00
9	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00
10	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00	$16.00
11	10	11	12	13	14	15	16	17
12	11	12	13	14	15	16	17	18
13	12	13	14	15	16	17	18	19
14	13	14	15	16	17	18	19	20
15	14	15	16	17	18	19	20	21
16	15	16	17	18	19	20	21	22
17	16	17	18	19	20	21	22	23
18	17	18	19	20	21	22	23	24
19	18	19	20	21	22	23	24	25

Sheet1

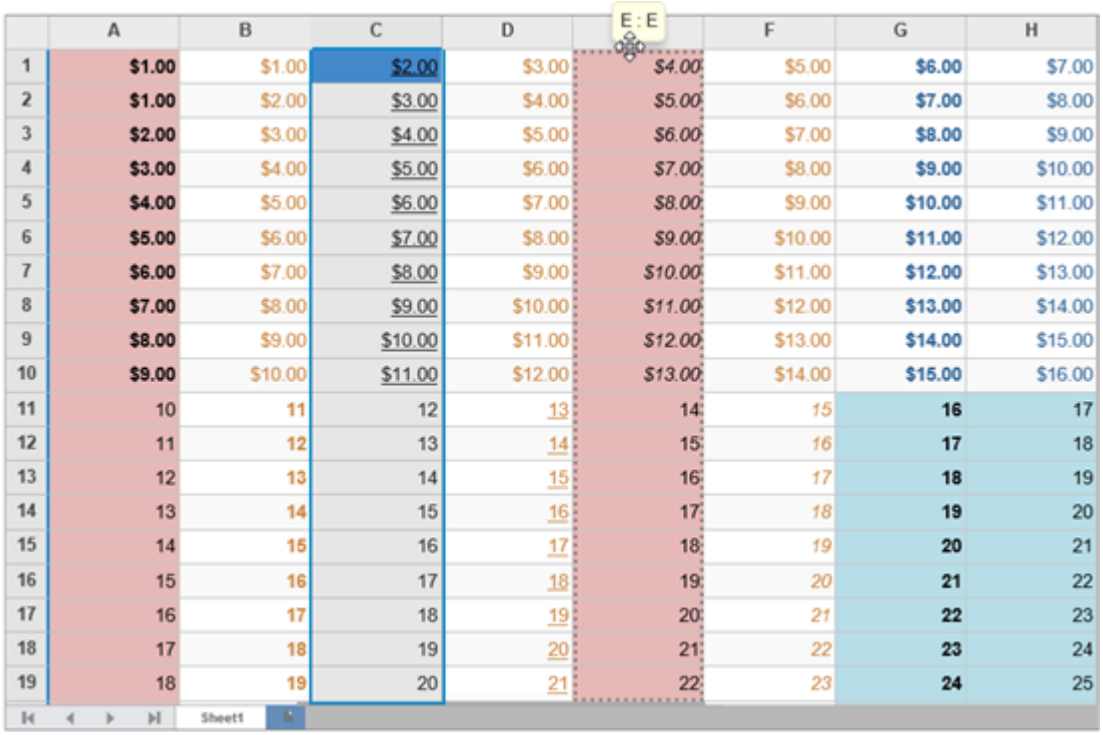
2. Select the row/column by holding down the mouse, along with 'Ctrl' or 'Shift' key pressed or without any key press.



	A	B	C : C	D	E	F	G	H
1	$1.00	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00
2	$1.00	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00
3	$2.00	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00
4	$3.00	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00
5	$4.00	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00
6	$5.00	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00
7	$6.00	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00
8	$7.00	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00
9	$8.00	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00
10	$9.00	$10.00	$11.00	$12.00	$13.00	$14.00	$15.00	$16.00
11	10	11	12	13	14	15	16	17
12	11	12	13	14	15	16	17	18
13	12	13	14	15	16	17	18	19
14	13	14	15	16	17	18	19	20
15	14	15	16	17	18	19	20	21
16	15	16	17	18	19	20	21	22
17	16	17	18	19	20	21	22	23
18	17	18	19	20	21	22	23	24
19	18	19	20	21	22	23	24	25

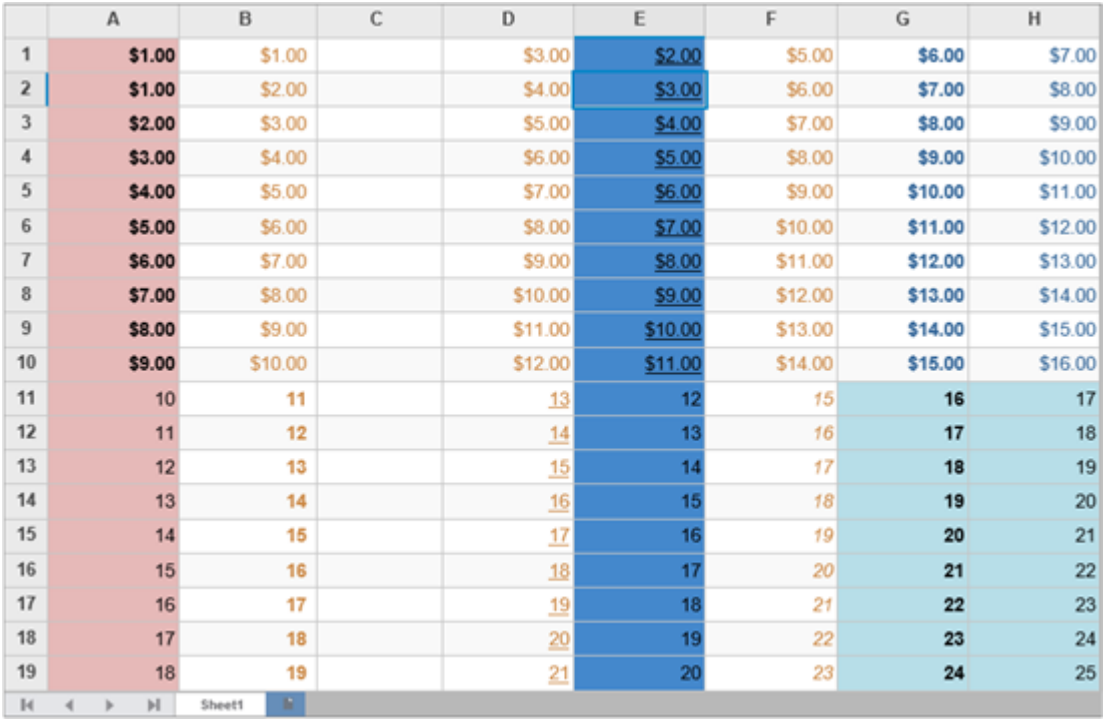
Sheet1

3. Drag the selection to a desired row/column location.

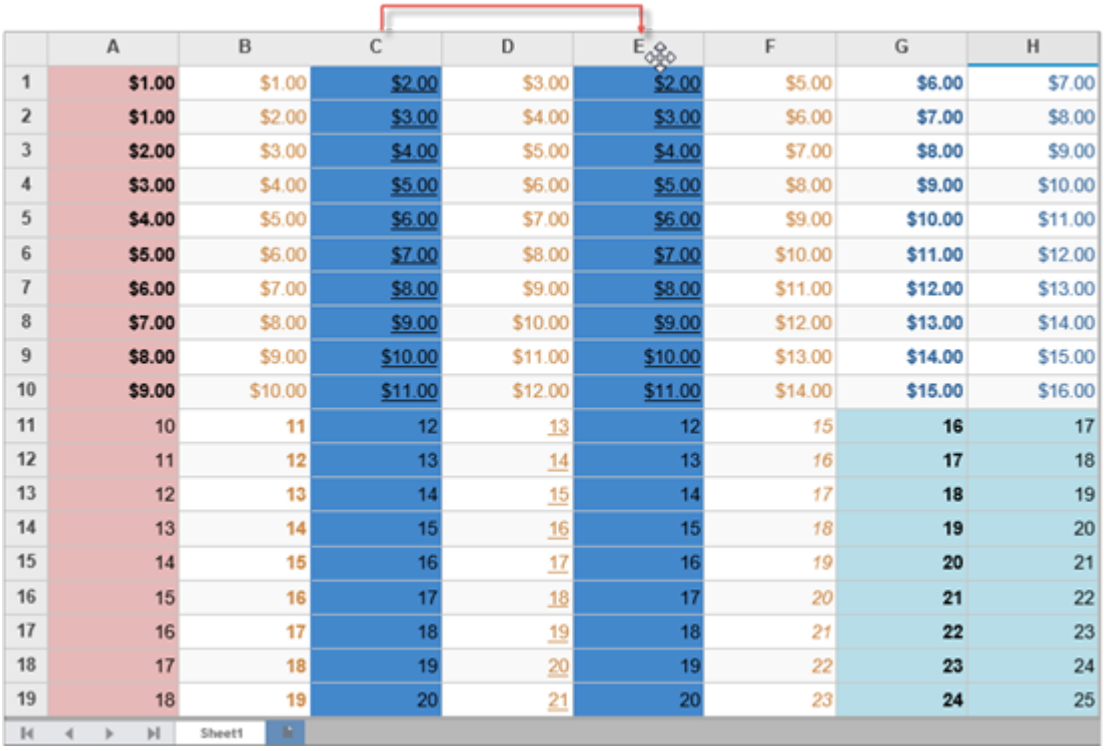


When you;

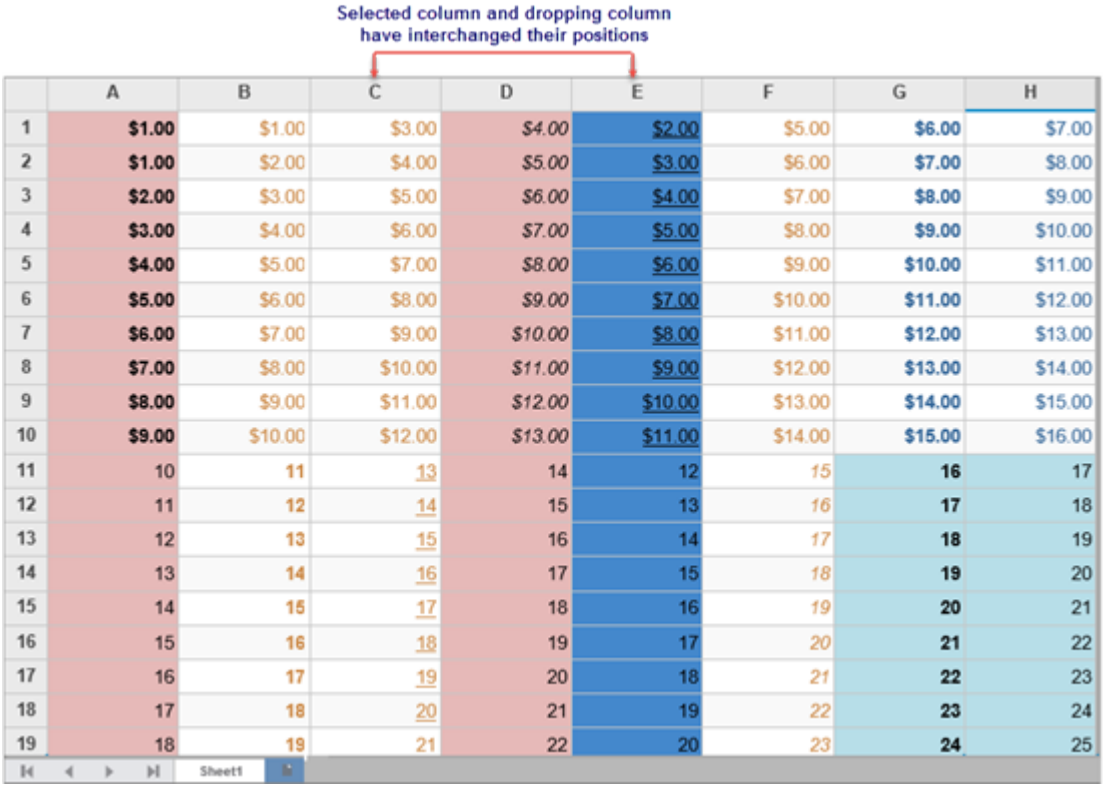
- Drag and drop without any key pressed, it moves the contents of selected columns or rows into the dropping columns or rows. Thereby, emptying the selected column/row.



- Drag and drop with '**Ctrl**' key pressed, it copies the selected columns or rows into the dropping columns or rows. This replaces the contents in the dropping columns/rows, while still retaining the contents of selected column/row.



- Drag and drop with '**Shift**' key pressed, it changes the position of the selected columns or rows with the dropping columns or rows. Therefore the contents of selected and dropping columns/rows get interchanged.



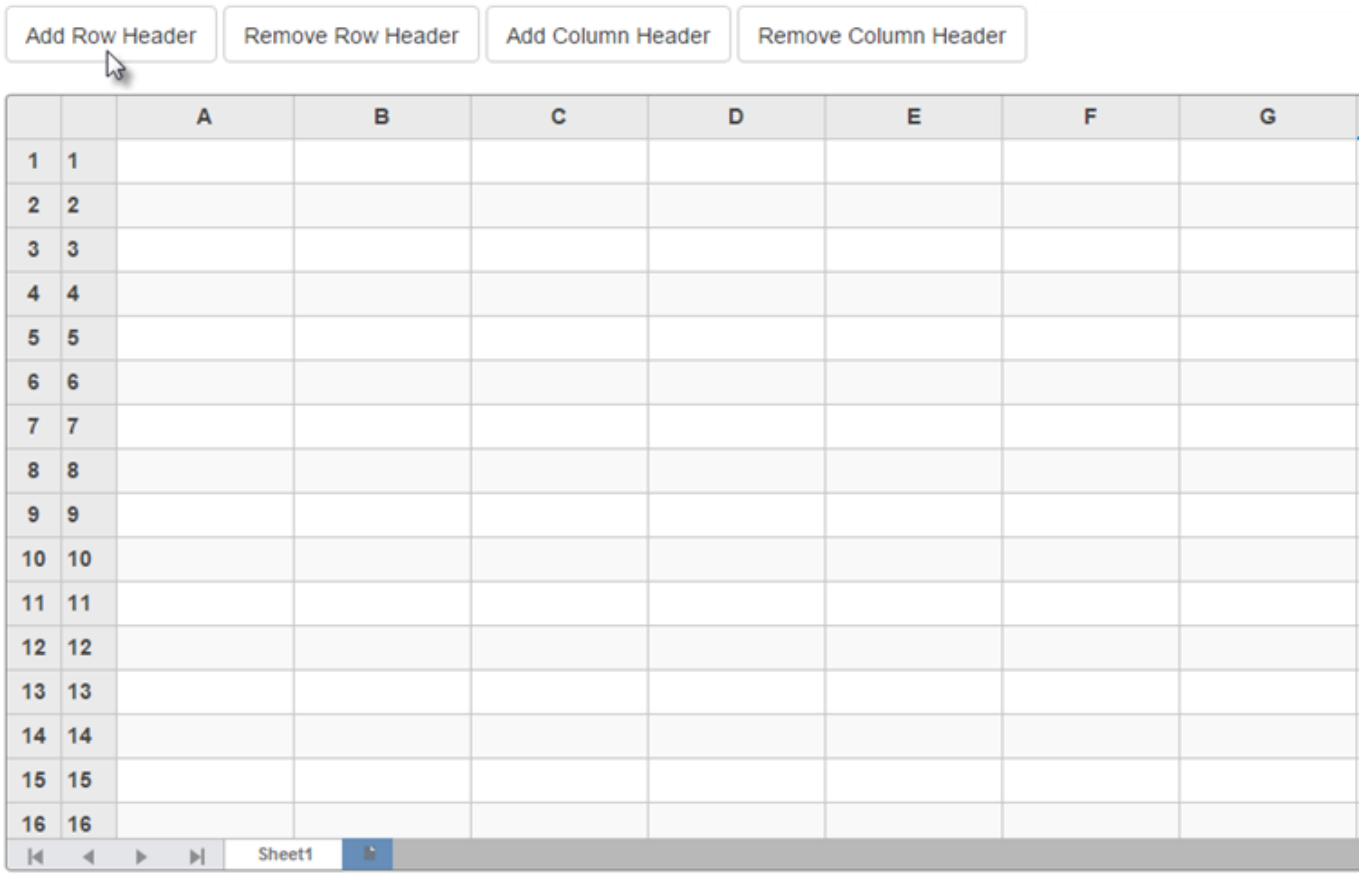
Back to Top

Multiple Headers

FlexSheet control can be customized to enable adding and removing row/column headers on the client side. You can, therefore, add multiple row/column headers to your worksheet and even remove the existing ones.

Your FlexSheet control by default contain a row and a column header, when you create and run a basic MVC application containing FlexSheet. The column header displays uppercase letters ("A", "B", "C", etc.) to represent different columns, while row header displays numbers ("1", "2", "3", etc.) to represent different rows. However, you can alter this default behavior by either adding more headers or removing the existing ones. This can be done by invoking **push()** function on **rowHeaders.columns** and **columnHeaders.rows** collection, for adding row headers and column headers respectively. However, to remove row and column headers, **removeAt()** function can be invoked on **rowHeaders.columns** and **columnHeaders.rows** collection respectively.

The following image shows an unbound FlexSheet control along with buttons to add and remove row/column headers. In the below example, removeAt() and push() are invoked on button click.



The following code examples demonstrate how to add or remove row and column headers in the FlexSheet:

In Code

MultipleHeaderController.cs

C# copyCode

```
public class MultipleHeaderController : Controller
{
    // GET: MultipleHeader
    public ActionResult MultipleHeaderIndex()
    {
        return View();
    }
}
```

MultipleHeader.cshtml

HTML Helpers

Razor copyCode

```
<script>
    function addRowHeader() {
        var flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.rowHeaders.columns.push(new wijmo.grid.Column());
    }
    function removeRowHeader() {
        var colCnt, flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.rowHeaders.columns.removeAt(colCnt - 1);
    }
    function addColumnHeader() {
        var rowCnt, flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.columnHeaders.rows.push(new wijmo.grid.Row());
    }
    function removeColumnHeader() {
        var flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.columnHeaders.rows.removeAt(rowCnt - 1);
    }
    </script>
<div>
    <button type="button" class="btn btn-default" onclick="addRowHeader()">Add Row
Header</button>
    <button type="button" class="btn btn-default" onclick="removeRowHeader()">Remove
Row Header</button>
    <button type="button" class="btn btn-default" onclick="addColumnHeader()">Add
Column Header</button>
    <button type="button" class="btn btn-default"
onclick="removeColumnHeader()">Remove Column Header</button>
    <br /><br />

@(Html.C1().FlexSheet().CssClass("flexSheet").Id("mHeadersSheet").IsReadOnly(false)
        .AddUnboundSheet("Sheet1", 20, 8))
</div>
```

Tag Helpers

HTML copyCode

```
<script>
    function addRowHeader() {
        var flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.rowHeaders.columns.push(new wijmo.grid.Column());
    }
    function removeRowHeader() {
```

```
        var colCnt, flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.rowHeaders.columns.removeAt(colCnt - 1);
    }
    function addColumnHeader() {
        var rowCnt, flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.columnHeaders.rows.push(new wijmo.grid.Row());
    }
    function removeColumnHeader() {
        var flex = wijmo.Control.getControl('#mHeadersSheet');
        flex.columnHeaders.rows.removeAt(rowCnt - 1);
    }
</script>

<div>
    <button type="button" class="btn btn-default" onclick="addRowHeader()">Add Row
Header</button>
    <button type="button" class="btn btn-default" onclick="removeRowHeader()">Remove
Row Header</button>
    <button type="button" class="btn btn-default" onclick="addColumnHeader()">Add
Column Header</button>
    <button type="button" class="btn btn-default"
onclick="removeColumnHeader()">Remove Column Header</button>
    <br /><br />
    <c1-flex-sheet id="mHeadersSheet" class="flexSheet" is-read-only="false">
            <c1-unbound-sheet column-count="8" row-count="20"></c1-unbound-sheet>
        </c1-flex-sheet>
    </div>
```

Back to Top

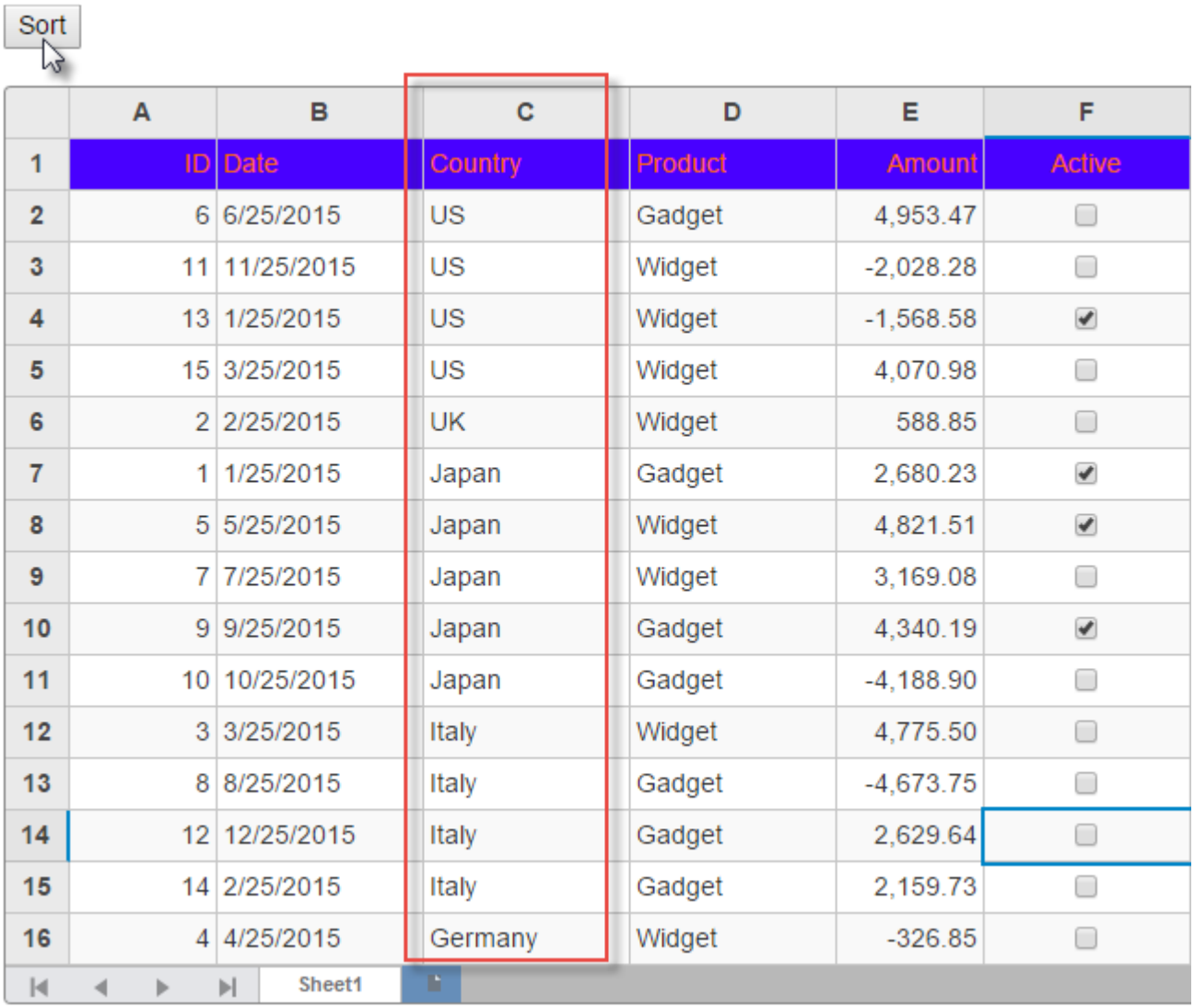
Sorting

FlexSheet supports sorting on the client side. Your FlexSheet data can be easily sorted by any column, to ensure effective reporting, browsing and analysis of data.

For instance, while analyzing sales report (in FlexSheet) for your organization, you might want to view the records in ascending order of Date or alphabetical order of Products. In such a situation, you need to sort the entries in the columns containing Date and Product entries in your FlexSheet. The below example demonstrates this scenario.

The SortManager property allows FlexSheet to manage the sort processing. You can set Order of sorting, Columns to be sorted and even change the sort order in the client side. Additionally, you can add further levels to sort multiple columns.

The following image shows a bound FlexSheet control, displaying date-wise Sales data for two products. In the below example, FlexSheet data gets sorted in descending order of entries in the third column, containing Country names (column index 2), on a button click.



The following code examples demonstrate how to enable Sorting in the FlexSheet:

In Code

Add a Sale.cs class to the Models folder.

Model

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 double Amount { get; set; }
    public bool Active { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "Germany", "UK",
"Japan", "Italy", "Greece" };
    private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget",
```

```
"Doohickey" };

    /// <summary>
    /// Get the data.
    /// </summary>
    /// <param name="total"></param>
    /// <returns></returns>
    public static IEnumerable<Sale> GetData(int total)
    {
        var rand = new Random(0);
        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 date = new DateTime(2015, i % 12 + 1, 25);

            return new Sale
            {
                ID = i + 1,
                Date = date,
                Country = country,
                Product = product,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Active = (i % 4 == 0)
            };
        });
        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;
    }
}
```

SortingController.cs

C# copyCode

```
public partial class SortController : Controller
{
    // GET: Sort
```

```
    public static List<Sale> SALES = Sale.GetData(15).ToList();

    public ActionResult SortIndex()
    {
        return View(SALES);
    }
}
```

Sorting.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<Sale>

<script>
    var flex, sortManager;
    c1.documentReady(function () {
        flex = wijmo.Control.getControl("#sortsheet");
        sortManager = flex.sortManager;
    });
    function sortFlex() {
        sortManager.sortDescriptions.items[0].columnIndex = 2;
        sortManager.sortDescriptions.items[0].ascending = false;
        sortManager.commitSort();
    }
</script>

<div>
    <input id="sort" type="button" onclick="sortFlex()" value="Sort" />
    <br /><br />
    @(Html.C1().FlexSheet().CssClass("flexSheet").Id("sortsheet")
        .AddBoundSheet(cv => cv.Bind(Model))
        )
</div>
```

Tag Helpers

HTML copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<sale>

<script>
        var flex, sortManager;
        c1.documentReady(function () {
            flex = wijmo.Control.getControl("#sortsheet");
            sortManager = flex.sortManager;
```

```
        });
        function sortFlex() {
            sortManager.sortDescriptions.items[0].columnIndex = 2;
            sortManager.sortDescriptions.items[0].ascending = false;
            sortManager.commitSort();
        }
</script>

<div>
    <input id="sort" type="button" onclick="sortFlex()" value="Sort" />
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="sortsheet">
        <c1-bound-sheet>
            <c1-items-source source-collection="Model"></c1-items-source>
        </c1-bound-sheet>
    </c1-flex-sheet>
</div>
```

Back to Top

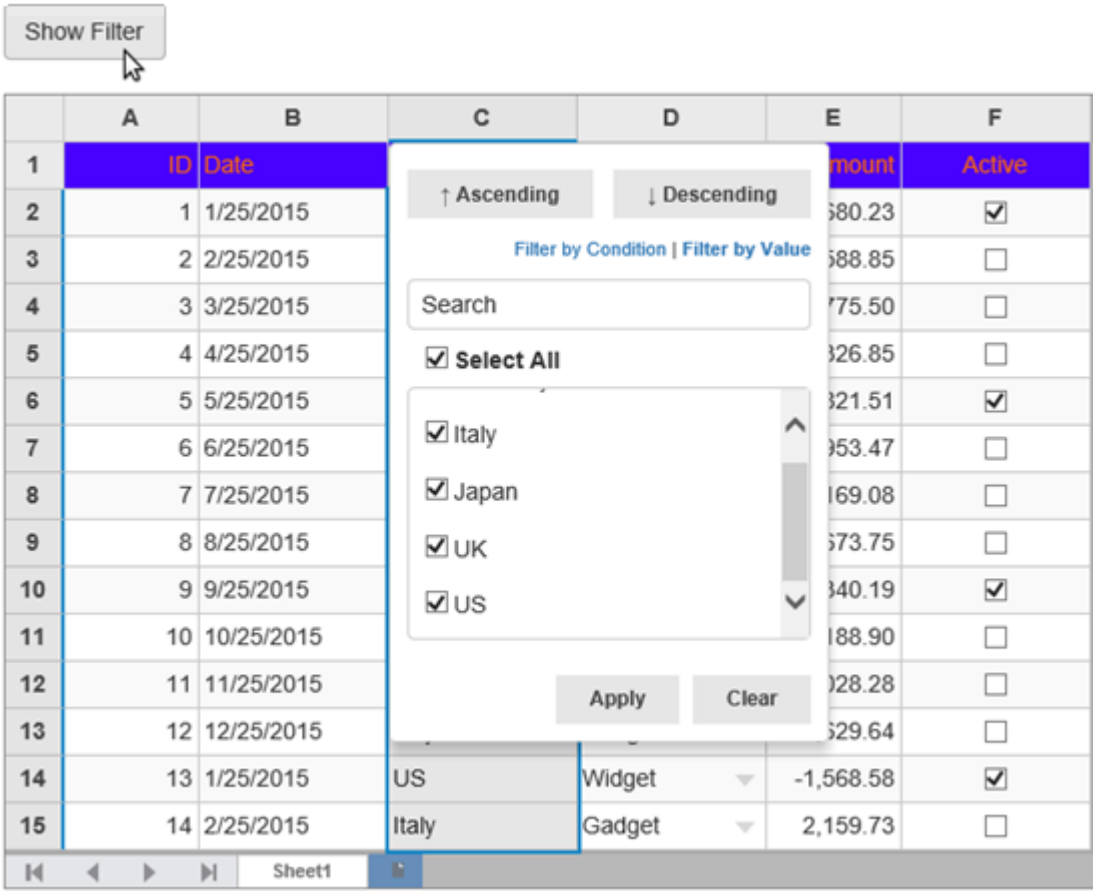
Filtering

FlexSheet supports filtering of column entries on the client side. It extends the FlexGrid filter, and enables you to apply **Condition filters** and **Value filters** to each column of your sheet. Condition filters and value filters are available in filter editor, which is displayed by invoking **showColumnFilter()** method.

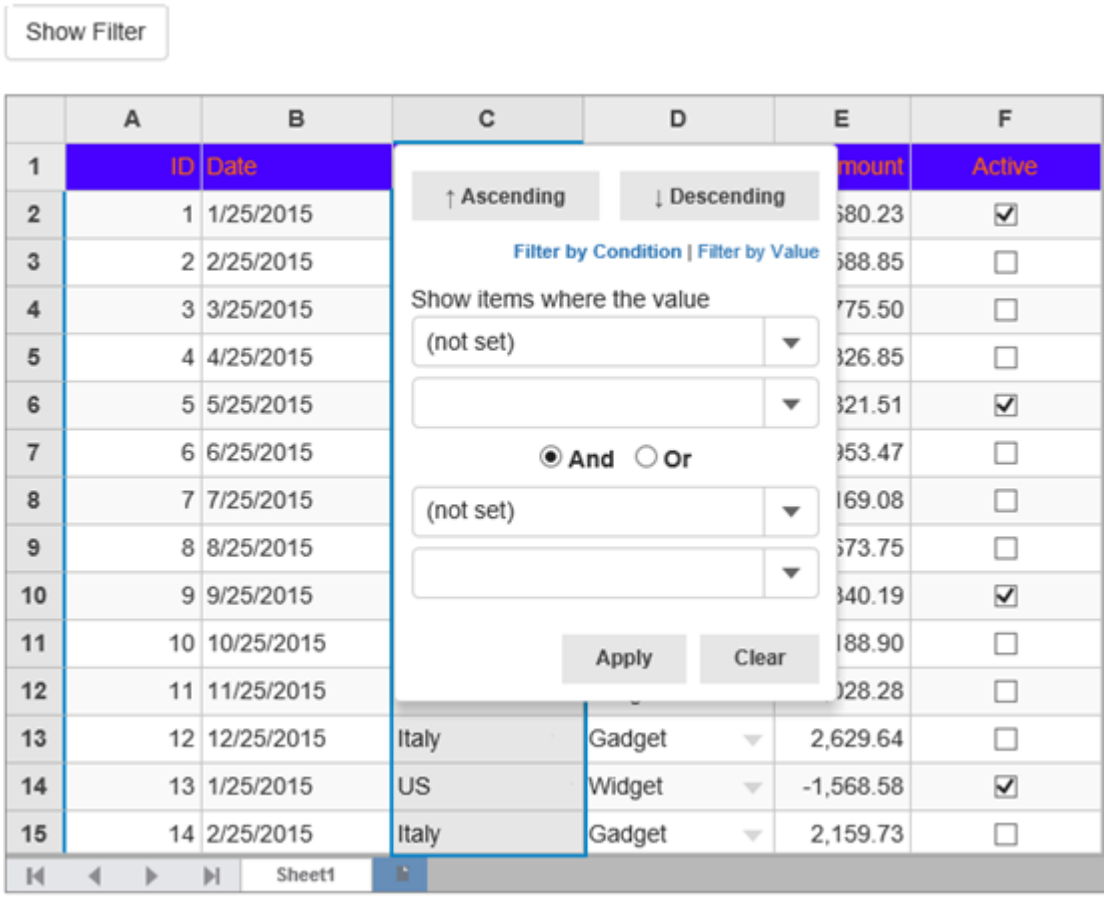
In the filter editor you can use **Filter by Condition** to apply conditions to narrow down your search, and **Filter by Value** to precisely locate data corresponding to the desired column value. The **Ascending** and **Descending** buttons enable you to sort the entries in a particular column in ascending and descending order respectively. Upon clicking the **Apply** button the filtered values are fetched, while on clicking the **Clear** button filter editor is closed.

This functionality helps users in quickly fetching the desired data corresponding to the specific column entries. It eases the task of analyzing and viewing complex data in a sheet. For example, from monthly sales data of an organization operating in four countries, you want to view sales details only for two countries. This can be easily accomplished by applying filter on the column containing country names. The below example demonstrates such a scenario.

The following image shows how a data bound FlexSheet control appears after applying **Filter by Value** on the selected column. The control in the below example displays data generated in Sale.cs model. The end users can select a particular column in the FlexSheet to display filter editor for it on a button click, and specify conditions or provide exact values to refine their search.



The following image shows a bound FlexSheet with Condition filter.



The following code examples demonstrate how to enable Filtering in the FlexSheet:

In Code

Add a Sale.cs class to the Models folder.

Model

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 double Amount { get; set; }
    public bool Active { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "Germany", "UK",
"Japan", "Italy", "Greece" };
    private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget",
"Doohickey" };

    /// <summary>
    /// Get the data.
    /// </summary>
```

```
    /// <param name="total"></param>
    /// <returns></returns>
    public static IEnumerable<Sale> GetData(int total)
    {
        var rand = new Random(0);
        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 date = new DateTime(2015, i % 12 + 1, 25);

            return new Sale
            {
                ID = i + 1,
                Date = date,
                Country = country,
                Product = product,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Active = (i % 4 == 0)
            };
        });
        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;
    }
}
```

FilteringController.cs

C# copyCode

```
public class FilterController : Controller
{
    // GET: Filter
    public static List<Sale> SALES = Sale.GetData(15).ToList();
    public ActionResult FilterIndex()
    {
        return View(SALES);
```

```
    }
}
```

Filtering.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<Sale>

<script>
    function filterFlex() {
        var flex = wijmo.Control.getControl("#fSheet");
        flex.showColumnFilter();
    }
</script>

<div>
    <input id="filter" type="button" onclick="filterFlex()" value="Show Filter" />
    <br /><br />
    @(Html.C1().FlexSheet().CssClass("flexSheet").Id("fSheet").IsReadOnly(false)
        .AddBoundSheet(sheet =>
            sheet.Bind(cv =>
                            cv.Bind(Model)))
    )
</div>
```

Tag Helpers

HTML copyCode

```
@using MVCFlexSheet.Models;
@model IEnumerable<sale>

    <script>
        function filterFlex() {
            var flex = wijmo.Control.getControl("#fSheet");
            flex.showColumnFilter();
        }
    </script>
<div>
    <input id="filter" type="button" onclick="filterFlex()" value="Show Filter" />
    <br /><br />
    <c1-flex-sheet class="flexSheet" id="fSheet" is-read-only="false">
            <c1-bound-sheet>
                <c1-items-source source-collection="Model"></c1-items-source>
            </c1-bound-sheet>
        </c1-flex-sheet>
```

```
    </div>
```

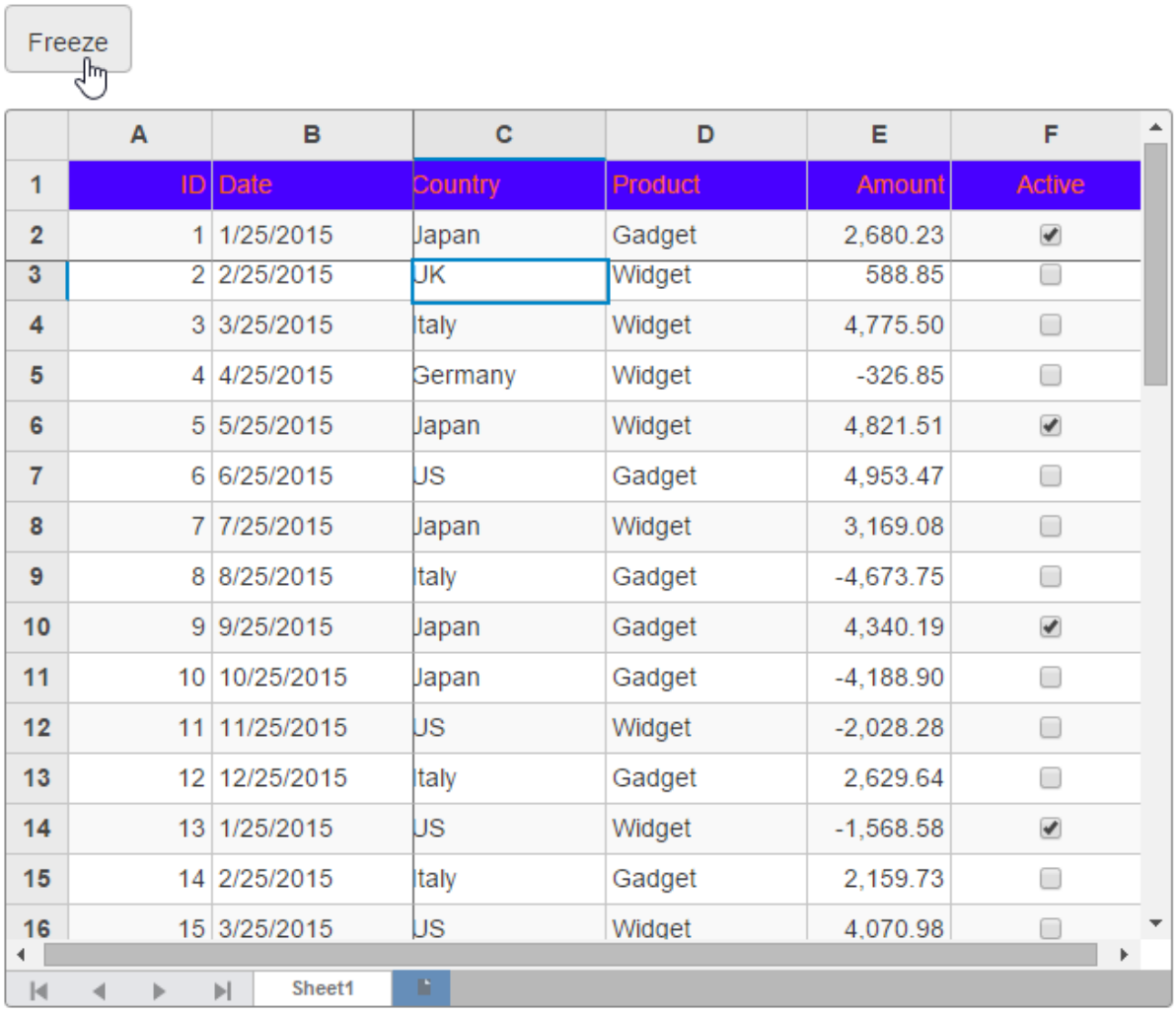
Back to Top

Frozen Cells

FlexSheet supports freezing panes on the client side, by locking rows and columns of a selected cell. This allows you to keep a particular area of your worksheet visible while you scroll to another area. It is possible through freezeAtCursor() method.

This is immensely helpful when you are working with huge data in your FlexSheet. For instance, you do not want the headings placed at the top to disappear, while you scroll too far down your worksheet. You simply need to select a cell, to lock or freeze the row and column containing that cell, and invoke the **freezeAtCursor()** method. However, if your FlexSheet control contains already frozen cells, the **freezeAtCursor()** will unfreeze them.

The following image shows a FlexSheet control, displaying Sales data, and Freeze button. In the below example, the **freezeAtCursor()** method is invoked on button click. When a user clicks the Freeze button after selecting a cell, the freezeAtCursor method is called and the cells in corresponding row and column are locked.



The following code examples demonstrate how to freeze cells in the FlexSheet:

In Code

Add a Sale.cs class to the Models folder.

Model

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 double Amount { get; set; }
    public bool Active { get; set; }

    private static List<string> COUNTRIES = new List<string> { "US", "Germany", "UK", "Japan",
"Italy", "Greece" };
    private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget", "Doohickey"
};

    /// <summary>
    /// Get the data.
    /// </summary>
    /// <param name="total"></param>
    /// <returns></returns>
    public static IEnumerable<Sale> GetData(int total)
    {
        var rand = new Random(0);
        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 date = new DateTime(2015, i % 12 + 1, 25);

            return new Sale
            {
                ID = i + 1,
                Date = date,
                Country = country,
                Product = product,
                Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                Active = (i % 4 == 0)
            };
        });
        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;
    }
}
```

FrozenController.cs

C# copyCode

```
public partial class FrozenCellsController : Controller
{
    // GET: FrozenCells
    public static List<Sale> SALES = Sale.GetData(50).ToList();

    public ActionResult FrozenIndex()
    {
        return View(SALES);
    }
}
```

FrozenCells.cshtml

HTML Helpers

Razor copyCode

```
@using C1MvcModelDB.Models
@model IEnumerable<Sale>

<script type="text/javascript">

    function freezeCells() {
        var flex = wijmo.Control.getControl("#freezeSheet");
        flex.freezeAtCursor();
    }
    </script>

<div>
<button type="button" class="btn btn-default" onclick="freezeCells()"
id="frozenBtn">Freeze</button>
    <br /><br />

@(Html.C1().FlexSheet().CssClass("flexSheet").Id("freezeSheet").Height("500px").Width("650px")
                .AddBoundSheet(cv => cv.Bind(Model)))

</div>
```

Tag Helpers

HTML copyCode

```
@using C1MvcModelDB.Models
@model IEnumerable<sale>

<script type="text/javascript">

        function freezeCells() {
            var flex = wijmo.Control.getControl("#freezeSheet");
            flex.freezeAtCursor();
```

```
        }
</script>

<div>
    <button type="button" class="btn btn-default" onclick="freezeCells()"
id="frozenBtn">Freeze</button>
    <br /><br />
    <c1-flex-sheet id="freezeSheet" class="flexSheet" Height="500px" Width="650px">
        <c1-bound-sheet>
            <c1-items-source source-collection="Model"></c1-items-source>
        </c1-bound-sheet>
    </c1-flex-sheet>
</div>
```

Back to Top

FlexSheet ASP.NET Core Tags

FlexSheet control supports the following ASP.NET Core Tags:

FlexSheet Class

- allow-delete
- allow-dragging
- allow-merging
- allow-resizing

AppendedSheets

- auto-clipboard
- column-layout
- class
- style
- defer-resizing
- disabled
- frozen-columns
- frozen-rows
- height
- id
- is-read-only
- scroll-position
- selection
- selection-mode
- show-marquee
- show-selected-headers
- selected-sheet-index
- is-tab-holder-visible
- file-path
- file-stream
- workbook
- load-action-url
- save-action-url
- save-content-type
- scroll-to-selection

- auto-sized-column
- auto-sized-row
- auto-sizing-column
- auto-sizing-row
- beginning-edit
- cell-edit-ended
- cell-edit-ending
- copied
- copying
- deleting-row
- dragged-column
- dragged-row
- dragging-column
- dragging-row
- format-item
- group-collapsed-changed
- groupCollapsedChanging
- items-source-changed
- loaded-rows
- loading-rows
- pasted
- pasting
- prepare-cell-for-edit
- resized-column
- resized-row
- resizing-column
- resizing-row
- row-added
- row-edit-ended
- row-edit-ending
- scroll-position-changed
- selection-changed
- selection-changing
- sorted-column
- sorting-column
- selected-sheet-changed
- dragging-row-column
- dropping-row-column
- loaded
- unknown-function
- remote-loading
- remote-loaded
- remote-saving
- remote-saved

UnBoundSheet

- column-count
- row-count
- name
- visible

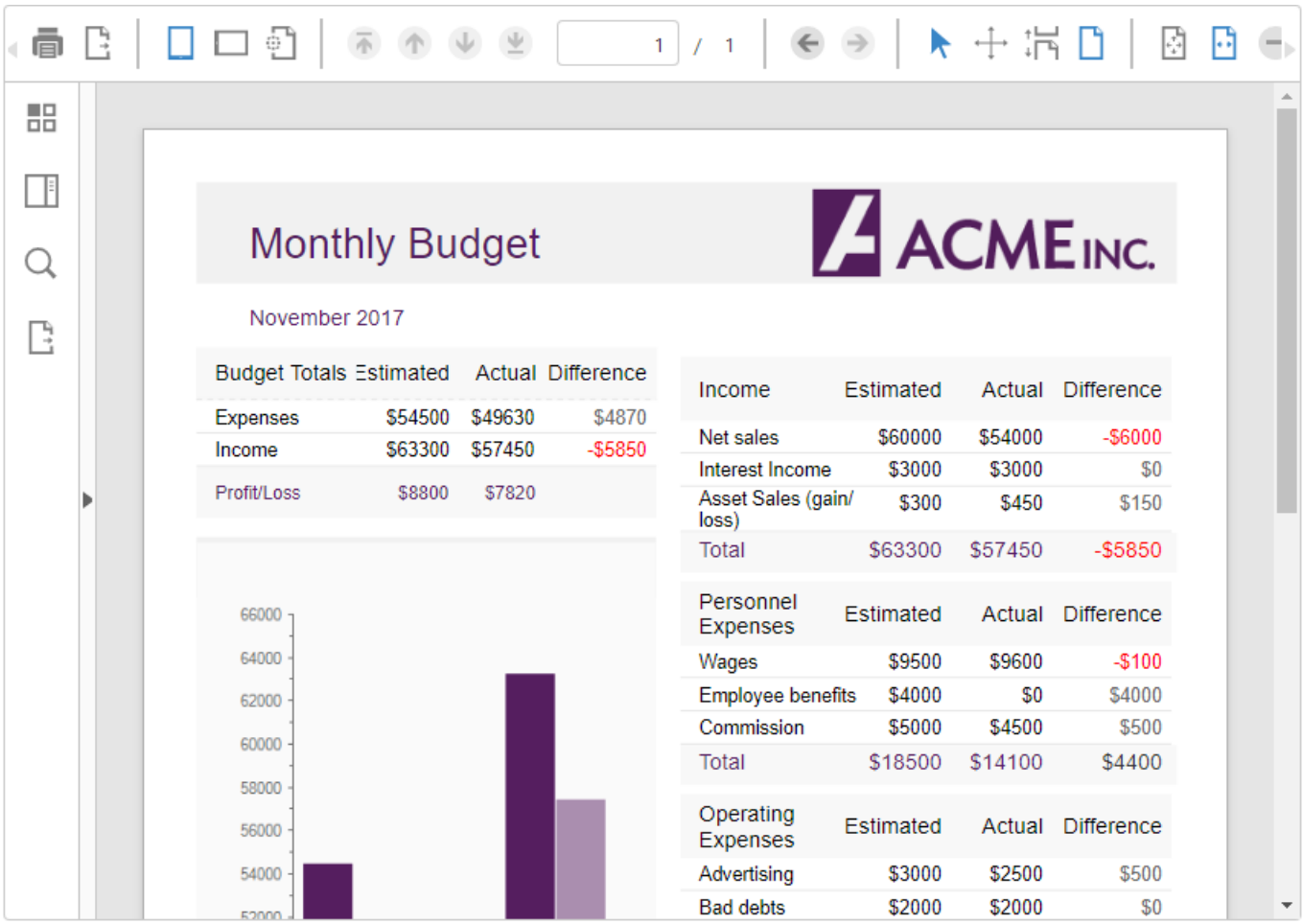
BoundSheet

- c1-items-source
- name
- visible

FlexViewer

ReportViewer

ReportViewer control for **ASP.NET MVC Edition** allows you to preview FlexReports in your web applications. ReportViewer is a fast and flexible HTML5 based report viewer that allows you to easily display C1Report, FlexReports, and SSRS Reports in your browsers. The ReportViewer control provides various options to view the report using the interactive and user-friendly user interface. Reports are rendered using the Scalable Vector Graphics (SVG) format which provides full print support and top-quality report viewing experience.

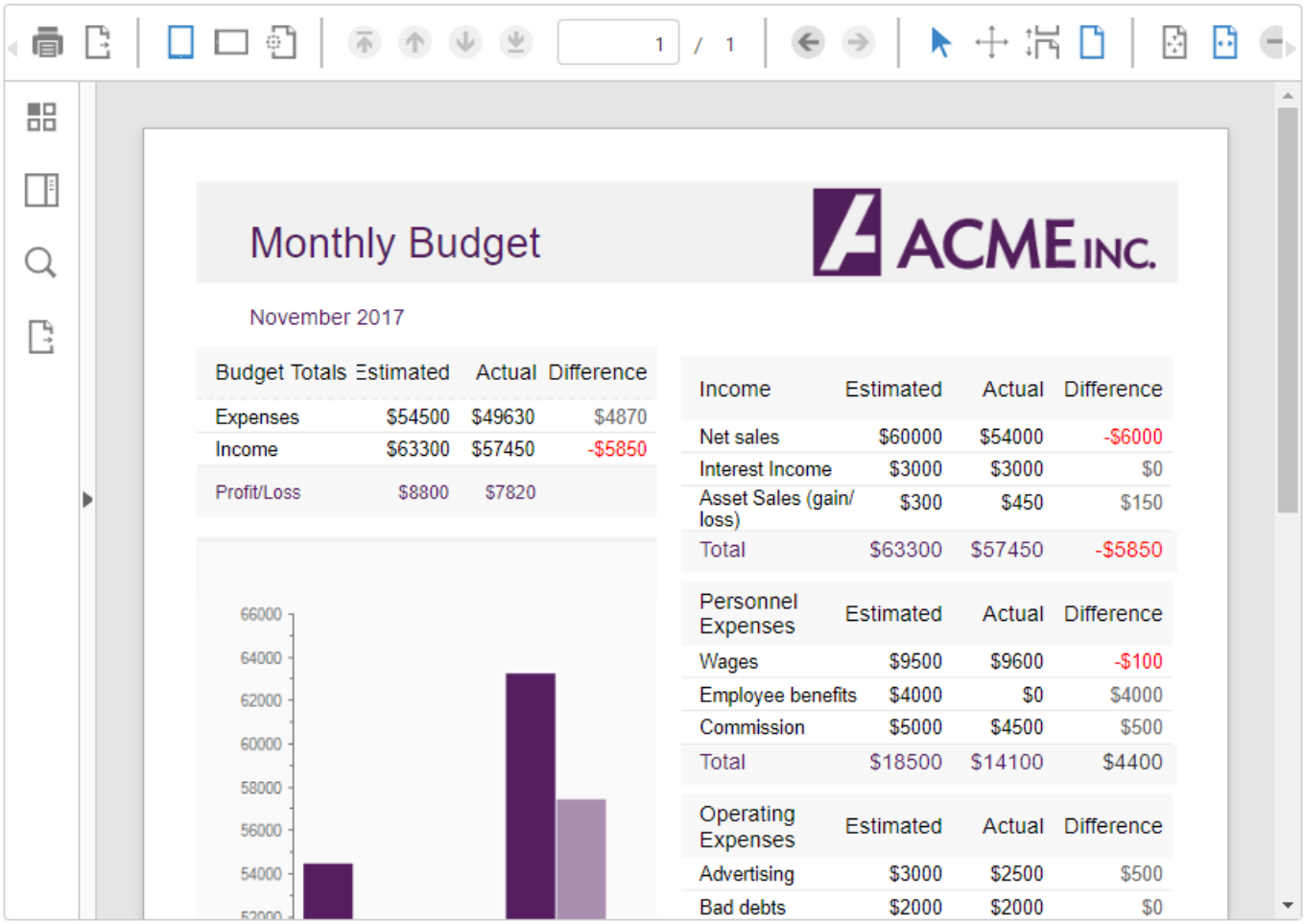


You can easily export a report to multiple formats using the Export options from the FlexViewer control. The export formats supported by FlexViewer control are Adobe PDF, HTML, RTF, Open XML Word, Open XML Excel, TIFF, BMP, PNG, JPEG, and GIF.

ReportViewer Elements

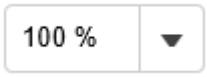
ReportViewer for ASP.NET MVC Edition is a single viewer for all your reports. There are lots of navigation tools available in the ReportViewer that helps the user to easily navigate and work with your report. Once you preview a report in the ReportViewer, there are lots of tools on sides of the ReportViewer. This topic will help you understand

the different navigation tools available on the ReportViewer.



The FlexViewer consists of the following elements

Elements	Name	Description
Toolbar Options - Consists of commonly used commands to perform basic operations on a report.		
	Print	Displays the Print dialog to specify printing options.
	Export	Displays the drop-down list of formats to export the report. The available formats are **Adobe PDF**, **HTML**, **RTF**, **Open XML Word**, **Open XML Excel**, **TIFF**, **BMP**, **PNG**, **JPEG**, and **GIF**.
	Portrait	Displays your report in portrait orientation.
	Landscape	Displays your report in Landscape orientation.
	Page Setup	Displays the Page Setup dialog that allows you to set the **Paper Kind**, **Orientation**, and **Margins(inches)** for the current selected page.

	Select Tool	Select the report content using a mouse cursor.
	Move Tool	Move the report content of the report.
	Continuous Page View	Displays your report in continuous scrolling page view.
	Single Page View	Fits the width of the page according to viewer dimensions.
	Fit Whole Page	Fits the whole page within the current viewer dimensions.
	Fit Page Width	Fits the width of the page according to viewer dimensions.
	Zoom In/Out	Zoom In - Increases the magnification of your report. Zoom Out - Decreases the magnification of your report.
	Zoom by Selection	Zooms to the current selection, ensuring that all selected entities are visible.
	Magnifier	Enlarges different parts of the report.
	Rotate Document	Rotates all the report pages.
	Rotate Page	Rotates the current selected page.
	Percentage Zoom	Allows you to set the zoom value based on percentage value.
	Full Screen	Open the report in full screen mode.
Sidebar Options - This section consists of Page Thumbnails, Document Map, Parameters and Search pane which are displayed for the report.		
	Page Thumbnails	The PageThumbnails pane appears by default in the sidebar when you click the **Toggle sidebar** from the Sidebar. This pane comprises of a thumbnail view of all the pages in a report. Click any thumbnail to navigate directly to the selected report page. You can also modify the size of the thumbnail when you click (+) or (-) button to zoom in and zoom out.

	Document Map	The Documents map pane appears when you click the **Document Map** button from the Sidebar. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.
	Parameters	The Parameters pane appears when you click the **Parameters** button from the Sidebar. FlexViewer allows you to view reports with parameters. If your report contains parameters, the Parameters pane shows up automatically. In Parameters pane, you are prompted to enter a value that filters the data to display.
	Search	The Search pane appears when you click the **Search** button from the Sidebar. Search pane allows you to search for a specific text in the report.

Using C1 MVC ReportViewer Template

This topic describes how to view a FlexReport in your MVC application using **C1 MVC ReportViewer** template. C1 MVC ReportViewer template provides three different options by which you can preview your report in FlexViewer;

- **Reports in current project**

  If you are working with **MVC 5** application, you can use the Reports in current project option in the C1 MVC ReportViewer template to add the report to your Visual Studio application, and then view it in FlexViewer.

  Once you create an application using C1 MVC Report Viewer, it automatically registers the required resources, adds the relevant Web API resources and packages to your application.

- **Report in other Report Service**

  If you are working with **MVC 3, 4, or ASP.NET Core** Framework applications, you can only use the Report in other report service option in the C1 MVC Report Viewer template to view the report in FlexViewer. You can use this option when your report is hosted on the Web API Service project

- **Reports in SSRS server**

  If you are working with **MVC 5 or ASP.NET Core** Framework applications, you can use the Reports in SSRS server option in the C1 MVC Report Viewer template to view the SSRS report in FlexViewer. You can use this option only for the SSRS reports that are hosted on a server.

Working with C1 MVC ReportViewer

The below steps demonstrates how you can use **C1 MVC ReportViewer** wizard to view a report in FlexViewer control using Visual Studio template.

- **Step 1: License your application**
- **Step 2: Register Resources**
- **Step 3: Add Controller**

Note: The **C1 ASP.NET MVC 5 Web Application** and **C1 ASP.NET Core MVC Application** template for ASP.NET MVC Edition automatically registers the required resources, and adds the relevant references and packages to your application. Therefore, you can directly use the **C1 MVC ReportViewer** template if your application is created using **ComponentOne** template.

Step 1: License your 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. In the **licenses.licx** file, add the following:

licenses.licx

```
C1.Web.Mvc.LicenseDetector, C1.Web.Mvc
C1.Web.Mvc.Viewer.LicenseDetector, C1.Web.Mvc.FlexViewer
```

Note: In case you are working with **ASP.NET Core** application, you can license the resources and your application using the **GrapeCity License Manager**. For more information, see Licensing topic.

Back to Top

Step 2: Register Resources

Complete the following steps to register the required resources for using ASP.NET MVC FlexViewer control:

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click `_Layout.cshtml` to open it.
3. Add the following code between the `<head></head>` tags.

ASP.NET

_Layout.cshtml

```
@Html.C1().Styles()@Html.C1().Scripts().Basic().FlexViewer()
```

ASP.NET Core

_Layout.cshtml

```
<c1-styles /><c1-scripts>        <c1-flex-viewer-scripts /></c1-scripts>
```

For more information on how to register resources for **FlexViewer**, refer to Registering Resources.

Back to Top

Step 3: Add Controller

Complete the following steps to add controller to your application.

ASP.NET

1. Right click the **Controllers** folder and select **Add** | **New Scaffolded Item...**.
2. In the **Add Scaffold** wizard select **MVC5 Controller - Empty**, and click **Add**.
3. Provide a name to the Controller. For example, we name the controller as **ReportController**.

A new controller is added to the application within the folder **Controllers**.

ASP.NET Core

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 the **Controller** tab towards left, and then select **MVC Controller - Empty**.
    2. Click **Add**, and then set a name for the controller. (for example: Report`Controller`)
    3. Click **Add**.

A new controller is added to the application within the folder **Controllers**.

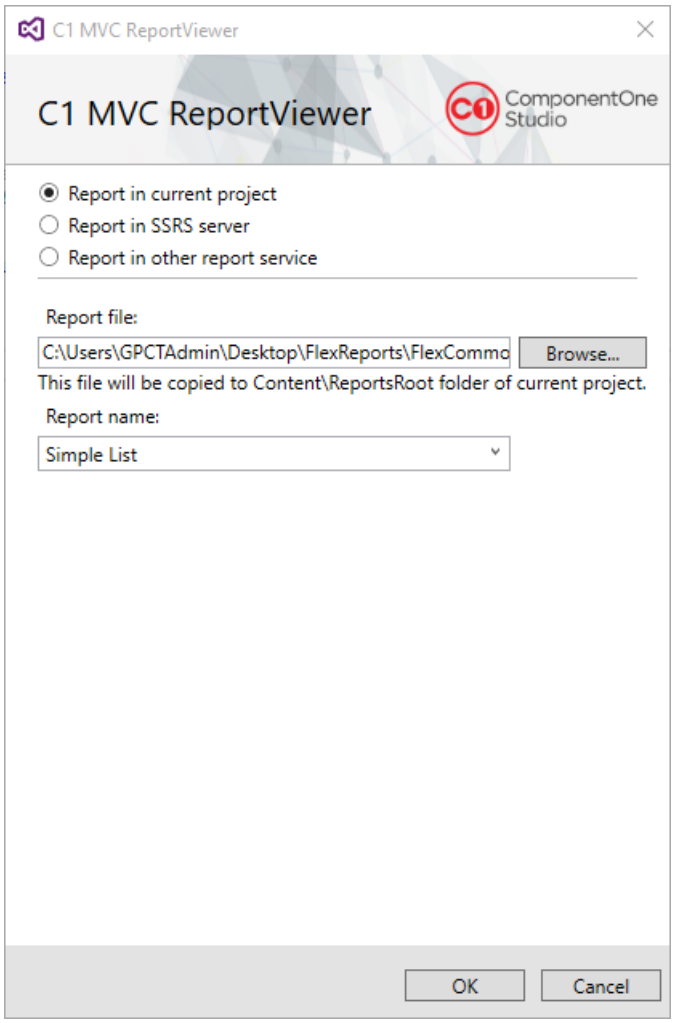
Views for the ReportViewer is discussed in the following topics:

Back to Top

View Reports in current project

To view a report that is stored in current project, you need to configure your MVC application. Complete the following steps to view reports that are stored in your current project. For more information about Report Viewer, see Using C1 ReportViewer Template.

1. Under **Views**, right-click the **Report** folder, and then select **Add | New Item...** to open the Add New Item dialog.
2. Under **Installed | Templates**, select **Visual C# | Web | C1 ReportViewer View Page** to open the C1 MVC ReportViewer dialog.
3. In the C1 MVC ReportViewer dialog, select **Report in current project** option.



4. In the **Report file** field, click **Browse...** to locate a FlexReport (.flxr) file on your system. In our case, we have used FlexCommonTasks.flxr report.
5. In the **Report name** drop down, select a **report name** from the list to view it in the FlexViewer. In our case, we have specified **Simple List** report.
6. Click **OK** to create the Index.cshtml view page.

Index.cshtml

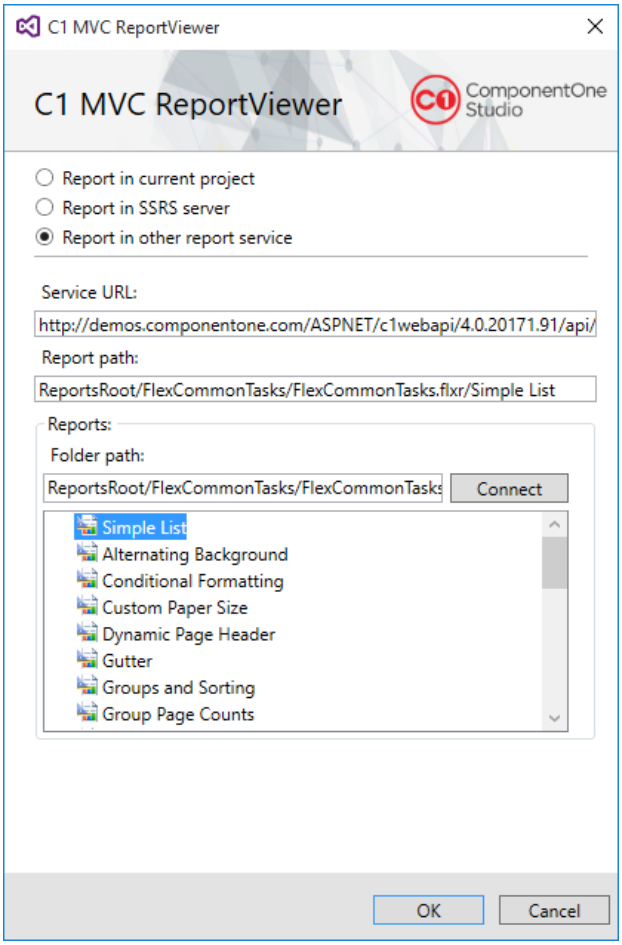
```
<head>
    <title>C1 MVC ReportViewer</title>
    @Html.C1().Styles()
    @Html.C1().Scripts().FlexViewer()
</head>
<body>
@(Html.C1().ReportViewer().FilePath(@"~/Content/ReportsRoot/FlexCommonTasks.flxr").ReportName(@"Simple List"))
</body>
```

View Reports using report service

To view a report that is stored on a report service, you need to configure your MVC application. Complete the following steps to add view page to your application. For more information about using Report Viewer, see Using C1 ReportViewer Template.

ASP.NET

1. Under **Views**, right-click the **Report** folder, and then select **Add | New Item...** to open the Add New Item dialog.
2. Under **Installed | Templates**, select **Visual C# | Web | C1 ReportViewer View Page** to open the C1 MVC ReportViewer dialog.
3. In the **C1 MVC ReportViewer** dialog, select **Report in other report service** option.



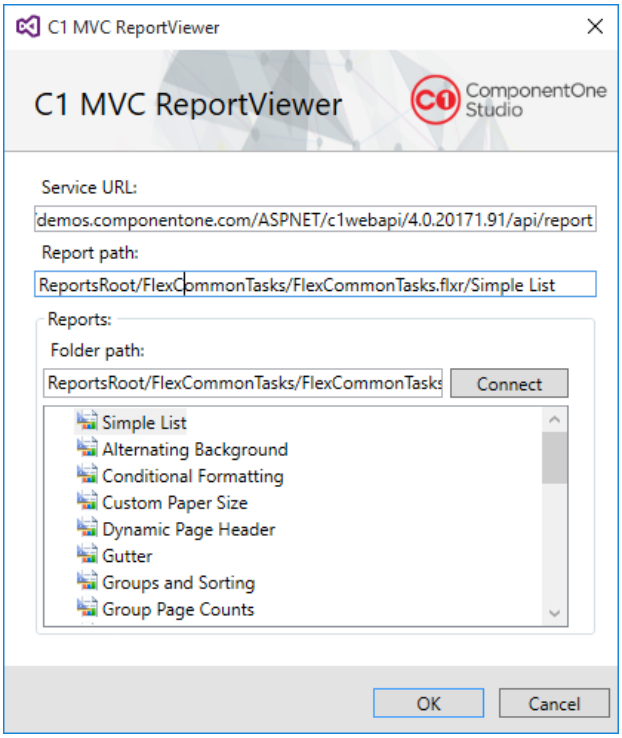
4. In the **Service URL** field, enter a Web API Service URL to access the reports. For example, http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report.
5. In the **Report Path** field, specify the report root location where the report is stored.
6. Click **Connect** to establish a connection to the WebApi server. Once you are connected to the WebApi server, **Reports** section displays all the report names.
7. Select a report from the list and, then click **OK** to create **Index.cshtml** view page.

Index.cshtml

```
<head>
    <title>C1 MVC ReportViewer</title>
    @Html.C1().Styles()
        @Html.C1().Scripts().FlexViewer()
</head>
<body>
@(Html.C1().ReportViewer().ServiceUrl(@"http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report")
.FilePath(@"ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr").ReportName(@"Simple List"))
</body>
```

ASP.NET Core

1. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
2. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: Report).
3. Right-click the **Report** folder, and then select **Add | New Item...** to open the Add New Item dialog.
4. Under **Installed | Templates**, select **.NET Core | C1 ReportViewer View Page (ASP.NET Core)** to open the C1 MVC ReportViewer dialog.



5. In the **Service URL**, enter the Web API Service URL to access the report file. For example, http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report.
6. In the **Report Path** field, specify the report root location where the reports are stored.
7. Click **Connect** to establish a connection to the WebApi server. Once you are connected to the WebApi server, **Reports** section displays all the report names.
8. Select a report from the list and, then click **OK** to create **Index.cshtml** view page.

Index.cshtml

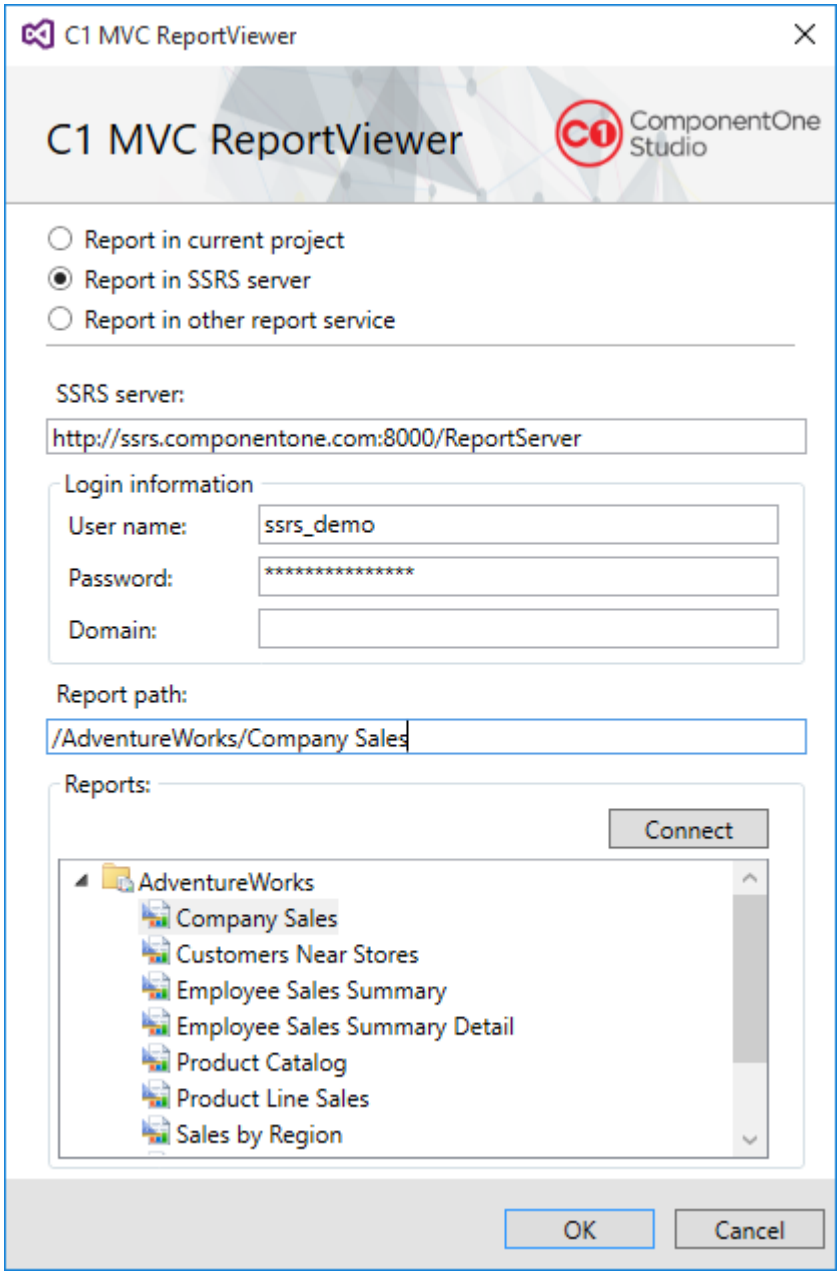
```
<head>
<title>C1 MVC ReportViewer</title>
<c1-styles />
<c1-scripts>
<c1-flex-viewer-scripts /></c1-scripts>
</head>
<body>
<c1-report-viewer file-path="ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr" report-name="Simple List"
service-url="http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report"></c1-report-viewer>
</body>
```

View SSRS reports

To view a report that is stored on a SSRS report server, you need to configure your MVC application. Complete the following steps to add the view page to your application. For more information about using Report Viewer, see Using C1 ReportViewer Template.

ASP.NET

1. Under **Views**, right-click the **Report** folder, and then select **Add | New Item...** to open the Add New Item dialog.
2. Under **Installed | Templates**, select **Visual C# | Web | C1 ReportViewer View Page** to open the C1 MVC ReportViewer dialog.
3. In the **C1 MVC ReportViewer** dialog, select **Report in SSRS server** option.



4. In the **SSRS server** field, enter a SSRS server URL to access the reports. For example, http://sh-tools-dev03/reportserver.
5. In the **Login information** section, specify the Username and the password to access the SSRS server URL.
6. In the **Report Path** field, specify the folder location where the SSRS reports is stored.
7. Click **Connect** to establish a connection to the SSRS server. Once you are connected to the SSRS server, **Reports** section displays all the report names.
8. Select a report from the list and, then click **OK** to create **Index.cshtml** view page.

Index.cshtml

```
<head>
    <title>C1 MVC ReportViewer</title>
    @Html.C1().Styles()
    @Html.C1().Scripts().FlexViewer()
</head>
<body>
@(Html.C1().ReportViewer().FilePath(@"SSRS/AdventureWorks/Company Sales"))
</body>
```

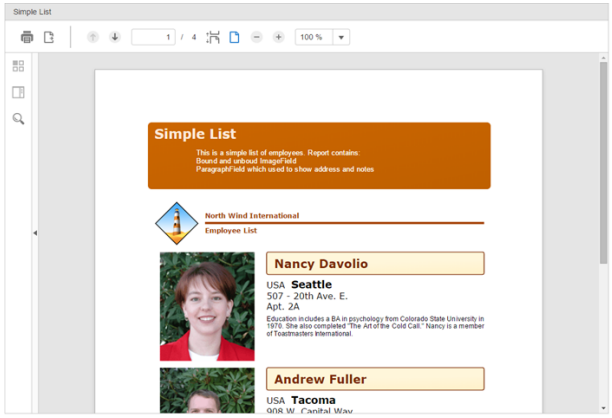
Manually Configuring ReportViewer

FlexViewer is available as MVC control that can be used in an MVC applications to view reports using C1 Web API Report Services. Complete the following steps to use the control to view a report.

- **Step 1: Create an MVC Application**
- **Step 2: License your application**
- **Step 3: Add the relevant references to your application**
- **Step 4: Configure the application to use FlexViewer**
- **Step 5: Register Resources**
- **Step 6: Add Controller**
- **Step 7: Add a View for the Controller**
- **Step 8: Build and Run the Project**

> **Note:** The **C1 ASP.NET MVC 5 Web Application** and **C1 ASP.NET Core MVC Application** template for ASP.NET MVC Edition automatically registers the required resources, and adds the relevant references and packages to your application. Therefore, you only need to follow Steps 6 to 8 above if your application is created using **ComponentOne** template.

The following image shows a report named Simple List, which can be viewed in FlexViewer on completing the steps above.



Step 1: Create an MVC Application

Create an ASP.NET Web Application (.NET Framework) using Visual Studio templates. For more information, see Configuring MVC application using Visual Studio template.

Back to Top

Step 2: License your 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. In the **licenses.licx** file, add the following:

licenses.licx

```
C1.Web.Mvc.LicenseDetector, C1.Web.Mvc
C1.Web.Mvc.Viewer.LicenseDetector, C1.Web.Mvc.FlexViewers
```

For more information on how to add license to your application, refer to Licensing.

Back to Top

Step 3: Add the relevant references to your application

ASP.NET

Complete the following steps to add the ASP.NET MVC Edition references and FlexViewer references to your project.

1. In the **Solution Explorer**, right click **References** and select **Add Reference.**
2. Browse to the location- **C:\Program Files (x86)\ComponentOne\ASP.NET MVC Edition\bin.**
3. Select **C1.Web.Mvc.dll** and **C1.Web.Mvc.Finance.dll**, and click **Add**.
4. Set the **Copy Local** property of the **C1.Web.Mvc.dll** and **C1.Web.Mvc.FlexViewer.dll** to **True**.

ASP.NET Core

1. In the **Solution Explorer**, right click **References** and select **Manage NuGet Packages**.
2. In **NuGet Package Manager**, select **GrapeCity** as the Package source. Search for **C1.AspNetCore.Mvc** package, and click **Install**. Refer to Configuring NuGet Package Sources for information on manually configuring GrapeCity NuGet source.
3. To work with **FlexViewer** control in your application, add **C1.AspNetCore.Mvc.FlexViewer** package. Once you restore the packages, "C1.AspNetCore.Mvc.FlexViewer" gets added under the "dependencies" in project.json file.

> **Note:** Once you have added the resources to ASP.NET Core application, you can license the resources and your application using the GrapeCity License Manager. For more information, see Licensing topic.

Back to Top

Step 4: Configure the application to use FlexViewer

ASP.NET

1. From the **Solution Explorer**, expand the folder **Views** and double click the `web.config` file to open it.
2. Add the following markups in <namespaces></namespaces> tags, within the `<system.web.webPages.razor></system.web.webPages.razor>` tags.

HTML

```
<add namespace="C1.Web.Mvc" />
<add namespace="C1.Web.Mvc.Viewer" />
<add namespace="C1.Web.Mvc.Viewer.Fluent" />
```

ASP.NET Core

1. From the **Solution Explorer**, expand the folder **Views** and double click the _ViewImports.cshtml file to open it.
2. Add the following references to work with FlexViewer control in your ASP.NET Core application.

_ViewImports

```
@addTagHelper *, C1.AspNetCore.Mvc
@addTagHelper *, C1.AspNetCore.Mvc.FlexViewer
```

Back to Top

Step 5: Register Resources

Complete the following steps to register the required resources for using ASP.NET MVC FlexViewer control:

1. From the **Solution Explorer**, open the folders **Views | Shared**.

2. Double click `_Layout.cshtml` to open it.
3. Add the following code between the `<head></head>` tags.

ASP.NET

_Layout.cshtml

```
@Html.C1().Styles()
@Html.C1().Scripts().FlexViewer()
```

ASP.NET Core

_Layout.cshtml

```
<c1-styles> <c1-scripts>
<c1-flex-viewer-scripts /></c1-scripts>
```

For more information on how to register resources for **FlexViewer**, refer to Registering Resources.

Back to Top

Step 6: Add Controller

Complete the following steps to add controller to your application:

ASP.NET

1. Right click the **Controllers** folder and select **Add | New Scaffolded Item...**.
2. In the **Add Scaffold** wizard select **MVC5 Controller - Empty**, and click **Add**.
3. Provide a name to the Controller. For example, we name the controller as **ReportController**.

A new controller is added to the application within the folder **Controllers**.

ASP.NET Core

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 New Item** dialog:
    1. Select the **Controller** tab towards left, and then select **MVC Controller - Empty**.
    2. Click **Add**, and then set a name for the controller. (for example: Report`Controller`)
    3. Click **Add**.

A new controller is added to the application within the folder **Controllers**.

Back to Top

Step 7: Add a View for the Controller

Complete the following steps to add corresponding view for the controller.

ASP.NET

1. Place the cursor inside the method Index() within your controller (in this example: ReportController).
2. Right click and select **Add View** from the options. The **Add View** dialog appears.
3. In the **Add View** dialog, set a **View name**. For example, Index in this example.
4. Click **Add**.
5. Once the **index.cshtml** page is added to your project, add the following code to view your report in the FlexViewer.

A view is added for the controller. In the code below, we have specified the **Service URL**, **FilePath**, and **Report Name**.

Index.cshtml

```
@(Html.C1().ReportViewer().ServiceUrl(@"http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report").FilePath(@"ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr").ReportName(@"Simple List"))
```

ASP.NET Core

1. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
2. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: Report).
3. Right click the **Report** folder, and select **Add | New Item**. The **Add New Item** dialog appears.
4. Complete the following steps in the **Add New Item** dialog:
    1. Expand the **Installed** tab towards left, and select **ASP.NET|MVC View Page**.
    2. Set name of the view (for example: Index.cshtml).
    3. Click **Add**.
5. Once the **index.cshtml** page is added to your project, add the following code to view your report in the FlexViewer.

A view is added for the controller. In the code below, we have specified the **Service URL**, **FilePath**, and **Report Name**.

Index.cshtml

```
<c1-report-viewer file-path="ReportsRoot/FlexCommonTasks/FlexCommonTasks.flxr"
report-name="Simple List" service-url="http://demos.componentone.com/ASPNET/c1webapi/4.0.20171.91/api/report">
</c1-report-viewer>
```

A view is added for the controller.

Back to Top

Step 8: Build and Run the Project

1. Click **Build** | **Build Solution** to build the project.
2. Press **F5** to run the project, and view your report in FlexViewer. For more information about FlexViewer, see FlexViewer Elements.

> **Note:** Append the folder name and view name to the generated URL (for example: http://localhost:1234/Report/Index) in the address bar of the browser to view the report.

> **Note:** To know further, explore the detailed demo for using FlexViewer.

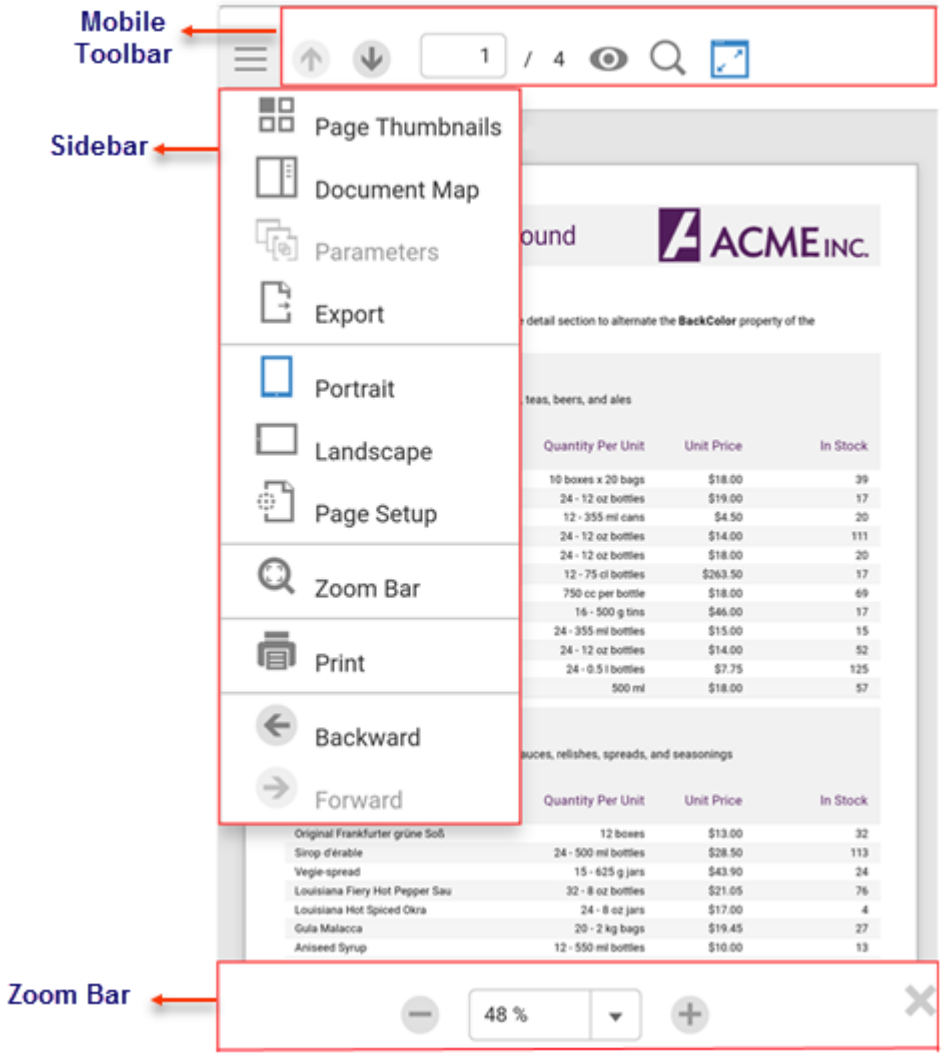
Back to Top

Mobile View

ReportViewer control for ASP.NET MVC Edition supports MobileView mode. MobileViewer helps you to render and view your reports on any hand held devices. It is a fast and flexible HTML5 based report viewer that allows you to display C1Report, FlexReports, and SSRS Reports. The responsive layout adapts itself to the device display for best viewing experience. It provides various options to view the report using the interactive and user-friendly user interface. For more information about the user interface, see MobileViewer Elements.

MobileViewer Elements

MobileViewer for ASP.NET MVC Edition helps you to view and render your reports on any hand held device. There are different navigation tools available in the MobileViewer which helps the user to easily navigate and work with the report. Once you preview the report in MobileViewer, you can use the options in the side menu while working with the report. This topic will help you understand the different navigation tools available on the MobileViewer.



The MobileViewer consists of the following elements:

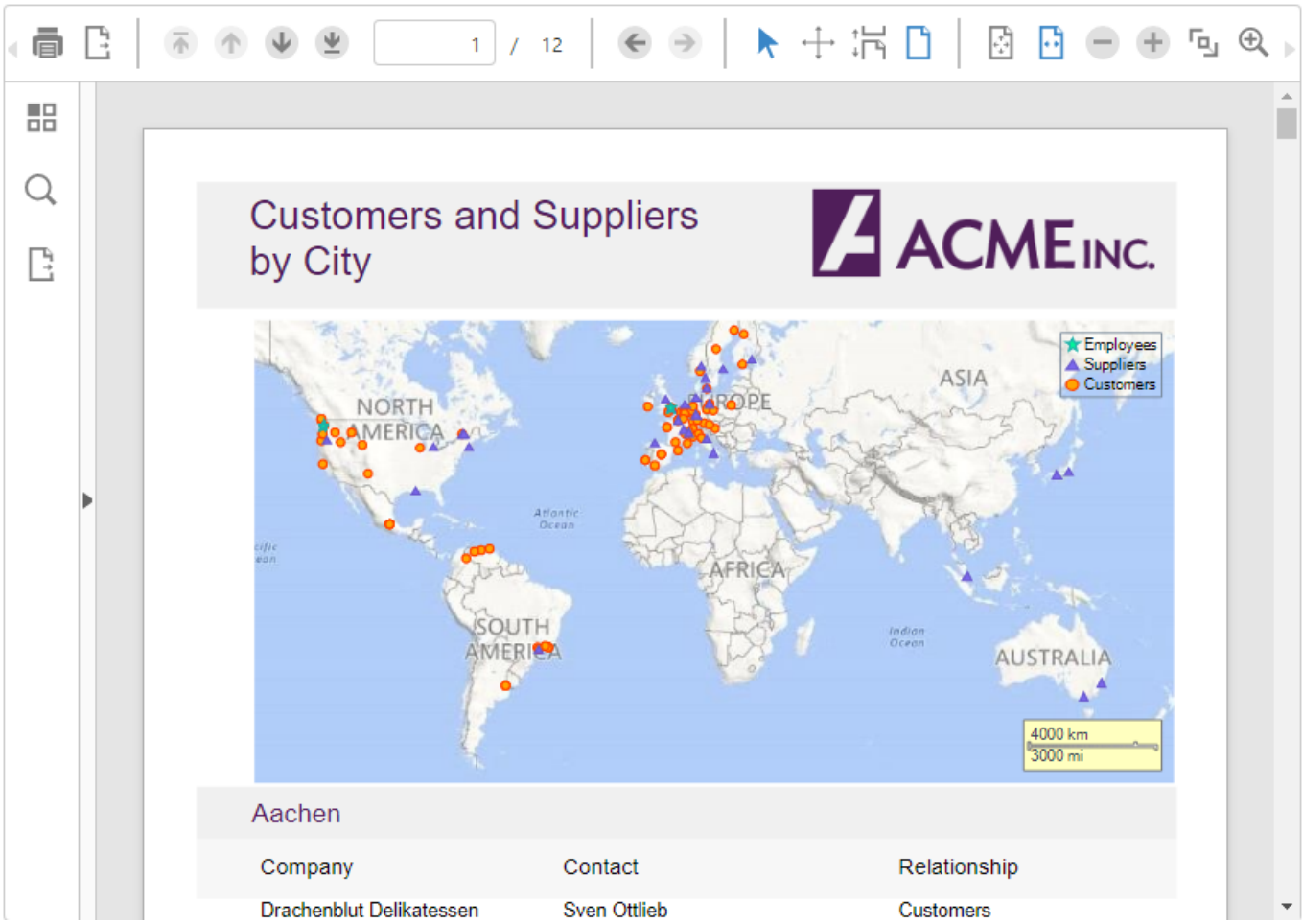
Elements	Name	Description
Mobile Toolbar Options - This section consists of commonly used commands to perform basic operations on a report. All commands provided by the toolbar operate on the report currently shown in the viewer.		
	Previous Page/Next Page	Allows the user to navigate the report by moving to the next page or previous page.
	Layout Options	Layout Options consists Single Page View, Continuous Page View, Fit Whole Page, and Fit Page Width options.
	Single Page View	Fits the width of the page according to viewer dimensions.
	Continuous Page View	Displays your report in continuous scrolling page view.
	Fit Whole Page	Fits the whole page within the current viewer dimensions.

	Fit Page Width	Fits the width of the page according to viewer dimensions.
	Show Search Bar	Click the **Show Search Bar** button from the Toolbar to display the search bar at the bottom of the Viewer. Search Bar allows you to search for a specific text in the report.
	Full Screen	Displays your report in Full Screen mode.
Sidebar Options - This section consists of Page Thumbnails, Document Map, Parameters and Search pane which are displayed for the report.		
	Page Thumbnails	The PageThumbnails pane appears by default in the sidebar when you click the **Toggle sidebar** from the Sidebar. This pane comprises of a thumbnail view of all the pages in a report. Click any thumbnail to navigate directly to the selected report page.
	Document Map	The Documents map pane appears when you click the **Document Map** button from the Sidebar. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.
	Parameters	The Parameters pane appears when you click the **Parameters** button from the Sidebar. FlexViewer allows you to view reports with parameters. If your report contains parameters, the Parameters pane shows up automatically. In Parameters pane, you are prompted to enter a value that filters the data to display.
	Export	The Export pane appears when you click the **Export** button from the Sidebar. Export pane allows you to set different option before you export your report. The available formats are Adobe PDF, HTML, RTF, Open XML Word, Open XML Excel, TIFF, BMP, PNG, JPEG, and GIF.
	Portrait	Displays your report in portrait orientation.
	Landscape	Displays your report in landscape orientation.
	Page Setup	Displays the Page Setup pane that allows you to set the **Paper Kind**, **Orientation**, and **Margins(inches)** for the current selected page.

	Zoom Bar	The ZoomBar appears at the bottom of the Viewer when you click the ZoomBar option from the Sidebar.
	Print	Displays the Print dialog to specify printing options.
Zoom Bar Options - This section consists of Zoom In/Out and Percentage Zoom options for the report.		
	Zoom In/Out	Zoom In - Increases the magnification of your report. Zoom Out - Decreases the magnification of your report.
100 %	Percentage Zoom	Allows you to set the zoom value based on percentage value.

PDFViewer

PDFViewer for ASP.NET MVC Edition enables the users to view PDF files on a web browser with the help of **C1 MVC PDFViewer** template. The PDFViewer provides many different options to view a PDF file using it's interactive features and user-friendly user interface. PDFViewer is a fast and flexible HTML5 based PDF viewer that allows the user to display PDF files in the web browser. Essentially, PDFViewer is a native ASP.NET MVC control that has no dependency on Adobe. It's lightweight and easy to distribute with your application.



PDFViewer also allows you to easily export PDF file to multiple file formats using the Export option from the Viewer. The export formats supported by PDFViewer control are HTML, Meta file, TIFF, BMP, PNG, JPEG, and GIF.

PDFViewer Elements

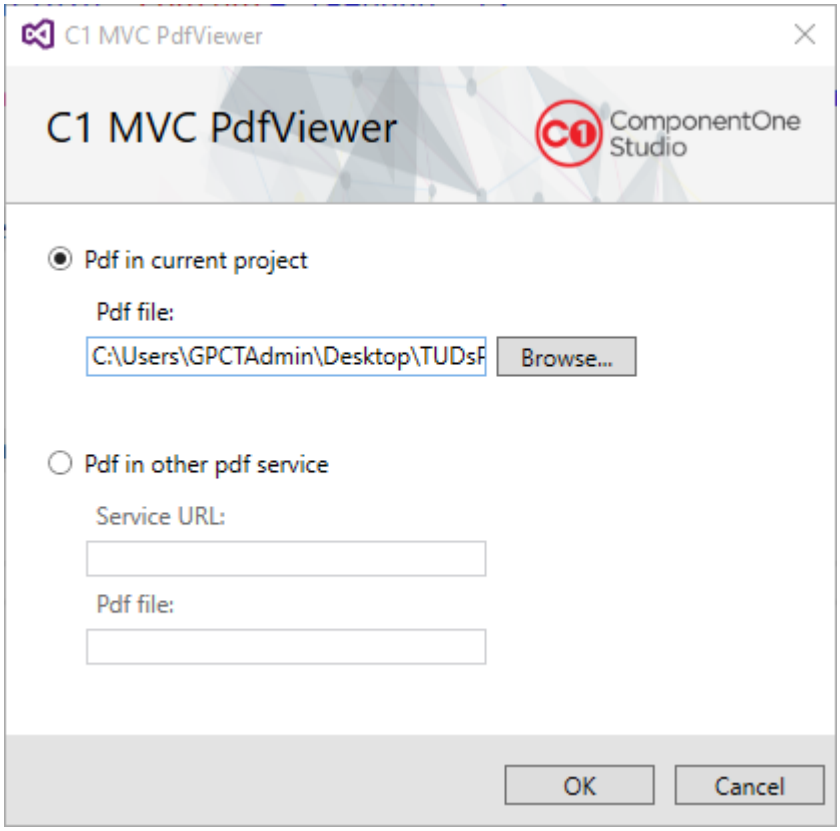
PDFViewer for ASP.NET MVC Edition helps the user to preview PDF files on the web browser. Once you preview a PDF file in the PDFViewer, you will find many different tools on sides of the PDFViewer. There are plenty of navigation tools and options available in the PDFViewer that helps the user to easily navigate and work with PDF files. This topic will help you understand the different navigation and interaction tools available on the PDFViewer.

Using C1 MVC PdfViewer

This topic describes how to view a Pdf file in your MVC application using **C1 PdfViewer** template. The C1 PdfViewer template provides two different options to preview PDF file on the web browser. Users can preview the Pdf files that are stored on local system using the **Pdf in current project** option or can view the Pdf files that hosted on WebApi service URL using **Pdf in other Pdf service** option.

ASP.NET

1. Under **Views**, right-click the **PDFViewer** folder, and then select **Add | New Item...** to open the Add New Item dialog.
2. Under **Installed | Templates**, select **Visual C# | Web | C1 PdfViewer View Page** to open the C1 MVC PdfViewer dialog.
3. In the **C1 MVC PdfViewer** dialog, select **Pdf in current project** option.



4. In the Pdf file field, click **Browse...** to locate a **Portable Document File(.pdf)** file on your system. In our case, we have used a sample pdf document.

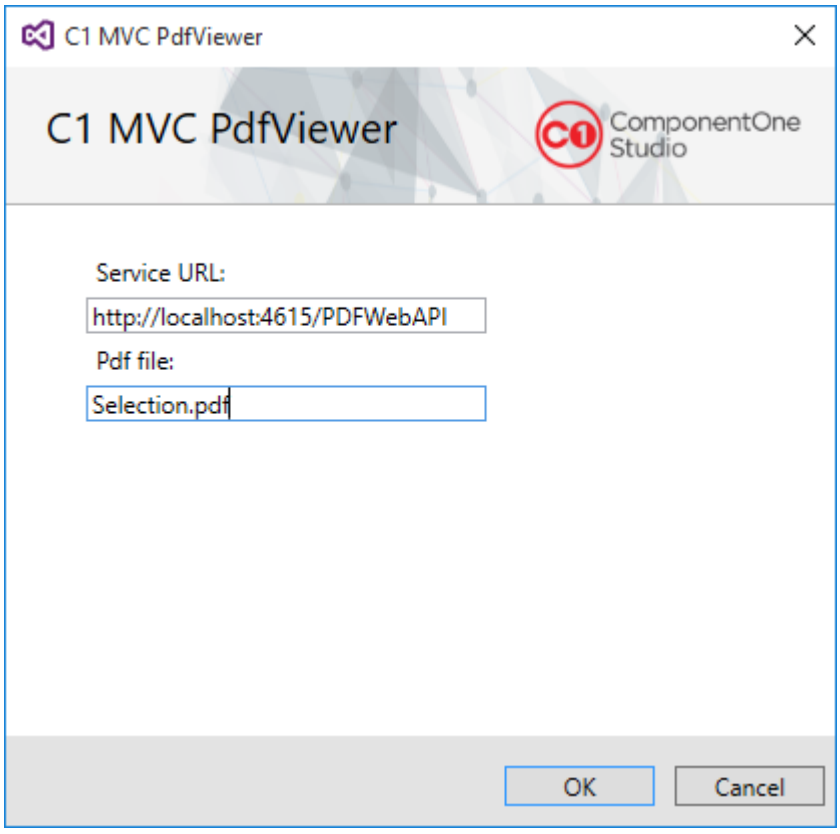
5. Click **OK** to create **Index.cshtml** view page.

Index.cshtml

```
<head>
    <title>C1 MVC PdfViewer</title>
    @Html.C1().Styles()
    @Html.C1().Scripts().FlexViewer()
</head>
<body>
@(Html.C1().PdfViewer().FilePath(@"~\Content\PdfFilesRoot\Selection.pdf"))
</body>
```

ASP.NET Core

1. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
2. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: PDFViewer).
3. Right-click the **PDFViewer** folder, and then select **Add | New Item...** to open the Add New Item dialog.
4. Under **Installed | Templates**, select **.NET Core | C1 PdfViewer View Page (ASP.NET Core)** to open the C1 MVC PdfViewer dialog.



5. In the **Service URL**, enter the Web API Service URL to access the Pdf file.
6. In the Pdf file field, enter the name of the Pdf file that you want to access from the service URL.
7. Click **OK** to create **Index.cshtml** view page.

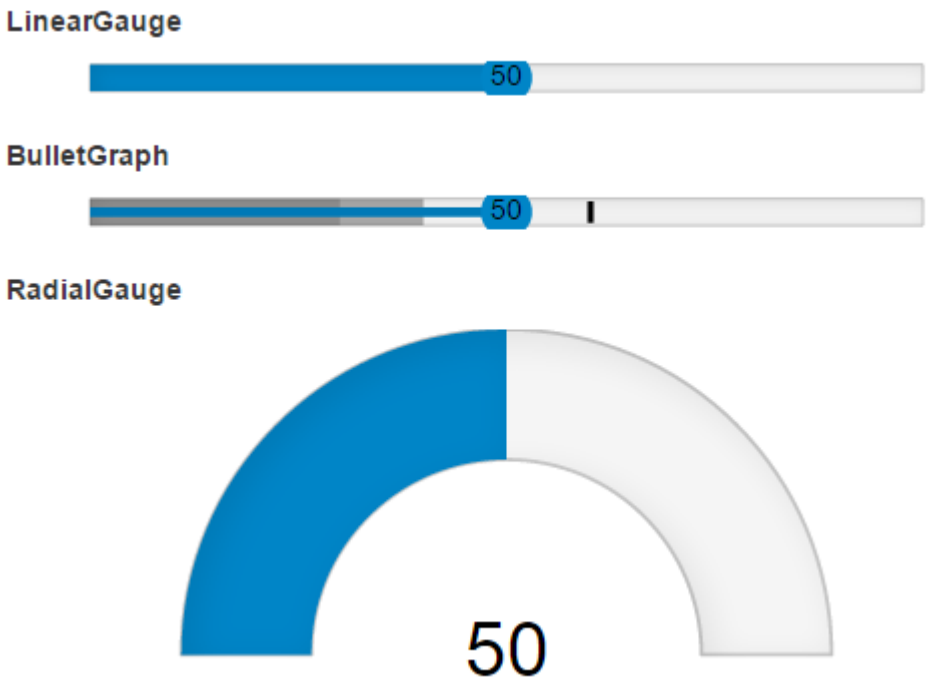
Index.cshtml

```
<head>
    <title>C1 MVC PdfViewer</title>
    <c1-styles />
```

```
    <c1-scripts>
        <c1-flex-viewer-scripts />
    </c1-scripts>
</head>
<body>
    <c1-pdf-viewer file-path="PDF/Selection.pdf" service-
url="http://localhost:4615/PDFWebAPI"></c1-pdf-viewer>
</body>
</html>
```

Gauge

The gauge control allows you to display information in a dynamic and unique way by delivering the exact graphical representation you require. Gauges are better than simple labels because they also display a range, allowing users to determine instantly whether the current value is low, high, or intermediate.

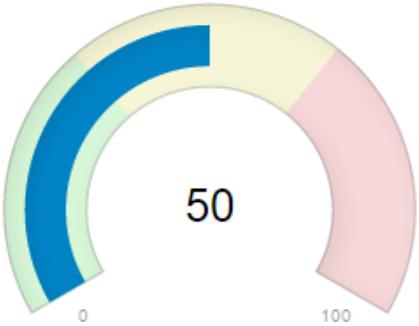


Key Features

- **Easy Customization**: Restyle the Gauges by changing a property to create gauge with custom colors, fills and more.
- **Ranges**: Add colored ranges to the gauge to draw attention to a certain range of values. Use simple properties to customize their start and end points, as well as appearance.
- **Direction**: Customize pointer direction of the LinearGauge and BulletGraph.
- **Pointer Customization**: Customize pointer color, thickness and more to make the gauge more appealing.

Gauge Types

ASP.NET MVC Edition controls comprise three kinds of gauges: Linear Gauge, Radial Gauge and Bullet Graph.

Type	Image	Usage
Linear Gauge: A linear gauge displays the value along a linear scale, using a linear pointer.		A linear gauge is commonly used to denote data as a scale value such as length, temperature, etc.
Radial Gauge: A radial gauge displays the value along a circular scale, using a curved pointer. The scale can be rotated as defined by the `StartAngle` and SweepAngle properties.		A radial gauge is commonly used to denote data such as volume, velocity, etc.
Bullet Graph: A bullet graph displays a single value on a linear scale, along with a target value and ranges that instantly indicate whether the value is good or bad using the Good and Bad properties.		A bullet graph is a variant of a linear gauge, designed specifically for use in dashboards that display a number of single value data, such as yearly sales revenue.

Quick Start: Add and Configure

BulletGraph Quick Start

This section describes how to add a BulletGraph control to your MVC webform and set its value. For information on how to add a control, See Adding controls.

This topic comprises of three steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Control**
- **Step 4: Build and Run the Project**

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

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Add a Model

Models are required to fetch data for the controls. See MVC Basics for more information.

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: `Class1.cs`).
3. Click **Add**. A new class is added to the application.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **BulletGraph** control and set the following properties:

    - Set the gauge `Value` to **35**. The `Value` property denotes the gauge's current value.
    - Set the Min and `Max` properties to 0 and 100 respectively. The Min and `Max` properties specify the range of the gauge. You can add multiple ranges to a single gauge.

- Set the `Target`, `Good` and `Bad` properties to 70, 60 and 40 respectively.
- Set the Gauge's `isReadOnly` property to **False**. The isReadOnly specifies whether a user can edit the gauge's `Value` using a keyboard or a mouse.

HTML Helpers

Razor copyCode

```
@(Html.C1().BulletGraph()
    .Min(0).Max(100)
    .Good(60)
    .Bad(40)
    .Value(50)
    .Target(70)
    .IsReadOnly(false).Step(1).Width(500)
    .IsAnimated(true)
)
```

Tag Helpers

HTML copyCode

```
<c1-bullet-graph min=0 max=100 good=60 bad=40 value=50 target=70 width="500px"
                 is-read-only="false" step="1" is-animated="true">
</c1-bullet-graph>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

LinearGauge Quick Start

This section describes how to add a LinearGauge control to your MVC webform and set its value. For information on how to add a control, See Adding controls.

This topic comprises of three steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Control**
- **Step 4: Build and Run the Project**

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

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **LinearGauge** control and set the following properties:
    - Set the gauge `Value` to **35**. The `Value` property denotes the gauge's current value.
    - Set the Min and `Max` properties to 0 and 100 respectively. The Min and `Max` properties specify the range of the gauge. You can add multiple ranges to a single gauge.
    - Set the Gauge's `isReadOnly` property to **False**. The isReadOnly specifies whether a user can edit the gauge's `Value` using a keyboard or a mouse.

HTML Helpers

Razor copyCode

```
@(Html.C1().LinearGauge()
    .Width(500)
    .Height(100)
    .Value(35)
    .Thickness(0.1)
    .Min(0).Max(100)

    .Direction(C1.Web.Mvc.GaugeDirection.Right)
```

```
        // Create and Customize Ranges
         .Ranges(items => items
            .Add(item => item.Min(0).Max(40).Color(System.Drawing.Color.Red))
            .Add(item =>
item.Min(40).Max(80).Color(System.Drawing.Color.Yellow))
            .Add(item =>
item.Min(80).Max(100).Color(System.Drawing.Color.Green))
        )
)
```

Tag Helpers

HTML copyCode

```
<c1-linear-gauge width="500px" height="100px" value="35"
                 thickness="0.1" min="0" max="100"
direction="C1.Web.Mvc.GaugeDirection.Right">
    <c1-gauge-range min="0" max="40" color="red"></c1-gauge-range>
    <c1-gauge-range min="40" max=80 color="yellow"></c1-gauge-range>
    <c1-gauge-range min="80" max="100" color="green"></c1-gauge-range>
</c1-linear-gauge>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

RadialGauge Quick Start

This section describes how to add a RadialGauge control to your MVC webform and set its value. For information on how to add a control, See Adding controls.

This topic comprises of three steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Control**
- **Step 4: Build and Run the Project**

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

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Add a Model

Models are required to fetch data for the controls. See MVC Basics for more information.

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: `Class1.cs`).
3. Click **Add**. A new class is added to the application.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **RadialGauge** and set the following properties:

- Set the gauge `Value` to **35**. The `Value` property denotes the gauge's current value.
- Set the Min and `Max` properties to 0 and 100 respectively. The Min and `Max` properties specify the range of the gauge. You can add multiple ranges to a single gauge.
- Set the Gauge's `isReadOnly` property to **False**. The isReadOnly specifies whether a user can edit the gauge's `Value` using a keyboard or a mouse.
- Set the StartAngle property to **-20**, and SweepAngle property to **220**. The StartAngle property specifies the RadialGauge's starting angle and the SweepAngle property specifies an angle representing the length of the RadialGauge's arc.
- Set the AutoScale property to True. The AutoScale property automatically scales the radial gauge to fill its containing element.

HTML Helpers

Razor copyCode

```
@(Html.C1().RadialGauge()
    .Value(35)
    .Width(500)
    .Height(200)
    .Min(0).Max(100)
    .StartAngle(-20)
    .SweepAngle(220)
    .ShowText(ShowText.None)
    .AutoScale(true)
         .Ranges(items => items
            .Add(item => item.Min(0).Max(40).Color(System.Drawing.Color.Red))
            .Add(item =>
item.Min(40).Max(80).Color(System.Drawing.Color.Yellow))
            .Add(item =>
item.Min(80).Max(100).Color(System.Drawing.Color.Green))
               )
 )
```

Tag Helpers

HTML copyCode

```
<c1-radial-gauge min=0 max=100 value=35 start-angle=-20 sweep-angle=220
         show-text=C1.Web.Mvc.ShowText.None width="300px" height="180px" auto-
scale="true">
    <c1-gauge-range min="0" max="40" color="red"></c1-gauge-range>
    <c1-gauge-range min="40" max=80 color="yellow"></c1-gauge-range>
    <c1-gauge-range min="80" max="100" color="green"></c1-gauge-range>
</c1-radial-gauge>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view

> to the home page.

Back to Top

Features

Animation

Gauges come with in-built animations to improve their appearance. The `IsAnimated` property allows you to enable or disable the animation effects. Whereas the `LoadAnimation` and `UpdateAnimation` properties allow you to select the easing, duration and other attributes of the animation.

The following code example demonstrates how to set this property in C#. These example uses the sample created in the LinearGauge Quick Start section.

In Code

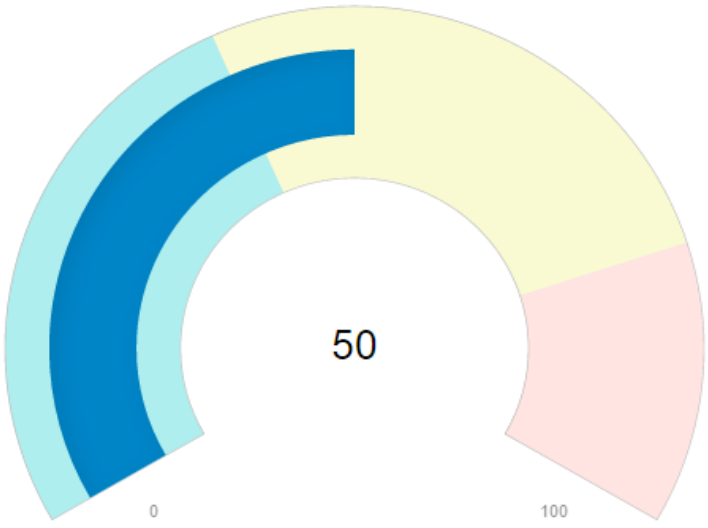
C# copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

Customize Appearance

You can add colors and borders to the gauge and its elements to make it visually attractive. The following code examples demonstrate how to set different styling properties to customize the RadialGauge.

The following image shows how the RadialGauge appears after its appearance is customized.



The following code example demonstrates how to customize the gauge in ASP.NET MVC Edition using Razor. The example uses the sample created in the RadialGauge Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
// Create and Customize Ranges
     .Ranges(items => items
          .Add(item => item.Min(0).Max(40).Color(System.Drawing.Color.Red))
          .Add(item => item.Min(40).Max(80).Color(System.Drawing.Color.Yellow))
          .Add(item => item.Min(80).Max(100).Color(System.Drawing.Color.Green))
          )
```

Tag Helpers

HTML copyCode

```
<c1-radial-gauge min=0 max=100 value=35 start-angle=-20 sweep-angle=220
         show-text=C1.Web.Mvc.ShowText.None width="300px" height="180px" auto-
scale="true">
    <c1-gauge-range min="0" max="40" color="red"></c1-gauge-range>
    <c1-gauge-range min="40" max=80 color="yellow"></c1-gauge-range>
    <c1-gauge-range min="80" max="100" color="green"></c1-gauge-range>
</c1-radial-gauge>
```

Direction

The GaugeDirection property of LinearGauge and BulletGraph allows you to change the direction in which the pointer moves.

There are four available options for the GaugeDirection property:

- **Up**: The Pointer starts from the bottom of the gauge and moves towards the top.
- **Right**: The Pointer starts from the left of the gauge and moves towards the right. This is the **default** value.
- **Down**: The Pointer starts from the top of the gauge and moves towards the bottom.
- **Left**: The pointer starts from the right of the gauge and moves towards the left.

The GaugeDirection property changes the direction of pointer, not orientation of the gauge, therefore its height and width must be set accordingly.

The following image shows how the LinearGauge appears after the GaugeDirection property has been set to Down, and height and width has been adjusted accordingly.

The following code examples demonstrate how to set this property in ASP.NET MVC Edition. These examples use the sample created in the LinearGauge Quick Start section.

HTML Helpers

Razor copyCode

```
.Direction(C1.Web.Mvc.GaugeDirection.Down)
.Height(400)
.Width(120)
```

Tag Helpers

HTML copyCode

```
<c1-linear-gauge width="500px" height="100px" value="35"
direction="C1.Web.Mvc.GaugeDirection.Down">
</c1-linear-gauge>
```

Displaying Values

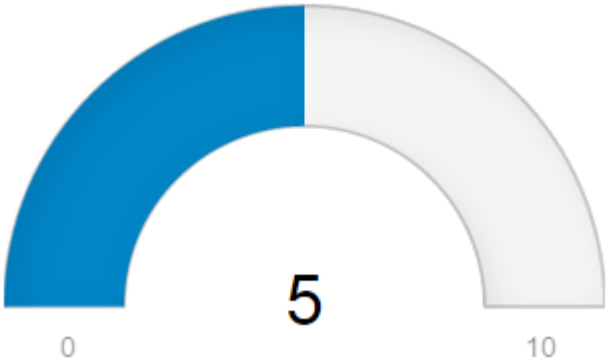
You can display values on a gauge to know the minimum/maximum value or current value of the gauge. The ShowText property allows you to select the values that you want to display as text by the gauge.

There are four valid values for the `ShowText` property:

- **Value**: Displays only the current value of gauge.

- **MinMax**: Displays only the min and max values of gauge.
- **All**: Displays the min, max, and current values of the gauge.
- **None**: Displays no text in the gauge control. This is the **default** value.

The following image shows how the RadialGauge appears after the `ShowText` property has been set to All.



The following code example demonstrates how to set this property in ASP.NET MVC Edition using Razor. The example uses the sample created in the RadialGauge Quick Start section.

In Code

HTML Helpers

Razor copyCode

```
@(Html.C1().RadialGauge()
    .Min(0).Max(10).Value(5)
    .Width(300).Height(180)
.ShowText(ShowText.All)
)
```

Tag Helpers

HTML copyCode

```
<c1-radial-gauge min=0 max=10 value=5
 show-text=C1.Web.Mvc.ShowText.All width="300px" height="180px">
</c1-radial-gauge>
```

Range

You can add multiple ranges to a single gauge. Each range denotes a region or a state which can help the user identify the state of the gauge's value. Every range has its `Min` and `Max` properties that specify the range's position on the gauge, as well as `Color` and `Thickness` properties that define the range's appearance.

The following code examples demonstrate how to add ranges to a gauge and set their properties.

In Code

The following code examples demonstrate how to create new instances of type `Ranges`, set their properties and add the newly created ranges to the LinearGauge (or RadialGauge/BulletGraph).

These examples use the sample created in the LinearGauge Quick Start section.

HTML Helpers

Razor copyCode

```
// Create and Customize Ranges
     .Ranges(items => items
          .Add(item => item.Min(0).Max(40).Color(System.Drawing.Color.Red))
          .Add(item => item.Min(40).Max(80).Color(System.Drawing.Color.Yellow))
          .Add(item => item.Min(80).Max(100).Color(System.Drawing.Color.Green))
          )
```

Tag Helpers

HTML copyCode

```
<c1-linear-gauge width="500px" height="100px" value="35"
                 thickness="0.1" min="0" max="100"
direction="C1.Web.Mvc.GaugeDirection.Right">
    <c1-gauge-range min="0" max="40" color="red"></c1-gauge-range>
    <c1-gauge-range min="40" max=80 color="yellow"></c1-gauge-range>
    <c1-gauge-range min="80" max="100" color="green"></c1-gauge-range>
</c1-linear-gauge>
```

Data Binding

While working in XAML, you can fetch the values for `Min`, `Max` and `Value` from an external data source, instead of directly setting the properties.

The following class can serve as a datasource for the gauge.

C#

```
    class GaugeData
    {
        double _value;
        double _min;
        double _max;


        public double Value
        {
            get { return _value; }
            set
            {
                _value = value;

            }
```

```
        }
        public double Min
        {
            get { return _min; }
            set
            {
                _min = value;
            }
        }
        public double Max
        {
            get { return _max; }
            set
            {
                _max = value;
            }
        }
    }
```

In XAML

Initialize the control in XAML, as shown below.

XAML	copyCode

```
Type your example code here. It will be automatically colorized when you switch to
Preview or build the help system.
```

In Code

To databind a Gauge, open the C# code behind of the XAML page and set the `BindingContext` for the gauge, inside the class constructor, as shown below.

C#

```
public DataBinding()
{
    InitializeComponent();
    BindingContext = new GaugeData() { Value = 25, Max=100, Min=0 };
}
```

Gauge ASP.NET Core Tags

Gauge control supports the following ASP.NET Core Tags:

LinearGauge class

- class
- style
- direction
- Face
- format

- has-shadow
- height
- id
- is-animated
- is-read-only
- max
- min
- value-changed
- Origin
- Pointer
- Ranges
- show-ranges
- show-text
- step
- template-bindings
- thickness
- value
- width

Face Range class

- color
- max
- min
- name
- property-changed
- thickness

Pointer Range class

- color
- max
- min
- name
- property-changed
- thickness

Ranges Range class

- color
- max
- min
- name
- property-changed
- thickness

RadialGauge class

- auto-scale
- class
- style
- Face
- format
- has-shadow
- height
- id

- is-animated
- is-read-only
- max
- min
- value-changed
- Origin
- Pointer
- Ranges
- show-ranges
- show-text
- start-angle
- step
- sweep-angle
- template-bindings
- thickness
- value
- width

Face Range class

- color
- max
- min
- name
- property-changed
- thickness

Pointer Range class

- color
- max
- min
- name
- property-changed
- thickness

Ranges Range class

- color
- max
- min
- name
- property-changed
- thickness

BulletGraph class

- bad
- class
- style
- direction
- Face
- format
- good
- has-shadow
- height

- id
- is-animated
- is-read-only
- max
- min
- value-changed
- Origin
- Pointer
- Ranges
- show-ranges
- show-text
- step
- target
- template-bindings
- thickness
- value
- width

Face Range class

- color
- max
- min
- name
- property-changed
- thickness

Pointer Range class

- color
- max
- min
- name
- property-changed
- thickness

Ranges Range class

- color
- max
- min
- name
- property-changed
- thickness

Input

ASP.NET MVC Edition offers various Input controls that collects data from users in the web UI. The controls let you create numeric, masked, percentage, date, currency and text inputs, depending on the type of data to be collected. These Input controls are highly customizable, and allow developers to create data-sensitive applications.

The table below lists **Input** controls supported by ASP.NET MVC Edition:

Control	Description

AutoComplete	Select values from lists retrieved dynamically from the server. The AutoComplete control retrieves options from the server as the user types.
Calendar	Select a date from a month calendar, navigating through days, months, and years. Use the Min and Max properties to restrict the selectable dates.
ColorPicker	Select a color by clicking the panels to adjust color channels (hue, saturation, brightness, alpha).
ComboBox	Select values by picking them from a list with auto-completion. Use the displayMemberPath and displayValuePath properties to select from tables with complex objects (e.g. key/value pairs).
InputColor	Select colors by typing in HTML-supported color strings, or from a drop-down that shows a ColorPicker control. The Value property of the control gets or sets the currently selected color.
InputDate	Edit date values or pick dates from a drop-down calendar. Use the min and max properties to restrict the valid date range.
InputMask	Edit strings using a mask that prevents invalid input and skips over literals. Use the mask property to specify the format of the input.
InputNumber	Edit numeric values or use spinner buttons to increment or decrement the current value. Use the Min and Max properties to restrict the valid numeric range.
InputTime	Edit time values or pick times from a drop-down list. Use the min and max properties to restrict the valid time range.
ListBox	Display a list of values containing plain text or HTML, and allow users to select a value. This control is displayed in the drop-down part of the ComboBox, AutoComplete, TimePicker, and Menu controls.
Menu	Shows a text element with a drop-down list of commands that the user can invoke by click.
Popup	Display arbitrary content as popovers and dialogs in your application, and seek desired information from end-user through these.
Multi-select	Display a list of custom objects or strings in the drop-down, and allow users to select multiple items from the list.
MultiAutoComplete	Select values from lists retrieved dynamically from the server. It allows user to pick items from lists that contain either custom objects or simple strings.

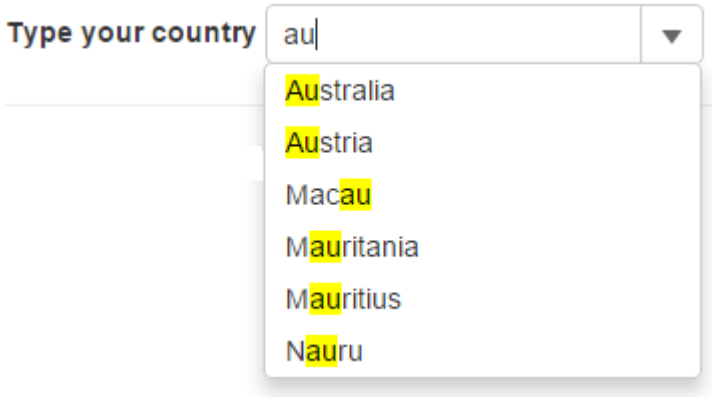
Controls

AutoComplete

The **AutoComplete** control allows you to filter item list as you type, and select a value directly from its drop-down list.

The control makes it easier for a user to search and filter a specific data field(s) from a pre-populated list.

The AutoComplete control below uses a string array as its itemsSource. For example, you type 'au' and it gives countries in the search results that have 'au' string in their names.



Key Features

- CSSMatch: Use the CSSMatch property to highlight parts of the content that match the search items as you type.
- ItemsSourceFunction: Provides the search items dynamically as the user types.
- MaxItems: Set its value to define the maximum number of items to be displayed in the dropdown.
- MinLength: Set its value to set the minimum input length required to trigger autocomplete suggestion
- Delay: Set its value to provide time delay (milliseconds) between a keystroke and the search performed.

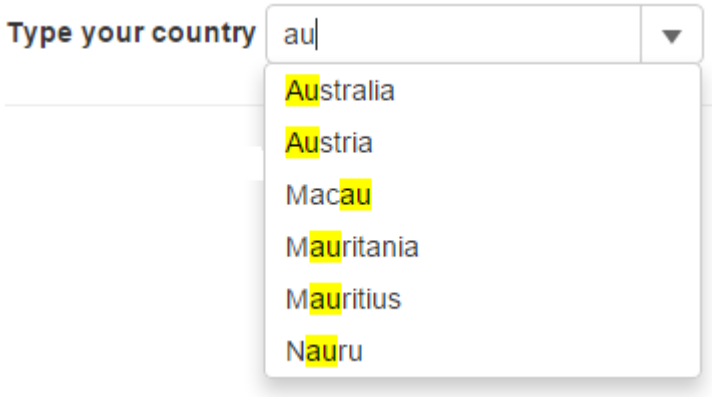
Quick Start

This section describes how to add an **AutoComplete** control to your MVC web application and add data to it. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for AutoComplete**
- **Step 3: Add an AutoComplete control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about

creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for AutoComplete

1. Add a new class to the folder **Models** (for example: `Countries.cs`).
2. Add the following code to the new model to define the classes that serve as a datasource for the AutoComplete control.

C#

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

namespace MVCFlexGrid.Models
{
    public class Countries
    {
        public static List<string> GetCountries()
        {
            return new List<string>
            {
                "Afghanistan", "Albania", "Algeria", "American Samoa",
"Andorra", "Angola", "Anguilla", "Antigua", "Argentina", "Armenia",
        "Aruba", "Australia", "Austria", "Azerbaijan", "Bahamas", "Bahrain",
"Bangladesh", "Barbados", "Belarus", "Belgium", "Belize",
        "Benin", "Bermuda", "Bhutan", "Bolivia", "Bonaire", "Bosnia",
"Botswana", "Brazil", "Brunei", "Bulgaria", "Burkina Faso", "Burundi",
        "Cambodia", "Cameroon", "Canada", "Canary Islands", "Cape Verde",
"Cayman Islands", "Central African Republic", "Chad", "Channel Islands",
        "Chile", "China", "Christmas Island", "Cocos Island", "Colombia",
"Comoros", "Congo", "Cook Islands", "Costa Rica", "Cote D'Ivoire",
        "Croatia", "Cuba", "Curacao", "Cyprus", "Czech Republic", "Denmark",
"Djibouti", "Dominica", "Dominican Republic", "East Timor", "Ecuador",
        "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea", "Estonia",
"Ethiopia", "Falkland Islands", "Faroe Islands", "Fiji", "Finland",
        "France", "French Guiana", "French Polynesia", "French Southern Ter",
"Gabon", "Gambia", "Georgia", "Germany", "Ghana", "Gibraltar",
        "Great Britain", "Greece", "Greenland", "Grenada", "Guadeloupe", "Guam",
"Guatemala", "Guinea", "Guyana", "Haiti", "Honduras",
        "Hong Kong", "Hungary", "Iceland", "India", "Indonesia", "Iran", "Iraq",
"Ireland", "Isle of Man", "Israel", "Italy", "Jamaica", "Japan",
        "Jordan", "Kazakhstan", "Kenya", "Kiribati", "Korea North", "Korea
South", "Kuwait", "Kyrgyzstan", "Laos", "Latvia", "Lebanon", "Lesotho",
        "Liberia", "Libya", "Liechtenstein", "Lithuania", "Luxembourg", "Macau",
"Macedonia", "Madagascar", "Malaysia", "Malawi", "Maldives",
        "Mali", "Malta", "Marshall Islands", "Martinique", "Mauritania",
"Mauritius", "Mayotte", "Mexico", "Midway Islands", "Moldova", "Monaco",
        "Mongolia", "Montserrat", "Morocco", "Mozambique", "Myanmar", "Nambia",
"Nauru", "Nepal", "Netherland Antilles", "Netherlands", "Nevis",
        "New Caledonia", "New Zealand", "Nicaragua", "Niger", "Nigeria", "Niue",
"Norfolk Island", "Norway", "Oman", "Pakistan", "Palau Island",
```

```
        "Palestine", "Panama", "Papua New Guinea", "Paraguay", "Peru",
"Philippines", "Pitcairn Island", "Poland", "Portugal", "Puerto Rico",
        "Qatar", "Republic of Montenegro", "Republic of Serbia", "Reunion",
"Romania", "Russia", "Rwanda", "St Barthelemy", "St Eustatius",
        "St Helena", "St Kitts-Nevis", "St Lucia", "St Maarten", "Saipan",
"Samoa", "Samoa American", "San Marino", "Saudi Arabia", "Scotland",
        "Senegal", "Serbia", "Seychelles", "Sierra Leone", "Singapore",
"Slovakia", "Slovenia", "Solomon Islands", "Somalia", "South Africa",
        "Spain", "Sri Lanka", "Sudan", "Suriname", "Swaziland", "Sweden",
"Switzerland", "Syria", "Tahiti", "Taiwan", "Tajikistan", "Tanzania",
        "Thailand", "Togo", "Tokelau", "Tonga", "Trinidad Tobago", "Tunisia",
"Turkey", "Turkmenistan", "Turks & Caicos Is", "Tuvalu", "Uganda",
        "Ukraine", "United Arab Emirates", "United Kingdom", "United States of
America", "Uruguay", "Uzbekistan", "Vanuatu", "Vatican City State",
        "Venezuela", "Vietnam", "Virgin Islands (British)", "Virgin Islands
(USA)", "Wake Island", "Yemen", "Zaire", "Zambia", "Zimbabwe"
            };
        }
    }
}
```

Back to Top

Step 3: Add an AutoComplete control

Complete the following steps to initialize an AutoComplete control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Include the MVC references as shown below.

```
C#
using C1.Web.Mvc.Serializition;
```

5. Replace the method `Index()` with the following method.

```
C#                                                                    copyCode
public ActionResult Index()
{
    ViewBag.Countries = MVCFlexGrid.Models.Countries.GetCountries();
    return View();
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller `QuickStartController` to open it.
2. Place the cursor inside the method `QuickStart()`.
3. Right click and select **Add View**. The **Add View** dialog appears.

4. In the **Add View** dialog, verify that the view name is **QuickStart** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.
6. Instantiate a FlexChart control in the view QuickStart as shown below.

HTML Helpers

Razor copyCode

```
@{
    List<string> countries = ViewBag.Countries;
}
<style>
    .highlight {
        background-color: #ff0;
        color: #000;
    }
</style>
<div>
    <label>Type your country</label>
    @(Html.C1().AutoComplete().Bind(countries).CssMatch("highlight")
    .Delay(2000)
    )
</div>
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@{
    List<string> countries = ViewBag.Countries;
}

<style>
    .highlight {
        background-color: #ff0;
        color: #000;
    }
</style>

<div>
    <label>Type your country</label>
    <c1-auto-complete css-match="highlight">
        <c1-items-source source-collection="@countries">
        </c1-items-source>
    </c1-auto-complete>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**QuickStart/QuickStart**) in the address bar of the browser to see the view.

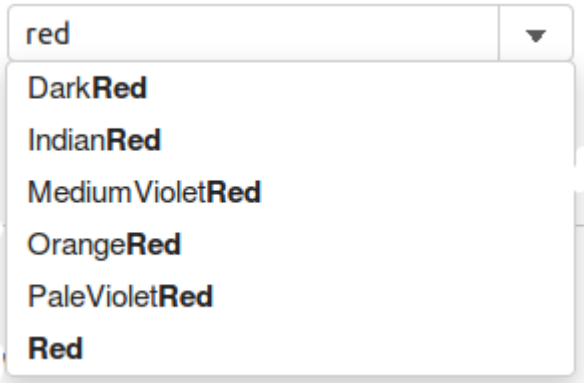
Back to Top

Features

AutoComplete with custom data source

You can bind AutoComplete control to a complex type list using the **SelectedValuePath** and **DisplayMemberPath** properties. Here, we are binding the AutoComplete control to SystemColors class of Namespace, System.Drawing. The control will provide the list of all the members available in this class.

The following image shows how the AutoComplete appears after setting the **SelectedValuePath** and **DisplayMemberPath** properties.



The following code example demonstrates how to enable complex type binding in AutoComplete:

Custom Datasource for AutoComplete

1. Add a new class to the folder **Models** (for example: `NamedColor.cs`).
2. Add the following code to the new model to define the classes that serve as a datasource for the AutoComplete control.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;namespace MVCFlexGrid.Models
{

    public class NamedColor
    {
        public string Name { get; set; }
        public string Value { get; set; }
    }
}
```

Back to Top

In Code

AutoCompleteController.cs

Razor copyCode

```
public ActionResult ComplexType()
{
    var list = GetSystemColors();
    return View(list);
}
private static NamedColor[] GetSystemColors()
{
    return Enum.GetValues(typeof(KnownColor))
        .Cast<KnownColor>()
        .Select(c => new NamedColor
        {
            Name = c.ToString(),
            Value = "#" +
Color.FromKnownColor(c).ToArgb().ToString("X8").Substring(2)
        })
        .ToArray();
}
```

AutoComplete.cshtml

HTML Helpers

Razor copyCode

```
@model IEnumerable<MVCFlexGrid.Models.NamedColor>
<div>
    <label>Complex Type List</label>
    @(Html.C1().AutoComplete()
        .Bind(Model)
        .DisplayMemberPath("Name")
        .SelectedValuePath("Value")
    )
</div>
```

Tag Helpers

HTML copyCode

```
@model IEnumerable<TagFlexGrid.Models.NamedColor>

<div>
    <label>Complex Type List</label>

    <c1-auto-complete display-member-path="Name" selected-value-path="Value" items-
source-action="@Url.Action("TypesInMscorlib")">
```

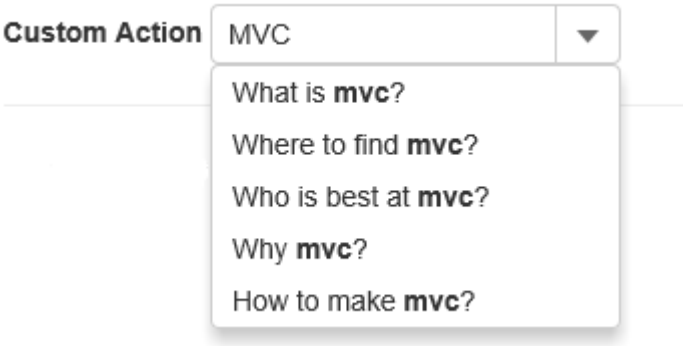
```
        <c1-items-source source-collection="Model"></c1-items-source>
    </c1-auto-complete>
</div>
```

Back to Top

Custom Action

Custom action is an another important feature, when you are working with **AutoComplete** in your MVC application. You can specify custom actions, that you want to perform using AutoComplete control. For example, in the code below we are using custom actions to filter data after certain number of characters are typed by a user.

The following image shows how the AutoComplete appears after using custom actions.



The following code example demonstrates how to use custom action in AutoComplete:

In Code

AutoCompleteController.cs

Razor

```
public ActionResult Index()
        {
            return View();
        }
        public ActionResult Heuristic(string query, int max)
        {
            var prefix = new[] { "What is ", "Where to find ", "Who is best at ",
"Why ", "How to make " };
            return this.C1Json(prefix.Select(f => f + query + "?").ToList(),
                        behavior: JsonRequestBehavior.AllowGet);
        }
```

AutoComplete.cshtml

HTML Helpers

Razor

```
<div>
    <label>Custom action</label>
    @(Html.C1().AutoComplete()
        .ItemsSourceAction(Url.Action("Heuristic"))
```

```
    )
</div>
```

Tag Helpers

Razor

```
<div>
    <label>Custom action</label>
    <c1-auto-complete items-source-action="@Url.Action("Heuristic")">
    </c1-auto-complete>
</div>
```

AutoComplete ASP.NET Core Tags

AutoComplete control supports the following ASP.NET Core Tags:

AutoComplete class

- class
- css-match
- style
- delay
- display-member-path
- height
- id
- is-content-html
- is-dropped-down
- is-editable
- item-formatter
- c1-items-source
- items-source-action
- items-source-function
- max-drop-down-height
- max-drop-down-width
- max-items
- min-length
- is-dropped-down-changed
- selected-index-changed
- text-changed
- placeholder
- required
- selected-index
- selected-item
- selected-value
- selected-value-path
- show-drop-down-button
- template-bindings
- text
- width

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- on-client-collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

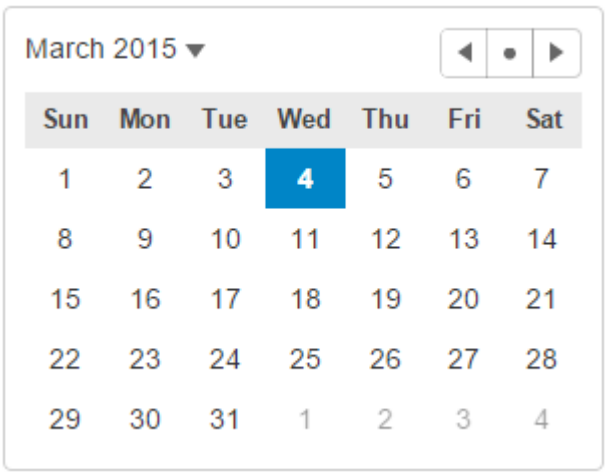
- client-converter
- property-name

SortDescriptions

- ascending
- property

Calendar

The Calendar control displays a one-month calendar and allows users to select a date. You can navigate through days, months and years. Use the Min and Max properties to restrict the selectable dates. The Calendar control is displayed in the drop-down part of the DatePicker control.



Key Features:

- FirstDayofWeek : Set its value to a Day that you want to display in the first column of the calendar.

- Value: Provides the currently selected date by the user.
- Max: Set its value to restrict range of the latest date that a user can select in the calendar.
- Min: Set its value to restrict range of the earliest date that a user can select in the calendar.
- MonthView: Indicates whether the calendar displays a month or a year.

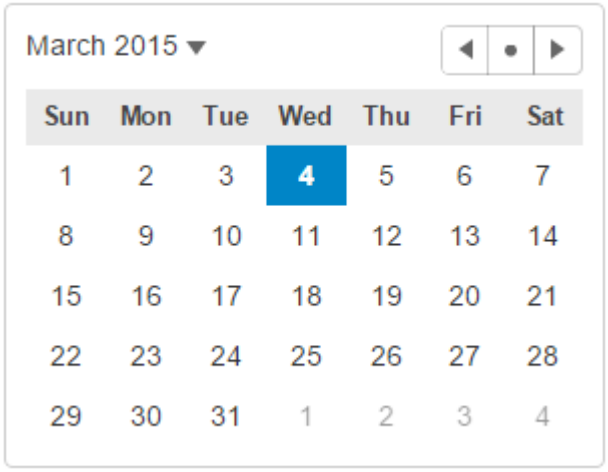
Quick Start

This topic demonstrates how to add a Calendar control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Calendar Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.

2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **Calendar** control.

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@{
    //create variables to set min, max and current values
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}
<div>
    //initialize a calendar control
    @(Html.C1().Calendar()
    //customize the control
        .Value(today).Min(minDate).Max(maxDate).MonthView(true)
        .Width(300)

    )

</div>
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<div>
    <c1-calendar  value="@today" min="@minDate" max="maxDate" month-view=true
width="300px">
    </c1-calendar>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

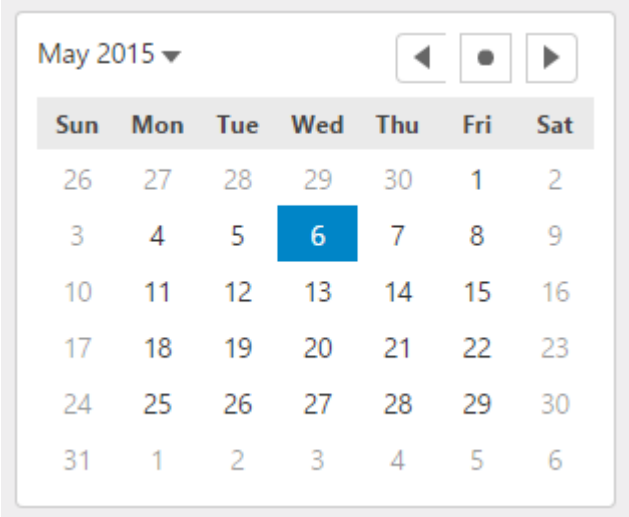
Features

Date Validation

ComponentOne MVC Edition's Calendar control has a built-in validator, **ItemValidator** that lets you control the type of data or the value that users can select from the calendar control. All you need to do is create a validation function, which is assigned to ItemValidator to allow or restrict the selection of dates by the user.

In this example, ItemValidator function is created so that a user can select a weekday, but not Saturdays and Sundays in the calendar control. This kind of validation is useful while planning your work sprints, workout days or diet plan.

The following image shows how the **Calendar** control appears after applying ItemValidator to it.



The following code example demonstrates how to add ItemValidator in the Calendar control:

In Code

Add a new Controller:

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:

    2. Select **Empty MVC Controller** template.
    3. Set name of the controller (for example: `Default1Controller`).
    4. Click **Add**.
4. A new controller is added to the application.

Calendar.cshtml

HTML Helpers

Razor copyCode

```
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<script>
    function itemValidator(date) {
        var weekday = date.getDay();
        return weekday != 0 && weekday != 6;
    }
</script>

<div>
    @(Html.C1().Calendar().Value(today).Min(minDate).Max(maxDate).Width(300)
    .ItemValidator("itemValidator"))
</div>
```

Tag Helpers

HTML

```
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}
<script>
    function itemValidator(date) {
        var weekday = date.getDay();
        return weekday != 0 && weekday != 6;
    }
</script>
<div>
    <c1-calendar value="@today" min="@minDate" max="@maxDate" item-
validator="itemValidator" width="300px">
    </c1-calendar>
</div>;
```

Calendar ASP.NET Core Tags

Calendar control supports the following ASP.NET Core Tags:

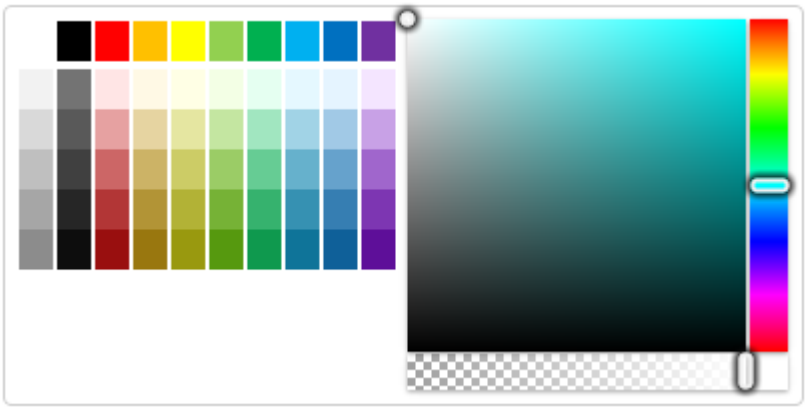
Calendar class

- class

- style
- display-month
- first-day-of-week
- height
- id
- item-formatter
- max
- min
- month-view
- display-month-changed
- value-changed
- show-header
- template-bindings
- value
- width

ColorPicker

The ColorPicker control lets users select a color by clicking on panels to adjust channels, such as hue, saturation, brightness and alpha. The control is used as a drop-down for the InputColor control.



Key Features:

- Value: Represents the currently selected color by the user.
- ShowAlphaChannel: Allows users to edit the color's alpha channel (transparency). By default,its value is true.
- ShowColorString: Shows a string representation of the current color.
- Palette: Represents an array that contains the colors in the palette. The Palette contains ten colors, represented by an array with ten strings.

Quick Start

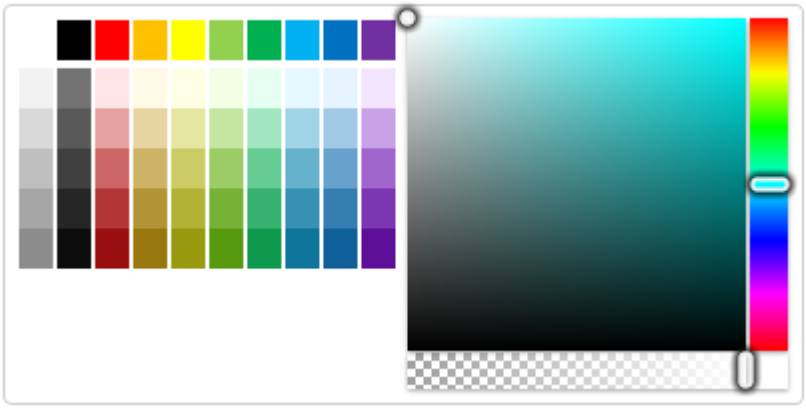
This topic demonstrates how to add a **ColorPicker** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the ColorPicker Control**

- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **ColorPicker** control.

HTML Helpers

Razor	copyCode

```
@using System.Drawing
@using MVCFlexGrid.Models
<div>
@(Html.C1().ColorPicker()
    .Width(400)
)
</div>
```

Tag Helpers

HTML copyCode

```
@using TagFlexGrid.Models
@using System.Drawing
<div>
    <label>ColorPicker</label>
    <c1-color-picker width="400px">
    </c1-color-picker>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

ColorPicker ASP.NET Core Tags

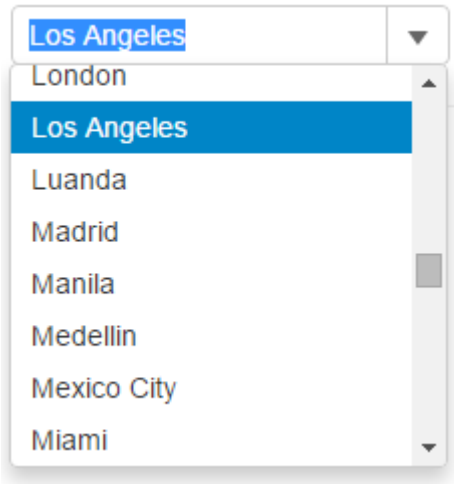
ColorPicker control supports the following ASP.NET Core Tags:

ColorPicker class

- class
- style
- height
- id
- value-changed
- palette
- show-alpha-channel
- show-color-string
- template-bindings
- value
- width

ComboBox

The **ComboBox** control lets a user pick items from lists. The control automatically complete entries as the user types, and allows users to show a drop-down list with the items available. You can populate the items list by binding the ComboBox control to a model.



Key Features

- SelectedItem: Represents the currently selected item in the drop-down list.
- SelectedIndex : Represents the index of the SelectedItem in the drop-down list.
- SelectedValue: Represents the value of the SelectedItem, obtained using the SelectedValuePath.
- SelectedValuePath: Represents the name of the property used to get the SelectedValue and SelectedItem.
- MaxDropDownHeight: Set the value of the MaxDropDownHeight property to set the maximum height of the drop-down list.
- MaxDropDownWidth: Set the value of the MaxDropDownWidth property to set the maximum width of the drop-down list.
- IsEditable: Determines whether a user can enter values that do not appear on the ComboBox's item list.

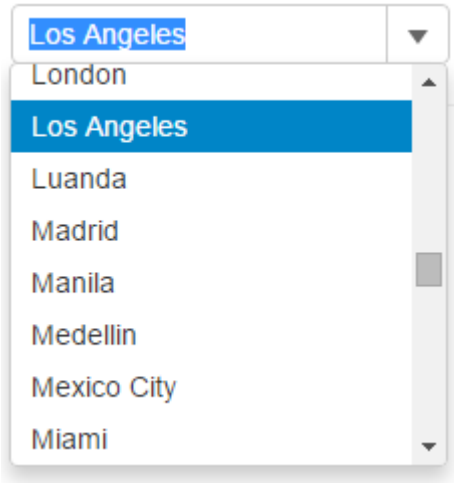
Quick Start

This section describes how to add a **ComboBox** control to your MVC web application and add data to it. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for ComboBox**
- **Step 3: Add a ComboBox control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for ComboBox

1. Add a new class to the folder **Models** (for example: `Cities.cs`).
2. Add the following code to the new model to define the classes that serve as a datasource for the ComboBox control.

C#

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

namespace MVCFlexGrid.Models
{
    public class Cities
    {
        public static List<string> GetCities()
        {
            return new List<string>
            {
                "Abidjan", "Accra", "Ahmedabad", "Alexandria", "Ankara",
"Atlanta", "Baghdad", "Bandung", "Bangkok", "Barcelona", "Beijing", "Belo
Horizonte",
        "Bengaluru", "Bogota", "Boston", "Buenos Aires", "Cairo", "Calcutta",
"Chengdu", "Chennai", "Chicago", "Chongqung", "Dalian", "Dallas", "Delhi",
        "Detroit", "Dhaka", "Dongguan", "Essen", "Fuzhou", "Guadalajara",
"Guangzhou", "Hangzhou", "Harbin", "Ho Chi Minh City", "Hong Kong", "Houston",
        "Hyderabad", "Istanbul", "Jakarta", "Johannesburg", "Karachi",
"Khartoum", "Kinshasa", "Kuala Lumpur", "Lagos", "Lahore", "Lima", "London",
        "Los Angeles", "Luanda", "Madrid", "Manila", "Medellin", "Mexico City",
"Miami", "Milan", "Monterrey", "Moscow", "Mumbai", "Nagoya", "Nanjing",
        "Naples", "New York", "Osaka", "Paris", "Pheonix", "Philadelphia",
"Porto Alegre", "Pune", "Qingdao", "Quanzhou", "Recife", "Rio de Janeiro",
```

```
        "Riyadh", "Rome", "Saint Petersburg", "Salvador", "San Francisco",
"Santiago", "Sao Paulo", "Seoul", "Shanghair", "Shenyang", "Shenzhen",
        "Singapore", "Surabaya", "Surat", "Suzhou", "Sydney", "Taipei",
"Tehran", "Tianjin", "Toronto", "Washington", "Wuhan", "Xi'an-Xianyang",
"Yangoon",
        "Zhengzhou", "Tokyo"
            };
        }
    }
}
```

Back to Top

Step 3: Add a ComboBox control

Complete the following steps to initialize an AutoComplete control.

Add a new Controller

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: `Default1Controller`).
    3. Click **Add**.
4. Replace the method `Index()` with the following method.

C# copyCode

```
ViewBag.Cities = MVCFlexGrid.Models.Cities.GetCities();
return View();
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller `QuickStartController` to open it.
2. Place the cursor inside the method `QuickStart()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **QuickStart** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.
6. Instantiate a FlexChart control in the view QuickStart as shown below.

HTML Helpers

Razor copyCode

```
@{
    List<string> cities = ViewBag.Cities;
}
<div>
    @(Html.C1().ComboBox().Bind(cities).SelectedIndex(0))
</div>
```

Tag Helpers

HTML copyCode

```
@{
    List<string> cities = ViewBag.Cities;
}

<div>

    <c1-combo-box selected-index=0>
        <c1-items-source source-collection=@cities></c1-items-source>
    </c1-combo-box>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**QuickStart/QuickStart**) in the address bar of the browser to see the view.

Back to Top

ComboBox ASP.NET Core Tags

ComboBox control supports the following ASP.NET Core Tags:

ComboBox class

- class
- style
- display-member-path
- height
- id
- is-content-html
- is-dropped-down
- is-editable
- item-formatter
- c1-items-source
- max-drop-down-height
- max-drop-down-width
- is-dropped-down-changed
- selected-index-changed
- text-changed
- placeholder
- required
- selected-index
- selected-item
- selected-value
- selected-value-path
- show-drop-down-button
- template-bindings

- text
- width

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- on-client-collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

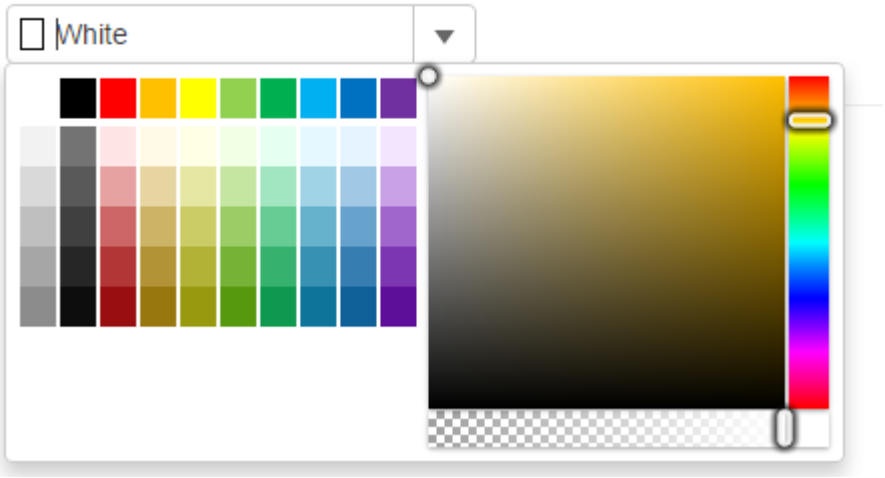
- client-converter
- property-name

SortDescriptions

- ascending
- property

InputColor

The InputColor control lets a user select colors by typing in HTML-supported color strings, and to pick colors from a drop-down list that shows a ColorPicker control. The Value property of the ColorPicker control gets or sets the currently selected color.



Key Features:

- Value: Represents the currently selected color by the user.
- ShowAlphaChannel: Allows users to edit the color's alpha channel (transparency). By default,its value is true.
- Required: Indicates whether the control value must be a color or whether it can be set to null.

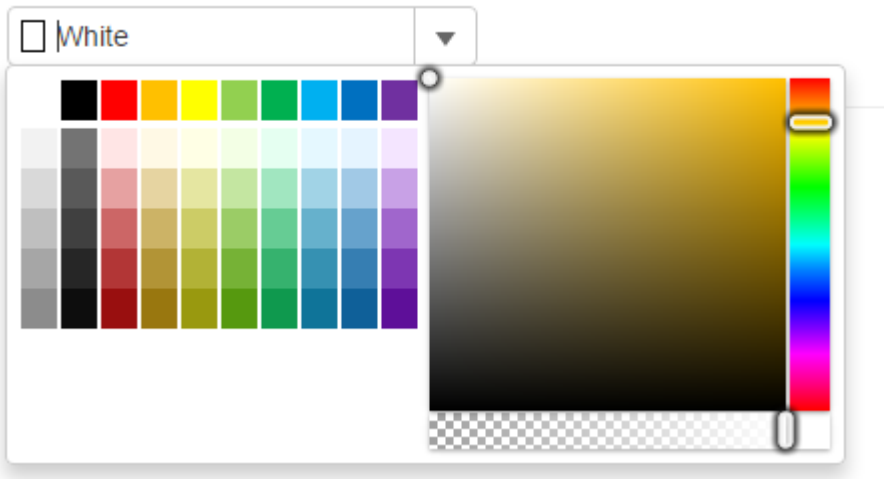
Quick Start

This topic demonstrates how to add an **InputColor** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the inputColor Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **InputColor** control.

HTML Helpers

Razor copyCode

```
@using System.Drawing
<script>
    function changeColor(sender, e) {
        document.getElementById("image1").style.backgroundColor = sender.value;
    }
</script>
// add an image and set its height and width
<p>
    <img id="image1" height="300" width="400"
src="@Href("~/Content/images/componentone_logo.png")" />
</p>
<div>
//set the background color of the image

@Html.C1().InputColor().Value(Color.White).OnClientValueChanged("changeColor")
</div>
```

Tag Helpers

HTML copyCode

```
@using System.Drawing

<p>
    <img id="image1" src="@Href("~/Content/images/room.png")" />
</p>

<script>
    function changeColor(sender, e) {
        document.getElementById("image1").style.backgroundColor = sender.value;
    }
</script>

<div>
    <label>Select a color</label>
    <c1-input-color value="White" on-client-value-changed="changeColor"></c1-
input-color>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

InputColor ASP.NET Core Tags

InputColor control supports the following ASP.NET Core Tags:

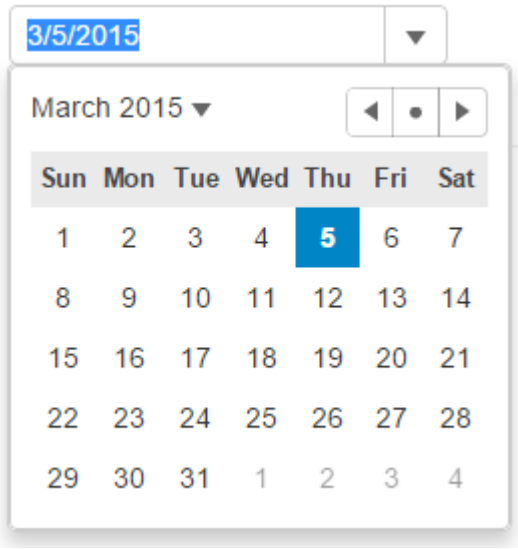
InputColor class

- class
- style
- height
- id
- is-dropped-down
- name
- is-dropped-down-changed
- text-changed
- value-changed
- placeholder
- required
- show-alpha-channel
- show-drop-down-button

- template-bindings
- text
- value
- width

InputDate

The **InputDate** control lets a user type in dates using any of the format supported by the Globalize class, and also pick dates from a drop-down box that shows a calendar control. To restrict the range of values a user can enter, you can use Min and Max properties.



Key Features

- Value: Represents the current date selected by the user.
- Max: Use the Max property to set value of the latest date that the user can enter.
- Min: Use the Min property to set value of the earliest date that the user can enter.
- Format: Use the Format property to represent dates in different formats.
- Required: Indicates whether the control must be a Date or whether it can be set to null.
- DateSelectionMode: Use this property to get or set a value to indicate where users can select days, months, or no values.

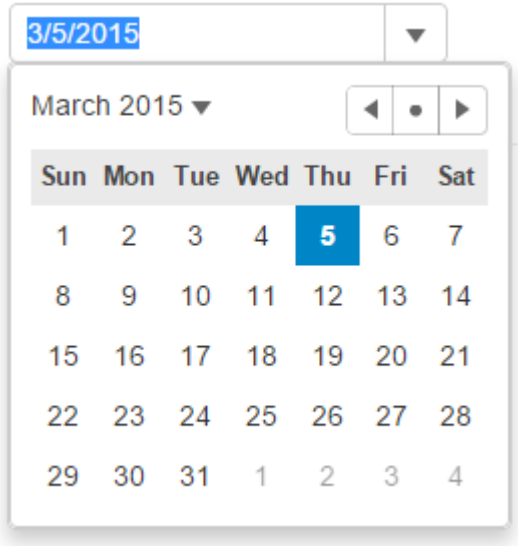
Quick Start

This topic demonstrates how to add an **InputDate** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the InputDate Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **InputDate** control.

HTML Helpers

Razor	copyCode
`@{`	

```
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<div>
    @(Html.C1().InputDate().Id("idcInputDate")

.Value(today).Min(minDate).Max(maxDate).OnClientValueChanged("changeDate")
    )
</div>
```

Tag Helpers

HTML copyCode

```
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<div>
    <label>Select a date</label>
    <c1-input-date id="idcInputDate" value="@today"
    min="@minDate" max="@maxDate"
    on-client-value-changed="changeDate">
    </c1-input-date>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

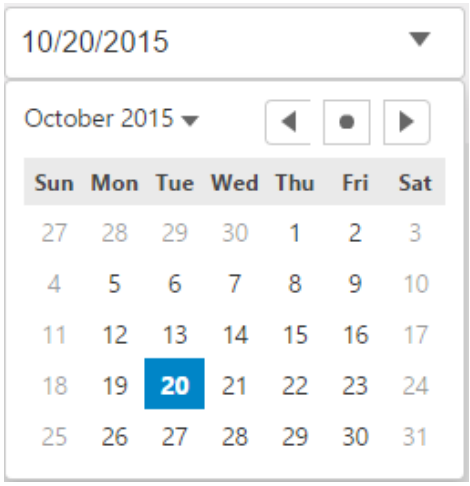
Features

Date Validation

ComponentOne MVC Edition's **InputDate** control has a built-in validator, **ItemValidator** that lets you control the type of data or the value that users select from the InputDate drop down. All you need to do is create a validation function, which is assigned to ItemValidator to allow or restrict the selection of dates by the user.

In this example, ItemValidator is created so that a user can select a weekday, but not Saturdays and Sundays i.e. in the InputDate control. This kind of validation is useful while planning your work sprints, workout days or diet plan.

The following images show how the **InputDate** control appears after applying ItemValidator function to it.



The following code examples demonstrate how to add ItemValidator in the InputDate control:

In Code

Add a new Controller:

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:

    2. Select **Empty MVC Controller** template.
    3. Set name of the controller (for example: `Default1Controller`).
    4. Click **Add**.
4. A new controller is added to the application.

InputDate.cshtml

HTML Helpers

Razor copyCode

```
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<script>
    function itemValidator(date) {
        var weekday = date.getDay();
        return weekday != 0 && weekday != 6;
    }
</script>

<div>


@(Html.C1().InputDate().Value(today).Min(minDate).Max(maxDate).ItemValidator("itemValidator"))
</div>
```

Tag Helpers

HTML

```
@{
    var today = DateTime.Now.Date;
    var minDate = new DateTime(today.Year, 1, 1);
    var maxDate = new DateTime(today.Year, 12, 31);
}

<script>
    function itemValidator(date) {
        var weekday = date.getDay();
        return weekday != 0 && weekday != 6;
    }
</script>

<div>
    <c1-input-date value="@today" min="@minDate" max="@maxDate" item-
validator="itemValidator">
    </c1-input-date>
</div>
```

InputDate ASP.NET Core Tags

InputDate control supports the following ASP.NET Core Tags:

InputDate class

- class
- style
- format
- height
- id
- is-dropped-down
- mask
- max
- min
- name
- is-dropped-down-changed
- text-changed
- value-changed
- placeholder
- required
- show-drop-down-button
- template-bindings
- text
- value
- width

InputMask

The InputMask control provides a way to govern values entered by a user. The control lets a user edit strings using a mask that prevents invalid input and skips over literals as the user types, such as slashes in date. You can use the Mask property to specify the value of the InputMask control.

The mask used to validate the input is defined by the mask property, which may contain one or more of the following special characters:

- **0** Digit.
- **9** Digit or space.
- **#** Digit, sign, or space.
- **L** Letter.
- **l** Letter or space.
- **A** Alphanumeric.
- **a** Alphanumeric or space.
- **.** Localized decimal point.
- **,** Localized thousand separator.
- **:** Localized time separator.
- **/** Localized date separator.
- **$** Localized currency symbol.
- **<** Converts characters that follow to lowercase.
- **>** Converts characters that follow to uppercase.
- **|** Disables case conversion.
- **** Escapes any character, turning it into a literal.

Key Features

- Mask: Set the Mask property to specify the control's mask, which is used to validate the input as the user types.
- Placeholder: Set the Placeholder value, which is shown as a hint when the control is empty.
- Value: Specifies the value for the InputMask control.
- PromptChar: Use the PromptChar property to set a symbol that is used to show input positions in the control.

Quick Start

This topic demonstrates how to add an **InputMask** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the InputMask Control**
- **Step 4: Build and Run the Project**

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

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **InputMask** control.

HTML Helpers

Razor copyCode

```
<div>
    <label>Social Security Number</label>
    @Html.C1().InputMask().Mask("000-00-0000").HtmlAttributes(new { title =
"Mask: 000-00-0000" })
</div>
```

Tag Helpers

HTML copyCode

```
<div>
    <label>Social Security Number</label>
    <c1-input-mask mask="000-00-0000" title="Mask: 000-00-0000"></c1-input-mask>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

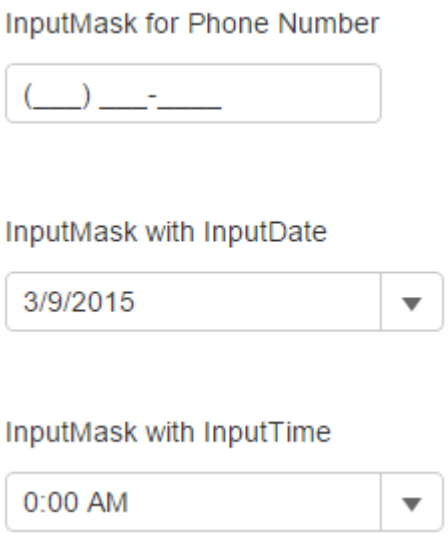
Back to Top

Features

Input Mask Types

You can use InputMask control for verifying different types of data using the Mask property. For example, you can add a mask that prefixes telephone number with three-digit area code. You can also use the control with InputDate and InputTime, which prevents user to enter invalid data accidentally.

The following images show how the **InputMask** control appears after applying different masks to it.



The following code examples demonstrate how to bind Menu to an InputNumber control:

In Code

InputMaskController.cs

Razor copyCode

```
public ActionResult Feature()
{
    return View();
}
```

InputMask.cshtml

HTML Helpers

Razor copyCode

```
<p> InputMask for Phone Number</p>
    <div>
        @(Html.C1().InputMask().Mask("(999) 000-0000")
          .HtmlAttributes(new { title = "Mask: (999) 000-0000" })
          )
    </div>
```

```
<p> InputMask with InputDate</p>
    <div>
        @(Html.C1().InputDate().Value(DateTime.Now)
            .Format("d").Mask("99/99/9999")
            .HtmlAttributes(new { title = "Mask: 99/99/9999" })
        )
    </div>

<p> InputMask with InputTime</p>
    <div>
        @(Html.C1().InputTime().Value(DateTime.Now)
            .Format("t").Mask("00:00 >LL").Step(new TimeSpan(0, 15, 0))
            .HtmlAttributes(new { title = "Mask: 00:00 >LL" })
        )
    </div>
```

Tag Helpers

HTML copyCode

```
<div>
    <label>Input Mask for Phone Number</label>
    <c1-input-mask mask="(999) 000-0000" title="Mask: (999) 000-0000"></c1-input-
mask>
</div>
<div>
    <label>InputMask with InputDate</label>
    <c1-input-date value="@DateTime.Now" format="d" mask="99/99/9999" title="Mask:
99/99/9999"></c1-input-date>
</div>
<div>
    <label>InputMask with InputTime</label>
    <c1-input-time value="@DateTime.Now" format="t" mask="00:00 >LL" step="@(new
TimeSpan(0, 15, 0).TotalMinutes)" is-editable=true title="Mask: 00:00 >LL"></c1-
input-time>
</div>
```

InputMask ASP.NET Core Tags

InputMask control supports the following ASP.NET Core Tags:

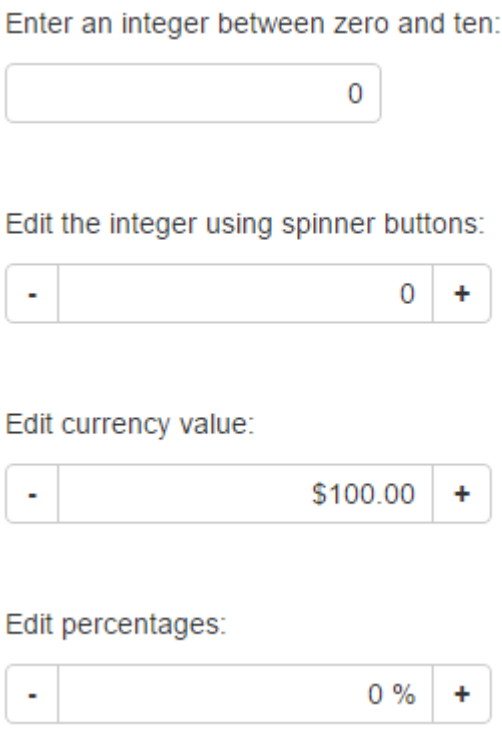
InputMask class

- class
- style
- height
- id
- mask
- name

- value-changed
- placeholder
- prompt-char
- template-bindings
- value
- width

InputNumber

The **InputNumber** control allows users to enter numbers. The control prevents users from entering the invalid data and formats as it is edited. To limit the range of acceptable values, you can use the Min and Max properties.



Key Features

- Value: Represents the current value of the control.
- Max: Use the Max property value to set the largest number that the user can enter.
- Min: Use the Min property value to set the smallest number that the user can enter.
- ShowSpinner: Its value indicates whether the control displays spinner buttons to increment or decrement the value.
- InputType: Use the InputType property to set the type attribute of the HTML input element hosted by the control.

Quick Start

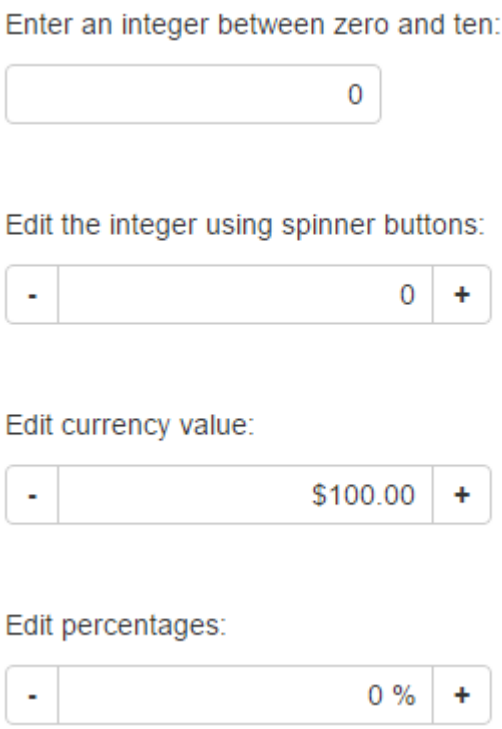
This topic demonstrates how to add an **InputNumber** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**

- **Step 3: Add the InputNumber Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **InputNumber** control.

HTML Helpers

Razor copyCode

```
<div>
    <p> Enter an integer between zero and ten:
    </p>
    @(Html.C1().InputNumber().Value(0).Format("n0")
    .Min(0) .Max(10)
    .Placeholder("integer between zero and ten")
    )
</div>

<div>
    <p> Edit the integer using spinner buttons:</p>
    @(Html.C1().InputNumber().Value(0).Format("n0")
    .Min(0) .Max(10)
    .Step(1)
    .Placeholder("integer between zero and ten")
    )
</div>

<div>
    <p> Edit currency value:</p>
    @(Html.C1().InputNumber()
        .Value(100).Format("c")
        .Step(100).Min(100).Max(1000)
        .Placeholder("amount below $1,000")
    )
</div>

<div>
    <p> Edit percentages:</p>
    @(Html.C1().InputNumber()
        .Placeholder("percentage")
        .Format("p0")
        .Step(.1)
    )
</div>
```

Tag Helpers

HTML copyCode

```
<div>
    <p> Enter an integer between zero and ten:</p>
```

```
    <c1-input-number value=0 format="n0" min="0" max="10" placeholder="Integer
between zero and ten" ></c1-input-number>
</div>

<div>
    <p> Edit the integer using spinner buttons:</p>
    <c1-input-number value=0 format="n0" min="0"
           max="10" placeholder="Integer between zero and ten" step="1"></c1-
input-number>
</div>

<div>
    <p>Edit currency value:</p>
    <c1-input-number value=100 format="c" step=100 min=100 max=1000
placeholder="amount below $1,000"></c1-input-number>
</div>

<div>
    <p>Edit percentages:</p>
    <c1-input-number placeholder="percentage" format="p0" step="0.1"></c1-input-
number>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

InputNumber ASP.NET Core Tags

InputNumber control supports the following ASP.NET Core Tags:

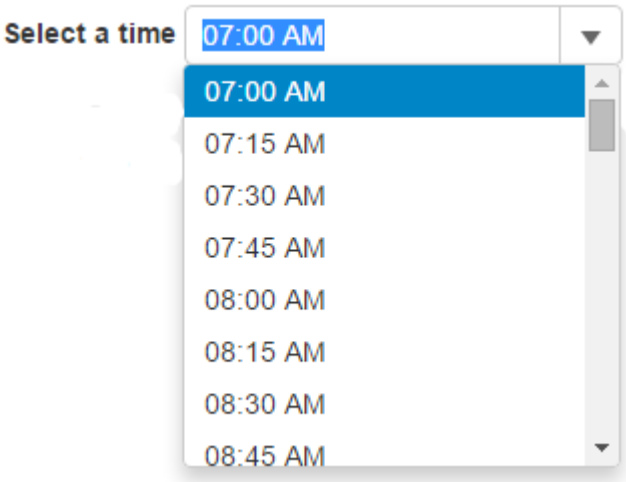
InputNumber class

- class
- style
- format
- height
- id
- input-type
- max
- min
- name
- text-changed
- value-changed
- placeholder

- required
- show-spinner
- step
- template-bindings
- text
- value
- width

InputTime

The InputTime control allows users to enter times using any format supported by the Globalize class, or to pick times from a drop-down list. The Min, Max and Step properties determine the values shown in the list.



Key Features

- Value: Represents the current input time.
- Max: Use the Max property value to set the latest time that a user can enter.
- Min: Use the Min property value to set the earliest time that a user can enter.
- Format: Use the Format property to display time in different formats.
- SelectedItem: Represents the currently selected item in the drop-down list.
- SelectedIndex: Indicates the index of the SelectedItem.
- SelectedValue: Represents the value of the SelectedItem, obtained using the SelectedValuePath.
- Step: Use the Step property to specify the number of minutes between entries in the drop-down list.

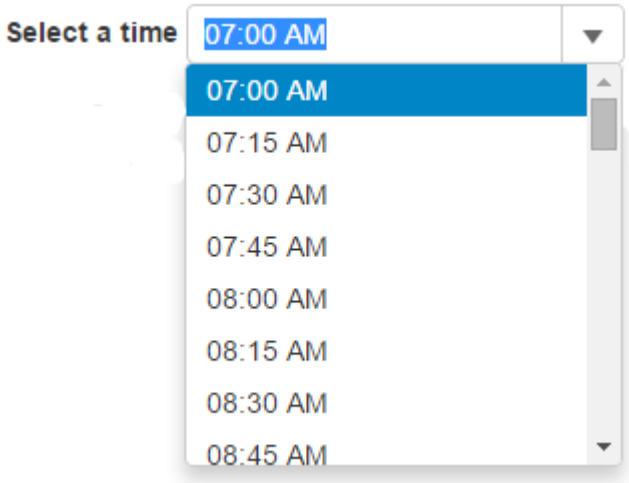
Quick Start

This topic demonstrates how to add an **InputTime** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the InputTime Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **InputTime** control.

HTML Helpers

Razor	copyCode

```
@{
    var minTime = DateTime.Parse("07:00");
    var maxTime = DateTime.Parse("19:00");
}
<div>
    <label>Select a time</label>
    @(Html.C1().InputTime().Id("iditInputTime")
        .Value(DateTime.Now)
        .Min(minTime)
        .Max(maxTime)
        .Step(new TimeSpan(0, 15, 0))
        .Format("hh:mm tt")
    )
</div>
```

Tag Helpers

HTML copyCode

```
@{
    var minTime = DateTime.Parse("07:00");
    var maxTime = DateTime.Parse("19:00");
}

<div>
    <label>Select a time</label>
    <c1-input-time id="iditInputTime" value="@DateTime.Now"
            min="@minTime" max="@maxTime"
            step="@(new TimeSpan(0, 15, 0).TotalMinutes)"></c1-input-time>
</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

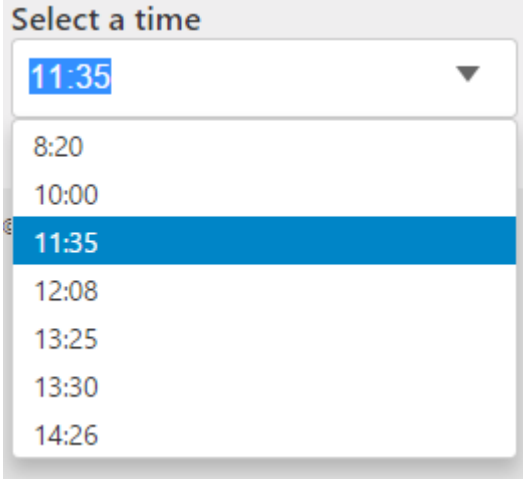
Back to Top

Features

Data Binding

InputTime allows a user to create a list of custom time to be displayed in InputTime dropdown. The `Bind` property in InputTime is used to bind it to a collection by passing a variable to carry out a specific operation. This topic demonstrates how to retrieve data from an existing data source. This is useful for binding custom timings in InpuTime control.

The following image shows how the InputTime appears after it is bound to a list, and uses Bind property to fetch data from the list:



The following code examples demonstrate how to bind InputTime to fetch data from a list:

In Code

DataBindController.cs

C# copyCode

```
public ActionResult Index()
{
    ViewBag.TimeList = new List<object> { "8:20", "10:00", "11:35", "12:08", "13:25",
"13:30", "14:26" };
    return View();
}
```

DataBind.cshtml

HTML Helpers

Razor copyCode

```
@{
    var timeList = ViewBag.TimeList as List<object>;
}
<div>
    <label>Select a time</label>
    @(Html.C1().InputTime().Bind(timeList).Value(DateTime.Parse("11:35")))
</div>
```

Tag Helpers

HTML

```
@{
    var timeList = ViewBag.TimeList as List<object>;
    DateTime dt = DateTime.Parse("11:35");
```

```
}
<div>
    <label>Select a time</label>
    <c1-input-time value="dt" format="hh:mm">
        <c1-items-source source-collection="timeList"></c1-items-source>
    </c1-input-time>
</div>
```

InputTime ASP.NET Core Tags

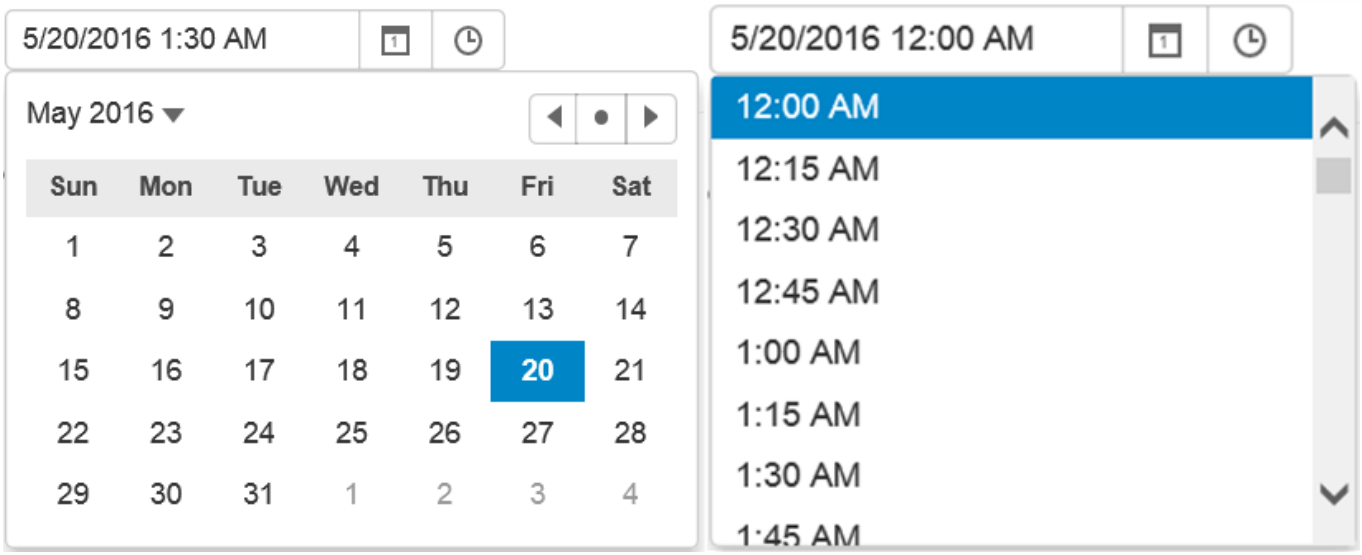
InputTime control supports the following ASP.NET Core Tags:

InputTime class

- class
- style
- format
- height
- id
- is-dropped-down
- is-editable
- mask
- max
- min
- name
- is-dropped-down-changed
- selected-index-changed
- text-changed
- value-changed
- placeholder
- required
- selected-index
- selected-value
- show-drop-down-button
- step
- template-bindings
- text
- value
- width

InputDateTime

The **InputDateTime** control allows users to enter date and time using any format which is supported by the Globalize class. It also allows the users to pick a date from the drop-down box that displays a calendar and pick a time from the drop-down list. You can define the range of values a user can enter by setting the **Min** and **Max** properties of the control.



Key Features

- Value: Represents the current date time selected by the user.
- Max: Sets the maximum time that the user can enter.
- Min: Sets the minimum time that the user can enter.
- Format: Use this property to represent time in different formats.
- TimeStep: Use this property to specify the number of minutes between entries in the drop-down list.

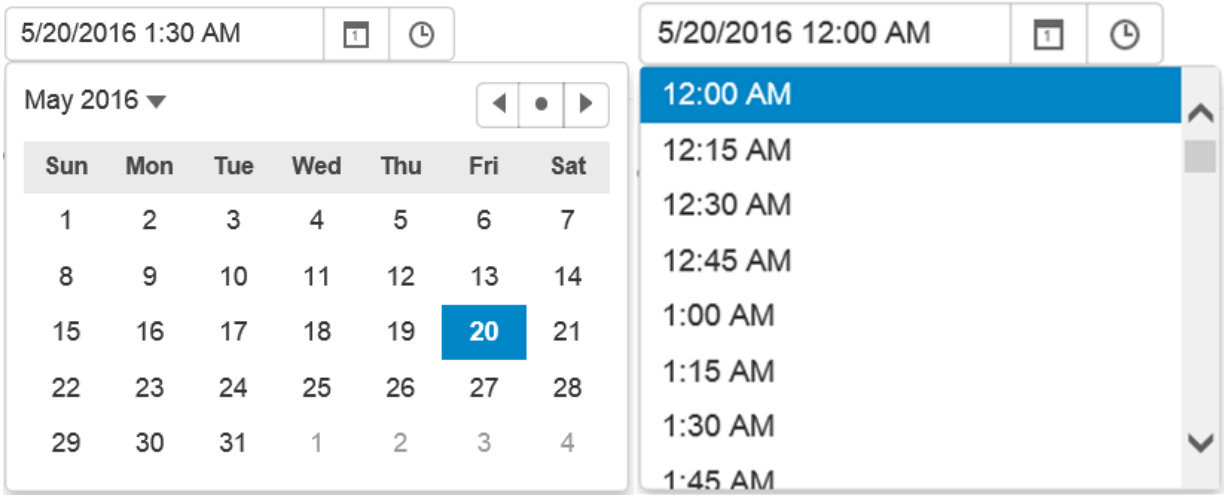
Quick Start

The topic demonstrates how to add an InputDateTime control to your application. For more information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the InputDateTime Control**
- **Step 4: Build and Run the Project**

The following image shows how the **InputDateTime** appears after completing the steps above.



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. Place the cursor inside the method `Index()`.
2. Right click and select **Add View**. The **Add View** dialog appears.
3. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
4. Click **Add** to add a view for the controller.

Step 3: Add the Control

1. Replace the default code of the `Index.cshtml` file with the code given below to initialize an **InputDateTime** control.

HTML Helpers

Razor

```
@{
  var today = DateTime.Now.Date;
  var minDate = new DateTime(today.Year, 1, 1);
  var maxDate = new DateTime(today.Year, 12, 31);
}
<div>
@(Html.C1().InputDateTime().Id("idtInputDateTime").Value(today).Min(minDate).Max(maxDate))
</div>
```

Tag Helpers

Razor

```
@{
  var today = DateTime.Now.Date;
  var minDate = new DateTime(today.Year, 1, 1);
  var maxDate = new DateTime(today.Year, 12, 31);
}
<div> <label> Select a date </label>
<c1-input-datetime id="idtInputDateTime" value="@today" min="@minDate" max="@maxDate">
</c1-input- datetime></div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Index/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

InputDateTime ASP.NET Core Tags

InputDateTime control supports the following ASP.NET Core Tags:

InputDateTime Class

- class
- style
- format
- height
- id
- is-dropped-down
- mask
- max
- min
- name
- is-dropped-down-changed
- text-changed
- value-changed
- placeholder
- required
- show-drop-down-button
- template-bindings
- text
- value
- width
- time-format
- time-max
- time-min
- time-step

ListBox

The ListBox control displays a list of items that may contain plain text or HTML, and lets a user select items with keyboard or mouse. You can populate the list of items in the ListBox by binding it to a model.

Key Features

- DisplayPath: Use the DisplayPath property to specify the name used for the visual representation of the items.
- IsContentHtml: Specifies whether items contain plain text or HTML.

- SelectedItem: Represents the currently selected item in the drop-down list.
- SelectedIndex: Indicates the index of the SelectedItem.
- SelectedValuePath: Represents the name of the property used to get the SelectedValue and SelectedItem.
- SelectedValue: Indicates the value of the SelectedItem, obtained using the SelectedValuePath.
- MaxHeight: Use the MaxHeight property to set the maximum height of the list.

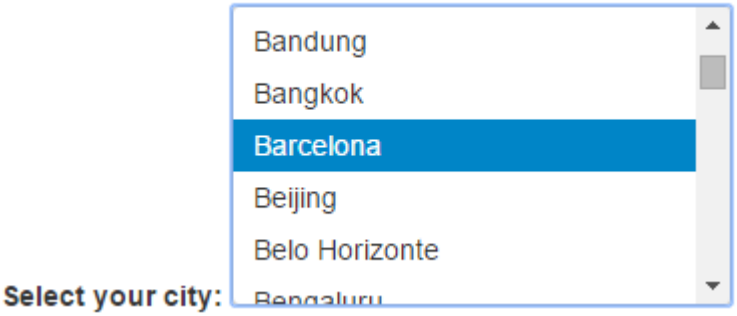
Quick Start

This topic demonstrates how to add a **ListBox** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the ListBox Control**
- **Step 4: Build and Run the Project**

The following image shows how the **ListBox** appears after completing the steps above. The example uses model **Cities.cs** added in the ComboBox QuickStart.



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.
5. Replace the method `Index()` with the following method.

C# copyCode

```
public ActionResult Index()
{
    ViewBag.Cities = MVCFlexGrid.Models.Cities.GetCities();
    return View();
```

```
}
```

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **ListBox** control.

HTML Helpers

Razor copyCode

```
@{
    List<string> cities = ViewBag.Cities;
}
<div>
    <label>Select your city:</label>
    @(Html.C1().ListBox()
        .Bind(cities)
        .Width(250).Height(150)
    )
</div>
```

Tag Helpers

HTML copyCode

```
@{
    List<string> cities = ViewBag.Cities;
}

<div>
    <label>Select your city:</label>
    <div id="list" style="width:250px;height:150px"></div>

    <c1-list-box id="list" selected-index=0 on-client-selected-index-
changed="selectedIndexChanged">
        <c1-items-source source-collection="@cities"></c1-items-source>
    </c1-list-box>

</div>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

Features

Custom ListBox

You can customize the ListBox control by using ItemTemplates to specify custom content in it. You can also use ASP.NET MVC Edition controls in the template for ListBox.

The following image shows how the **ListBox** appears on applying ItemTemplates:

The following code examples demonstrate how to use ItemTemplates to customize the ListBox control:

In Code

ListBoxController.cs

C# copyCode

```
public ActionResult ItemTemplate()
{
    var list = MVCFlexGrid.Models.CustomerOrder.GetOrderData(100).ToList();
    return View(list);
}
```

ListBox.cshtml

HTML Helpers

Razor copyCode

```
@model List<MVCFlexGrid.Models.CustomerOrder>
<style>
    .badge {
        color: white;
        background-color: darkred;
        border-radius: 10px;
        padding: 1px 10px;
    }
    .label {
        color: black;
        background-color: orange;
        border-radius: 10px;
        padding: 1px 10px;
    }
</style>

<script id="template1" type="text/template">
    <span>
        <span class="label">{{Product}}</span>
        <span class="badge">{{Count}}</span>
    </span>
</script>
<div>
    <label>Custom HTML</label>
    @(Html.C1().ListBox()
        .Bind(Model)
        .ItemTemplateId("template1")
        .Width(300).Height(250)
    )
</div>
```

Tag Helpers

HTML copyCode

```
@model List<CustomerOrder>

<style>
    .badge {
        color: white;
        background-color: darkred;
        border-radius: 10px;
        padding: 1px 10px;
    }

    .label {
        color: black;
        background-color: orange;
        border-radius: 10px;
        padding: 1px 10px;
```

```
    }
</style>

<div>
    <label>Custom HTML</label>
    <c1-list-box width="300px" height="250px">
        <c1-input-item-template>
            <span>
        <span class="label">{{Product}}</span>
        <span class="badge">{{Count}}</span>
            </span>
        </c1-input-item-template>
        <c1-items-source source-collection=@Model></c1-items-source>
    </c1-list-box>
</div>

<div>
    <label>C1 MVC controls</label>
    <c1-list-box width="300px" height="250px">
        <c1-input-item-template>
            <span>
                <c1-input-number template-bindings="@(new { Value = "Count"})"
min=0 max=10 step=1 is-template=true></c1-input-number>
            </span>
        </c1-input-item-template>
        <c1-items-source source-collection=@Model></c1-items-source>
    </c1-list-box>
</div>
```

Back to Top

Multi-select ListBox

ListBox supports multi-select functionality, through CheckedMemberPath property. By default, only one item can be selected at a time from the list. However, by using the **CheckedMemberPath** property you can make it function as multi-item selector, with check boxes available corresponding to each item in the listbox.

The **CheckedMemberPath** property controls the check boxes corresponding to each ListBox item. Once a check box is checked or unchecked, the corresponding boolean value will bind to the specified property.

The following image shows a multi-select ListBox with check boxes next to each item. The example uses entity data model to bind listbox control with the **Products** table of the **C1NWind.mdf** database.

The following code examples demonstrate how to enable multi-select functionality in ListBox control:

In Code

MultiSelectController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();
public ActionResult MultiSelect()
{
    return View(db);
}

public ActionResult
ListUpdateProducts([C1JsonRequest]CollectionViewEditRequest<Product> requestData)
{
    return this.C1Json(CollectionViewHelper.Edit<Product>(requestData, item =>
    {
        string error = string.Empty;
        bool success = true;
        try
        {
            db.Entry(item as object).State = EntityState.Modified;
            db.SaveChanges();
        }
        catch (DbEntityValidationException e)
        {
            error = string.Join(",", e.EntityValidationErrors.Select(result =>
            {
                return string.Join(",", result.ValidationErrors.Select(err =>
err.ErrorMessage));
            }));
            success = false;
        }
        catch (Exception e)
        {
            error = e.Message;
            success = false;
        }
```

```
        return new CollectionViewItemResult<Product>
        {
            Error = error,
            Success = success && ModelState.IsValid,
            Data = item
        };
    }, () => db.Products.ToList<Product>()));
}
```

MultiSelect.cshtml

HTML Helpers

Razor copyCode

```
@model C1NWindEntities
@(Html.C1().ListBox()
    .Bind(ib => ib.Bind(Model.Products)
        .Update(Url.Action("ListUpdateProducts")))
    .DisplayMemberPath("ProductName")
    .CheckedMemberPath("Discontinued")
    .Width(400).Height(200)
)
```

Tag Helpers

HTML copyCode

```
@model C1NWindEntities

<div>
    <label>Select an item</label>
    <c1-list-box display-member-path="ProductName"
checked-member-path="Discontinued" selected-value-path="Value"
width="400px" height="200px">
<c1-items-source source-collection="@Model.Products">
</c1-items-source>
    </c1-list-box>
</div>
```

Back to Top

ListBox ASP.NET Core Tags

ListBox control supports the following ASP.NET Core Tags:

ListBox class

- class
- style

- display-member-path
- height
- id
- is-content-html
- item-formatter
- ItemsSource
- max-height
- items-changed
- selected-index-changed
- selected-index
- selected-item
- selected-value
- selected-value-path
- template-bindings
- width
- on-client-item-checked
- checked-member-path
- item-template
- Id

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- on-client-collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

- client-converter
- property-name

SortDescriptions

- ascending
- property

Menu

The Menu control displays a text element with a drop-down list of commands that the user can invoke by click. Menu inherits from ComboBox, so you can use the **ItemsSource** property to populate and style the control similarly. The Menu control adds an ItemClicked event that fires when the user selects an item from the menu. The event handler event inspects the Menu control to determine which item was clicked.

The Menu control also supports an ItemClicked event, which fires when a user selects an item from the menu. The event handler inspects the control to determine the item that is clicked.

Key Features

- **ItemsSource**: Use the ItemSource property to set the array or ICollectionView object that contains the items to select from.
- **Placeholder**: Use the Placeholder property to set string shown as a hint when the control is empty.
- **SelectedIndex**: Use the SelectedIndex property to specify the index of the SelectedItem.
- **SelectedValue**: Use the SelectedVaue property to set the value of the SelectedItem, obtained using the SelectedValuePath.

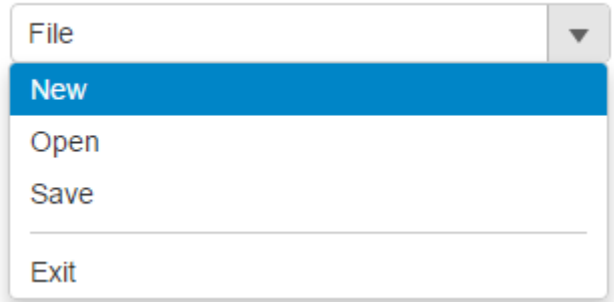
Quick Start

This topic demonstrates how to add an **Menu** control to your application. For information on how to add ASP.NET MVC Edition controls , see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Menu Control**
- **Step 4: Build and Run the Project**

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



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `Default1Controller`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **Menu** control.

HTML Helpers

Razor copyCode

```
<div>
    @(Html.C1().Menu().Header("File")
        .Width(300)
        .MenuItems(items => items
            .Add(item => item.Header("New"))
            .Add(item => item.Header("Open"))
            .Add(item => item.Header("Save"))
            .Add(item => item.IsSeparator(true))
            .Add(item => item.Header("Exit"))
        )
    )
</div>
```

Tag Helpers

HTML copyCode

```
<div>
    <label>ItemClicked Event</label>
    <c1-menu header="File" on-client-item-clicked="itemClicked">
        <c1-menu-item header="New"></c1-menu-item>
        <c1-menu-item header="Open"></c1-menu-item>
        <c1-menu-item header="Save"></c1-menu-item>
        <c1-menu-item is-separator=true></c1-menu-item>
        <c1-menu-item header="Exit"></c1-menu-item>
```

```
    </c1-menu>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default1/index**) in the address bar of the browser to see the view. Or link the view to the home page.

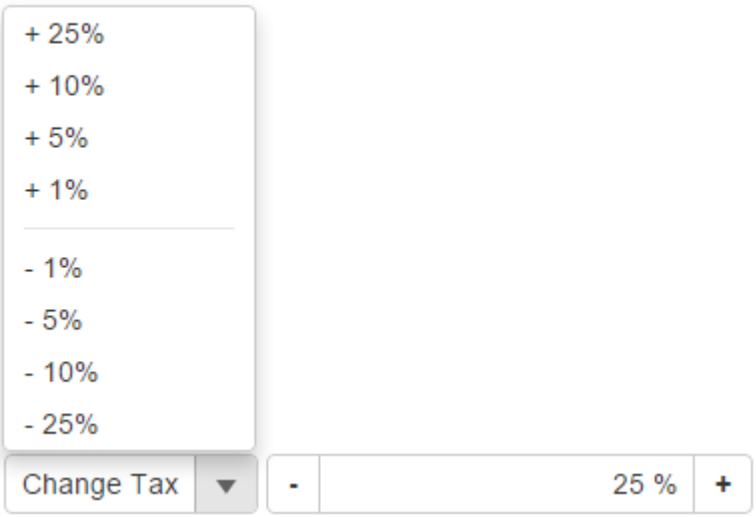
Back to Top

Features

Menu with Input Number

You can add different types of values to the **Menu** control that are as simple as strings shown in the Quick Start, or binding Menu with another control. This section describes how Menu control is used with InputNumber and how is the Menu's value updated with respect to value selected in the InputNumber control.

The following image shows how the **Menu** appears after binding it to Input Number.



The following code examples demonstrate how to bind Menu to an InputNumber control:

In Code

MenuController.cs

Razor copyCode

```
public ActionResult Feature()
{
    return View();
}
```

Menu.cshtml

HTML Helpers

Razor copyCode

```
<script>
    function execute(arg) {
        var inputNumber = wijmo.Control.getControl("#mInputNumber");
        // convert argument to Number
        arg = wijmo.changeType(arg, wijmo.DataType.Number);
        // check if the conversion was successful
        if (wijmo.isNumber(arg)) {
           // update the value
            inputNumber.value += arg;
        }
    }
    function canExecute(arg) {
        var inputNumber = wijmo.Control.getControl("#mInputNumber");
        // convert argument to Number
        arg = wijmo.changeType(arg, wijmo.DataType.Number);
        // check if the conversion was successful
        if (wijmo.isNumber(arg)) {
            var val = inputNumber.value + arg;
            // check if the value is valid
            return val >= 0 && val <= 1;
        }
        return false;
    }
</script>
<div>
    @(Html.C1().Menu().Header("Change Tax")
        .Command("execute", canExecute: "canExecute")
        .MenuItems(items =>
        {
            items.Add("+ 25%", 0.25);
            items.Add("+ 10%", 0.10);
            items.Add("+ 5%", 0.05);
            items.Add("+ 1%", 0.01);
            items.AddSeparator();
            items.Add("- 1%", -0.01);
            items.Add("- 5%", -0.05);
            items.Add("- 10%", -0.10);
            items.Add("- 25%", -0.25);
        })
    )
    @(Html.C1().InputNumber().Id("mInputNumber")
        .Value(0.07).Step(0.05).Format("p0").Min(0).Max(1)
    )
</div>
```

Tag Helpers

HTML copyCode

```
    <script>

    function itemClicked(sender) {
        alert('You\'ve selected option ' + sender.selectedIndex + ' from the ' +
sender.header + ' menu!');
    }

    function execute(arg) {
        var inputNumber = wijmo.Control.getControl("#mInputNumber");

        // convert argument to Number
        arg = wijmo.changeType(arg, wijmo.DataType.Number);

        // check if the conversion was successful
        if (wijmo.isNumber(arg)) {

            // update the value
            inputNumber.value += arg;
        }
    }

    function canExecute(arg) {
        var inputNumber = wijmo.Control.getControl("#mInputNumber");

        // convert argument to Number
        arg = wijmo.changeType(arg, wijmo.DataType.Number);

        // check if the conversion was successful
        if (wijmo.isNumber(arg)) {
            var val = inputNumber.value + arg;

            // check if the value is valid
            return val >= 0 && val <= 1;
        }

        return false;
    }
    </script>

    <c1-menu header="Change Tax" execute-command="execute" can-execute-
command="canExecute">
        <c1-menu-item header="+ 25%" command-parameter="0.25"></c1-menu-item>
        <c1-menu-item header="+ 10%" command-parameter="0.10"></c1-menu-item>
        <c1-menu-item header="+ 5%" command-parameter="0.05"></c1-menu-item>
        <c1-menu-item header="+ 1%" command-parameter="0.01"></c1-menu-item>
        <c1-menu-item is-separator=true></c1-menu-item>
        <c1-menu-item header="- 1%" command-parameter="-0.01"></c1-menu-item>
```

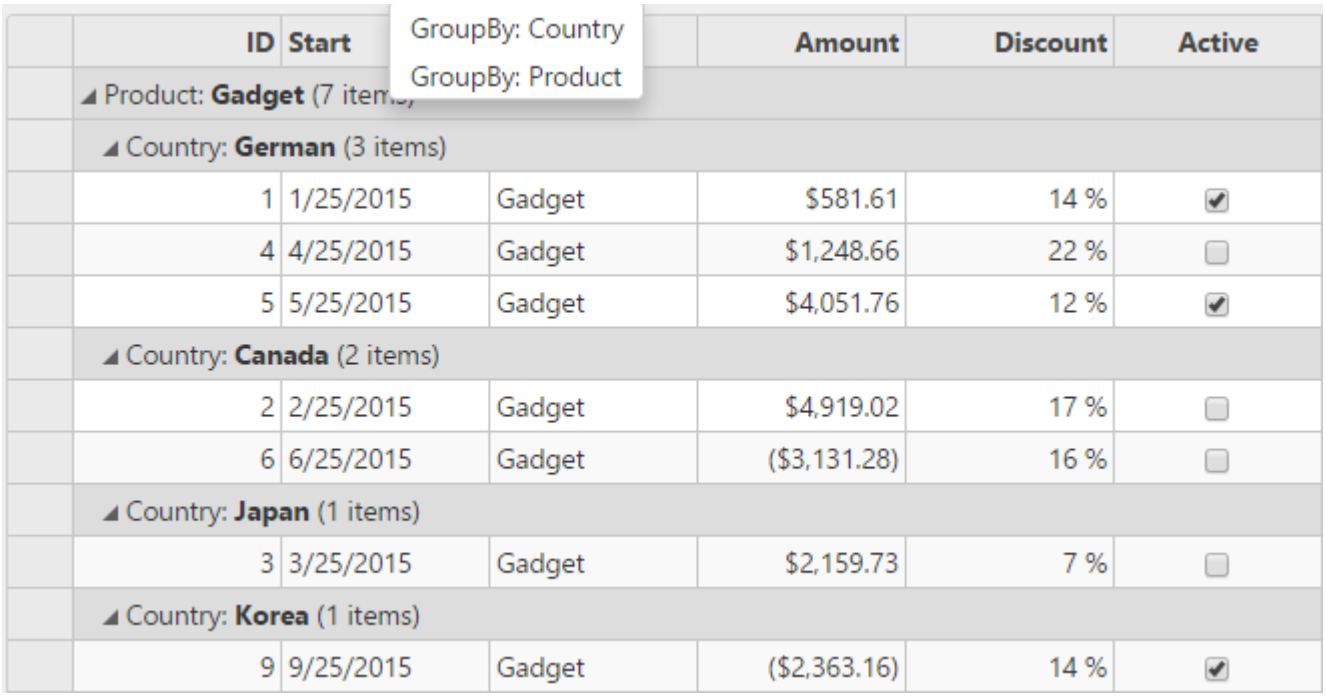
```
        <c1-menu-item header="- 5%" command-parameter="-0.05"></c1-menu-item>
        <c1-menu-item header="- 10%" command-parameter="-0.10"></c1-menu-item>
        <c1-menu-item header="- 25%" command-parameter="-0.25"></c1-menu-item>
    </c1-menu>
    <c1-input-number id="mInputNumber" value=0.07 step=0.05 format="p0" min=0 max=1>
</c1-input-number>
</div>
```

Context Menu

You can use the **Menu** control as a context menu with different MVC controls, such as **FlexGrid**, **ListBox**. When a user right clicks within the control, Menu appears as the context menu with custom fields added by the user. You can add different items in the Menu to perform various operations, such as **filtering**, **sorting** and **grouping** in FlexGrid. This seamless integration of the Menu control with other MVC controls, lets a user customize the control's context menu by adding items in it, and assigning the ID of the control to the Menu's **Owner** property.

In this example, Menu is used as a context menu to enable **grouping** in the FlexGrid control. Two fields, **GroupBy: Country** and **GroupBy: Product** are added in Menu to group FlexGrid's data by column Country and Product. A user can select an item from the context menu by right-clicking within the grid.

The following image shows how **FlexGrid** appears after adding Menu control as a context menu in it:



ID	Start		Amount	Discount	Active
Product: **Gadget** (7 items)					
Country: **German** (3 items)					
1	1/25/2015	Gadget	$581.61	14 %	☑
4	4/25/2015	Gadget	$1,248.66	22 %	☐
5	5/25/2015	Gadget	$4,051.76	12 %	☑
Country: **Canada** (2 items)					
2	2/25/2015	Gadget	$4,919.02	17 %	☐
6	6/25/2015	Gadget	($3,131.28)	16 %	☐
Country: **Japan** (1 items)					
3	3/25/2015	Gadget	$2,159.73	7 %	☐
Country: **Korea** (1 items)					
9	9/25/2015	Gadget	($2,363.16)	14 %	☑

The following code example demonstrates how to use menu control as a Context Menu with FlexGrid:

In Code

ContextMenuController.cs

Razor copyCode

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

ContextMenu.cshtml

HTML Helpers

Razor copyCode

```
@using MVCFlexGrid.Models
@using C1.Web.Mvc.Grid


@model IEnumerable<Sale>

<style>
    .flexGrid {
        margin: 10px;
        padding: 20px;
        color: white;
        display: inline-block;
    }
</style>
<script>
    //Grid Sorting Function using CollectionView
    function setGrouping(arg) {
        var grid = wijmo.Control.getControl("#Sales");
        var cv = grid.collectionView;
        var gd = cv.groupDescriptions;
        gd.push(new wijmo.collections.PropertyGroupDescription(arg));

    }
</script>

<div class="multi-grid">
    @(Html.C1().FlexGrid<Sale>()
    .AutoGenerateColumns(false)
    .Id("Sales")
    .Height(450)
    .Width(700)
    .AllowAddNew(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Cell)
    .CssClass("grid")
    .Bind(Model)

 // Defining columns to be displayed in FlexGrid

     .Columns(bl =>
       {
           bl.Add(cb => cb.Binding("ID"));
           bl.Add(cb => cb.Binding("Start"));
           bl.Add(cb => cb.Binding("Product"));
           bl.Add(cb => cb.Binding("Amount").Format("c"));
           bl.Add(cb => cb.Binding("Discount").Format("p0"));
```

```
            bl.Add(cb => cb.Binding("Active"));
        })
    )

</div>
//Initialize C1Menu control
@(Html.C1().Menu().Header("Group By")
        .Id("ctxMenu")
        .Command("setGrouping")
        //Add items to be displayed in Context Menu
        .MenuItems(items =>
        {
            items.Add("GroupBy: Country", "Country");
            items.Add("GroupBy: Product", "Product");


        })
    .CssStyle("display", "none")
    .Owner("#Sales")
)
```

Tag Helpers

HTML

```
@using TagHelperExplorer.Models
@using C1.Web.Mvc.Grid
@{
    List<string> group = new List<string> {
        "Country","Product"
    };
}
@model IEnumerable<Sale>

<style>
    .flexGrid {
        margin: 10px;
        padding: 20px;
        color: white;
        display: inline-block;
    }
</style>
<script>
    //Grid Sorting Function using CollectionView
    function setGrouping(arg) {
        var grid = wijmo.Control.getControl("#Sales");
        var cv = grid.collectionView;
        var gd = cv.groupDescriptions;
        gd.push(new wijmo.collections.PropertyGroupDescription(arg));

    }
```

```
</script>

<div >
    <c1-flex-grid auto-generate-columns="false" id="Sales" height="450" width="700"
allow-add-new="true"
                  selection-mode="SelectionMode.Cell" class="grid" > </c1-flex-grid>

     // Defining columns to be displayed in FlexGrid

    <c1-flex-grid-column binding="ID"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Start" format="MMM d yy"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Product"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Amount" format="c"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Discount" format="p0"></c1-flex-grid-column>
    <c1-flex-grid-column binding="Active"></c1-flex-grid-column>


</div>
//Initialize C1Menu control

<c1-menu header="Group By" execute-command="setGrouping" style="display: none"
id="ctxMenu" owner="#Sales">
    <c1-menu-item header="GroupBy: Country" command-parameter="group[0]"></c1-menu-
item>
    <c1-menu-item header="GroupBy: Product" command-parameter="group[1]"></c1-menu-
item>
</c1-menu>
```

Menu ASP.NET Core Tags

Menu control supports the following ASP.NET Core Tags:

Menu class

- can-execute-command
- execute-command
- command-parameter-path
- command-path
- class
- style
- display-member-path
- header
- height
- id
- is-content-html
- is-dropped-down
- is-editable
- item-formatter
- c1-items-source
- max-drop-down-height
- max-drop-down-width

- is-dropped-down-changed
- item-clicked
- selected-index-changed
- text-changed
- placeholder
- required
- selected-index
- selected-item
- selected-value
- selected-value-path
- show-drop-down-button
- template-bindings
- text
- width
- MenuItem
- can-execute-command
- execute-command
- command-parameter
- header
- is-separator
- item-template id

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- filter
- c1-property-group-description
- initial-items-count
- on-client-collecting-query-data
- page-index
- page-size
- read-action-url
- c1-sort-description
- source-collection
- update-action-url

GroupDescription

- client-converter
- property-name

SortDescriptions

- ascending
- property

Popup

The **Popup** control enables you to add small overlays of arbitrary content to your applications, as popovers or modal dialogs. Popovers are the popups which have owner elements to control their visibility, and are placed relative to these owner elements when they appear on the screen. Whereas, dialogs are the popups with no owner elements, and they appear towards the center of the screen distinguished from the rest of the content on the screen.

Key Features

- **Owner:** Specify the element which owns the popup. Owner is null for a dialog.
- **ShowTrigger:** Use the ShowTrigger property to set the action that shows the popup.
- **HideTrigger:** Use the HideTrigger property to set the action that hides the popup.
- **Modal:** Set the boolean value to specify whether the popup should behave as a modal dialog, with dark background and requires the user to respond before program should progress.

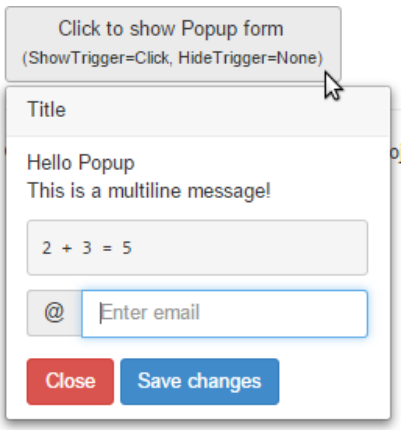
Quick Start

This topic demonstrates how to add a simple popover (**Popup** with owner element) control to your application. Such popup control is similar to a rich tooltip which will show or hide as per the actions specified by ShowTrigger and HideTrigger. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Popup Control**
- **Step 4: Build and Run the Project**

The following image shows how a **Popup** control appears after completing the above steps:



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `DefaultController`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Popup Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **Popup** control.

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc
@Styles.Render("~/Content/css/bootstrap/css/bootstrap.css")
<style>
    .btn {
        height: auto;
    }

    .popover {
        display: block;
    }

    .tab-content > .tab-pane {
        display: block;
    }
</style>

<!-- The contents of popup controls. -->
<div class="popover" id="popup">
    <h3 class="popover-title">
        Title
    </h3>
    <div class="popover-content ">
        <form name="popoverForm">
            <p>Hello Popup<br>This is a multiline message!</p>
            <pre>2 + 3 = <span class="">5</span></pre>
            <div class="form-group">
                <div class="input-group">
                    <div class="input-group-addon">@@</div>
                    <input class="form-control" type="email" placeholder="Enter email">
                </div>
            </div>
            <div class="form-actions">
                <button type="button" class="btn btn-danger wj-hide">Close</button>
                <button type="button" class="btn btn-primary wj-hide">Save changes</button>
            </div>
        </form>
    </div>
</div>
<br />
```

```
<!-- The popup owners. -->
<button id="btn" type="button" class="btn btn-default">
    Click to show Popup form
    <br>
    <small>(ShowTrigger=Click, HideTrigger=None)</small>
</button>

@* Create popup controls with HtmlHelper *@
@(Html.C1().Popup("#popup").Owner("#btn").ShowTrigger(PopupTrigger.Click).HideTrigger(PopupTrigger.None))
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc;

<link href="~/Content/css/bootstrap/css/bootstrap.css" rel="stylesheet" />
<style>
    .btn {
        height: auto;
    }

    .popover {
        display: block;
    }

    .tab-content > .tab-pane {
        display: block;
    }
</style>

<!-- The popup owners. -->
<button id="btn3" type="button" class="btn btn-default">
    Click to show Popup form
    <br>
    <small>(ShowTrigger=Click, HideTrigger=None)</small>
</button>

<c1-popup class="popover" id="popup3" owner="#btn3" hide-trigger="PopupTrigger.None">
    <h3 class="popover-title">
        Title
    </h3>
    <div class="popover-content ">
        <form name="popoverForm">
            <p>Hello Popup<br>This is a multiline message!</p>
            <pre>2 + 3 = <span class="">5</span></pre>
            <div class="form-group">
                <div class="input-group">
                    <div class="input-group-addon">@@</div>
                    <input class="form-control" type="email" placeholder="Enter email">
                </div>
            </div>
            <div class="form-actions">
                <button type="button" class="btn btn-danger wj-hide">Close</button>
                <button type="button" class="btn btn-primary wj-hide">Save changes</button>
            </div>
        </form>
    </div>
</c1-popup>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Default/Index**) in the address bar of the

> browser to see the view. Or link the view to the home page.

Back to Top

Features

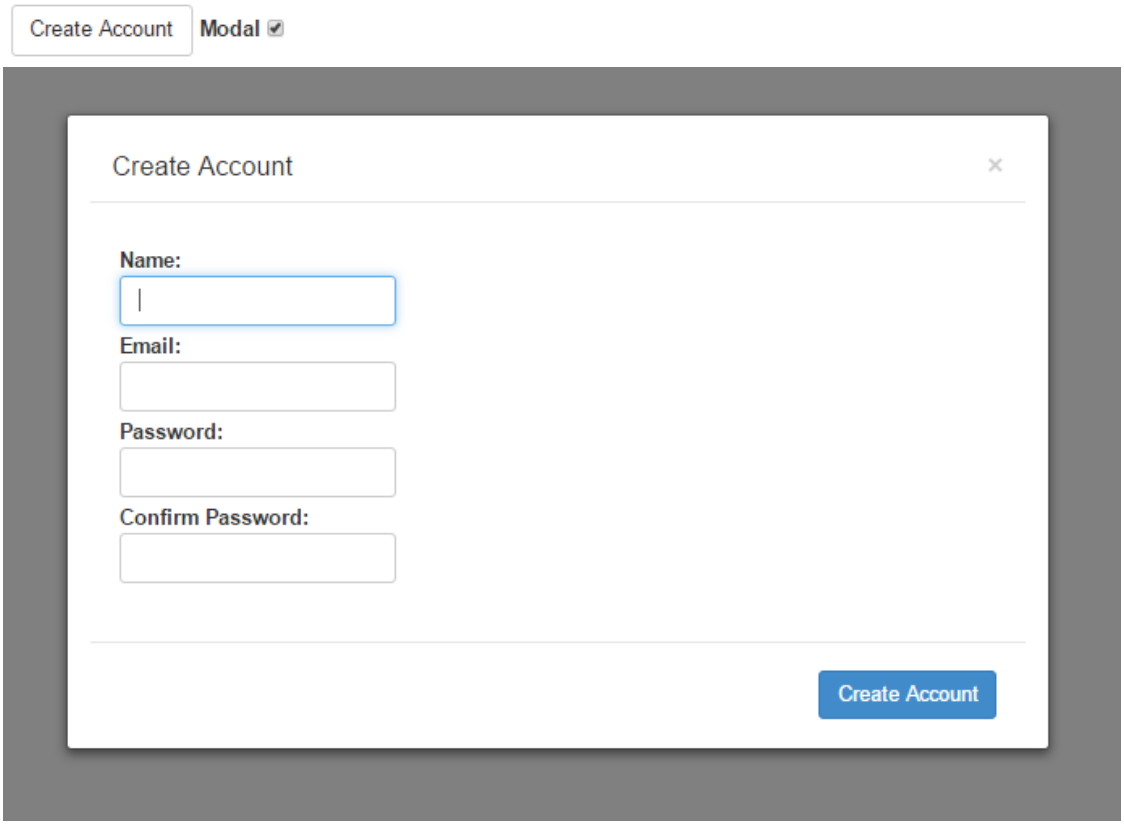
Dialog: Popup with No Owner

Dialog is a popup control with no owner elements to control their visibility. A dialog is displayed by calling the client **show** method. A dialog can be modal or modaless.

Modal dialogs have dark backdrops and need user response to continue the program, while modaless dialogs stay on the screen yet permit other user activities to take place.

A true modal dialog has its hideTrigger property set to **None**, and cannot be closed by simply clicking the backdrop or any other part of the screen. To close a dialog you can call client **hide** method, or use 'wj-hide' class to close it through a clickable element.

The following image shows a modal dialog, that stands out from the other content on the screen and has dark backdrop.



The following code examples demonstrate how to add a modal dialog to your application:

In Code

DialogController.cs

C# copyCode

```
public ActionResult Index()
{
    return View();
}
```

PopupDialog.cshtml

HTML Helpers

Razor copyCode

```
@using C1.Web.Mvc
@Styles.Render("~/Content/css/bootstrap/css/bootstrap.css")
<style>
    .modal-body label {
        padding: 0px;
    }
</style>

<script>
    var popups, modal = true;
    function showPopup(name) {
        if (!popups) {
            popups = {};
            popups["create"] = wijmo.Control.getControl("#popupCreate");

        }
        if (popups[name]) {
            popups[name].modal = modal;
            popups[name].show();
        }
    }

    function check(pwd1, pwd2) {
        if (pwd1.value !== pwd2.value) {
            pwd2.setCustomValidity("Passwords don't match.");
        } else {
            pwd2.setCustomValidity("");
        }
    }

    function checkCreate() {
        var pwd1 = document.getElementById("createPwd1"),
            pwd2 = document.getElementById("createPwd2");
        check(pwd1, pwd2);
    }

    function changeModal() {
        modal = wijmo.getElement("#modal").checked;
    }

    //The first input element will get focus after dialog shown.
    function autoFocus(control) {
        control.hostElement.querySelector("input").focus();
    }
</script>

<!-- The content of popup dialog to create account. -->
<div id="popupCreate" class="modal-content col-md-6">
    <form>
        <h4 class="modal-header">
            Create Account
            <button type="button" tabindex="-1" class="close wj-hide">×</button>
        </h4>
        <div class="modal-body">
            <label>
                Name:
                <input class="form-control" required="" pattern=".{2,}" title="Please enter 2 characters
or more.">
            </label>
            <br>
```

```
            <label>
                Email:
                <input class="form-control" required="" type="email">
            </label>
            <br>
            <label>
                Password:
                <input class="form-control" type="password" id="createPwd1" required="" pattern=".{4,}"
title="Please enter 4 characters or more.">
            </label>
            <br>
            <label>
                Confirm Password:
                <input class="form-control " type="password" id="createPwd2" required="" pattern=".{4,}"
title="Please enter 4 characters or more.">
            </label>
        </div>
        <div class="modal-footer">
            <button class="btn btn-primary" type="submit" onclick="checkCreate()">
                Create Account
            </button>
        </div>
    </form>
</div>
<br />
<button class="btn btn-default" onclick="showPopup('create')">Create Account</button>
<label>Modal <input type="checkbox" id="modal" checked="checked" onchange="changeModal()" /></label>


@(Html.C1().Popup("#popupCreate").Modal(true).HideTrigger(PopupTrigger.None).OnClientShown("autoFocus"))
```

Tag Helpers

HTML copyCode

```
@using C1.Web.Mvc;
<link href="~/Content/css/bootstrap/css/bootstrap.css" rel="stylesheet" />
<style>
    .modal-body label {
        padding: 0px;
    }
</style>
<script>
    var popups, modal = true;
    function showPopup(name) {
        if (!popups) {
            popups = {};
            popups["create"] = wijmo.Control.getControl("#popupCreate");
        }
        if (popups[name]) {
            popups[name].modal = modal;
            popups[name].show();
        }
    }

    function check(pwd1, pwd2) {
        if (pwd1.value !== pwd2.value) {
            pwd2.setCustomValidity("Passwords don't match.");
        } else {
            pwd2.setCustomValidity("");
        }
    }
```

```
    function checkCreate() {
        var pwd1 = document.getElementById("createPwd1"),
            pwd2 = document.getElementById("createPwd2");
        check(pwd1, pwd2);
    }

    function changeModal() {
        modal = wijmo.getElement("#modal").checked;
    }

    //The first input element will get focus after dialog shown.
    function autoFocus(control) {
        control.hostElement.querySelector("input").focus();
    }
</script>

<label>Dialogs (Popups with no owner)</label><br />
<!-- The content of popup dialog to create account. -->
<c1-popup class="modal-content col-md-6" id="popupCreate" hide-trigger="PopupTrigger.None"
shown="autoFocus">
    <form>
        <h4 class="modal-header">
            Create Account
            <button type="button" tabindex="-1" class="close wj-hide">×</button>
        </h4>
        <div class="modal-body">
            <label>
                Name:
                <input class="form-control" required="" pattern=".{2,}" title="Please enter 2 characters
or more.">
            </label>
            <br>
            <label>
                Email:
                <input class="form-control" required="" type="email">
            </label>
            <br>
            <label>
                Password:
                <input class="form-control" type="password" id="createPwd1" required="" pattern=".{4,}"
title="Please enter 4 characters or more.">
            </label>
            <br>
            <label>
                Confirm Password:
                <input class="form-control " type="password" id="createPwd2" required="" pattern=".{4,}"
title="Please enter 4 characters or more.">
            </label>
        </div>
        <div class="modal-footer">
            <button class="btn btn-primary" type="submit" onclick="checkCreate()">
                Create Account
            </button>
        </div>
    </form>
</c1-popup>


<button class="btn btn-default" onclick="showPopup('create')">Create Account</button>
<label>Modal <input type="checkbox" id="modal" checked="checked" onchange="changeModal()" /></label>
```

Back to Top

Popup ASP.NET Core Tags

Popup control supports the following ASP.NET Core Tags:

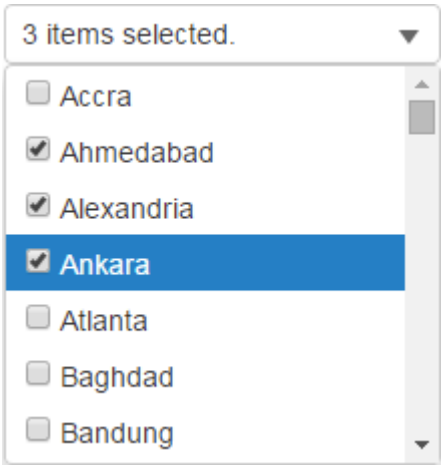
Popup class

- c1-hidden
- class
- fade-in
- fade-out
- height
- hide-trigger
- hiding
- id
- is-template
- modal
- owner
- showing
- shown
- show-trigger
- style
- template-bindings
- width

Back to Top

Multi-select

The Multi-select control enables you to add drop-down list with multiple selection in your application. The control can bound to a collection of custom objects or simple strings, which will show in the dropdown. It uses CheckedMemberPath property, which controls the check boxes corresponding to each ListBox item.
The control has a fully customizable header which, by default, displays upto two items selected from the list. If more than two items are selected from the list, then the header displays the count of selected items. However, you can control this default behavior to- adjust the maximum number of selected items to display in header (MaxHeaderItems), customize the message to be displayed when no items are selected, and set the format string to show the item count (HeaderFormatter).



Key Features

- **CheckedMemberPath:** Set the property name which will control the check boxes placed next to each item.
- **ItemsSource:** Specify the array or ICollectionView object containing the items to select from.
- **MaxHeaderItems:** Specify the count of maximum number of selected items to display on the header.
- **HeaderFormat:** Customize the format string for header content when the number of items checked is more than the count specified in MaxHeaderItems.

Quick Start

This topic demonstrates how to add a **multi-select** control to your application. For information on how to add ASP.NET MVC Edition controls, see Adding Controls.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Add a new Controller and View**
- **Step 3: Add the Multi-select Control**
- **Step 4: Build and Run the Project**

The following image shows how a **Multi-select** control appears after completing the above steps:

Note: The example uses **Countries.cs** model added in AutoComplete QuickStart.

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Add a new Controller and View

Add a new Controller:

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: `Default1Controller`).
    3. Click **Add**.
4. Include the following references.

```
C#                                                                copyCode
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

5. Replace the method Index() with the following method.

```
C#                                                                copyCode
public ActionResult Index()
{
    ViewBag.Countries = Countries.GetCountries();
    return View();
}
```

A new controller is added to the application.

Add a View for the controller:

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the controller (for example: `DefaultController`) to open it.
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add**. A view is added for the controller.

Step 3: Add the Multi-select Control

1. From the **Solution Explorer**, expand the folder **Views**.
2. Double click `Index.cshtml` to open it.
3. Replace the default code of the `Index.cshtml` file with the code given below to initialize a **Multi-select** control.

HTML Helpers

```
Razor                                                             copyCode
@{
```

```
    List<string> countries = ViewBag.Countries;
}
<script>
    var checkedItemsChanged = function (sender, e) {
        var i, result = document.getElementById("result"),
            items = sender.checkedItems;

        result.innerHTML = "";

        for (i = 0; i < items.length; i++) {
            result.innerHTML += "<span>" + (i + 1) + ". " + items[i] + "<\/span>
<br>";
        }
    }
</script>
<div>
<label>Select Countries:</label>@(Html.C1().MultiSelect()
.Name("countries")
.Id("multiselect")
.Bind(countries)
.Placeholder("Please select countries")
.HeaderFormat("{count} countries selected")
.OnClientCheckedItemsChanged("checkedItemsChanged"))
<label>Select results:</label>
<div id="result"></div>
</div>
```

Tag Helpers

HTML copyCode

```
@{
    List<string> countries = ViewBag.Countries;
}
<script>
    var checkedItemsChanged = function (sender,e) {
        var i,result = document.getElementById("result"),
            items = sender.checkedItems;

        result.innerHTML = "";

        for (i = 0; i < items.length; i++) {
            result.innerHTML+="<span>" + (i+1) + ". " + items[i] + "</span>
<br>";
        }
    }
</script>
<div>
    <c1-multi-select id="multiselect" placeholder="Please select countries"
header-format="{count} countries selected" checked-items-
changed="checkedItemsChanged">
```

```
        <c1-items-source source-collection="countries" />
    </c1-multi-select>

    <label>Select results:</label>
    <div align="justify" id="result"></div>
</div>
```

Step 4: Build and Run the Project

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

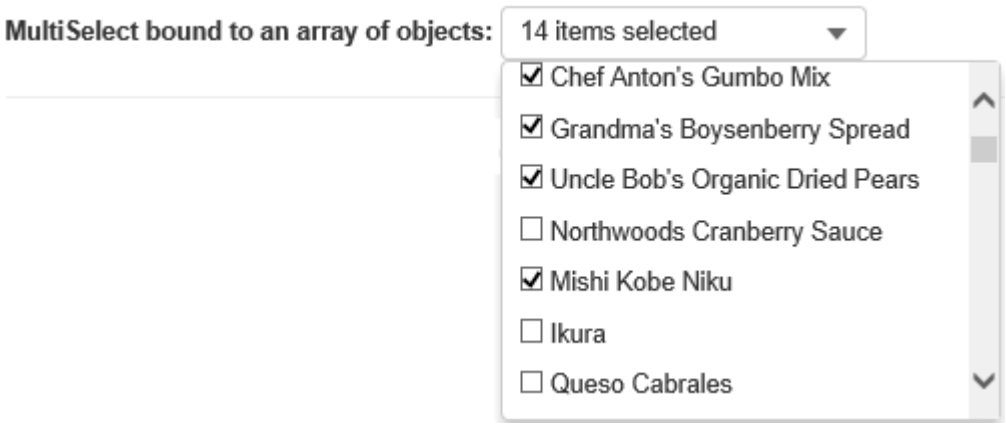
> Append the folder name and view name to the generated URL (for example: http://localhost:1234/**Default/Index**) in the address bar of the browser to see the view. Or link the view to the home page.

Back to Top

Features

Multi-select Bound to an Array of Objects

Multi-select control can be customized to display a list of custom objects in the dropdown. This is accomplished by by binding it with a collection of objects using “DisplayMemberPath”, “SelectedValuePath”, and “CheckedMemberPath” properties.
The following image shows a MultiSelect control bound to a collection of Products from a local NorthWind database file **C1NWind.mdf**.



The following code examples demonstrate how to bind a Multi-select control to a collection of objects:

In Code

ComplexTypeController.cs

C# copyCode

```
private C1NWindEntities db = new C1NWindEntities();
public ActionResult ComplexType()
```

```
{
    return View(db);
}
```

ComplexType.cshtml

HTML Helpers

Razor copyCode

```
@model C1NWindEntities
<br />
<div>
    <label>MultiSelect bound to an array of objects:</label>

    @(Html.C1().MultiSelect()
    .Bind(Model.Products)
    .Name("products")
    .DisplayMemberPath("ProductName")
    .SelectedValuePath("ProductID")
    .CheckedMemberPath("Discontinued")
    )
</div>
```

Tag Helpers

HTML copyCode

```
@model C1NWindEntities

<div>
    <label>MultiSelect bound to an array of objects:</label>

    <c1-multi-select display-member-path="ProductName" selected-value-
path="ProductID" checked-member-path="Discontinued">
        <c1-items-source source-collection="Model.Products" />
    </c1-multi-select>
</div>
```

Back to Top

Multi-select ASP.NET Core Tags

Multi-select control supports the following ASP.NET Core Tags:

MultiSelect class

- c1-input-item-template
- checked-indexes
- checked-items-changed

- checked-values
- checked-member-path
- class
- display-member-path
- for
- header-format
- header-formatter
- height
- id
- is-content-html
- is-dropped-down
- is-dropped-down-changed
- is-editable
- is-template
- item-formatter
- **c1-items-source**
- max-drop-down-height
- max-drop-down-width
- max-header-items
- name
- placeholder
- required
- selected-index
- selected-index-changed
- selected-item
- selected-value
- selected-value-path
- show-drop-down-button
- style
- template-bindings
- text
- text-changed
- width

ItemsSource

- batch-edit
- batch-edit-action-url
- create-action-url
- delete-action-url
- disable-server-read
- **c1-property-group-description**
- initial-items-count
- collecting-query-data
- page-index
- page-size
- read-action-url
- **c1-sort-description**
- source-collection
- update-action-url

GroupDescriptions

- client-converter

- property-name

SortDescriptions

- ascending
- property

Back to Top

MultiAutoComplete

The MultiAutoComplete control allows user to pick multiple items from a list that contains either custom objects or simple strings. Once you enter some text in the control field, MultiAutoComplete starts searching for items that match the entered string and displays a list of possible results. After selecting one of the items from the list, it shows up in the control field.

The control allows you to select multiple items in the similar manner. It also provides an option to set a limit to number of items that can be selected.

Key Features

The MultiAutoComplete control provides various features that enable the developers to build intuitive and professional-looking applications.

- **Maximum Selected Items**
  MultiAutoComplete allows you to set the maximum number of items that can be selected in the control. Set the property to **null** to select any number of items in MultiAutoComplete.

- **Maximum Items Displayed**
  MultiAutoComplete allows you to control the number of items that are displayed in the dropdown.

- **Set Header Path**
  MultiAutoComplete allows you to distinguish the value displayed in the input element from the values shown in the drop-down list.

- **List Items Dynamically**
  MultiAutoComplete allows you to set a function that helps the control to provide list items dynamically as the user types the string.

Quick Start

The topic describes how to add MultiAutoComplete control to your MVC web application and add data to it using model binding. This topic comprises following four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for MultiAutoComplete**
- **Step 3: Add a MultiAutoComplete Control**
- **Step 4: Build and Run the Project**

The following image shows MultiAutoComplete control, after completing the above steps.

In the MultiAutoComplete control below, we are using a list of countries as our datasource. If you type 'ind' in the input element, it provides you with a list of countries that include 'ind' string in their names.

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 1: Create a Datasource for MultiAutoComplete

1. Add a new class to the **Models** folder (for example: `Countries.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `Countries.cs` model. We are using **`Countries`** class to represent list of countries.

C#

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

public class Countries
    {
        public static List<string> GetCountries()
        {
            return new List<string>
            {
                "Afghanistan", "Albania", "Algeria", "American Samoa",
"Andorra", "Angola", "Anguilla", "Antigua", "Argentina", "Armenia",
        "Aruba", "Australia", "Austria", "Azerbaijan", "Bahamas", "Bahrain",
```

```
"Bangladesh", "Barbados", "Belarus", "Belgium", "Belize",
        "Benin", "Bermuda", "Bhutan", "Bolivia", "Bonaire", "Bosnia",
"Botswana", "Brazil", "Brunei", "Bulgaria", "Burkina Faso", "Burundi",
        "Cambodia", "Cameroon", "Canada", "Canary Islands", "Cape Verde",
"Cayman Islands", "Central African Republic", "Chad", "Channel Islands",
        "Chile", "China", "Christmas Island", "Cocos Island", "Colombia",
"Comoros", "Congo", "Cook Islands", "Costa Rica", "Cote D'Ivoire",
        "Croatia", "Cuba", "Curacao", "Cyprus", "Czech Republic", "Denmark",
"Djibouti", "Dominica", "Dominican Republic", "East Timor", "Ecuador",
        "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea", "Fiji",
"Finland",
        "France", "French Guiana", "French Polynesia", "French Southern Ter",
"Gabon", "Gambia", "Georgia", "Germany",
        "Great Britain", "Greece", "Greenland", "Grenada", "Guadeloupe", "Guam",
"Guatemala", "Guinea", "Guyana", "Haiti", "Honduras",
        "Hong Kong", "Hungary", "Iceland", "India", "Indonesia", "Iran", "Iraq",
"Ireland", "Isle of Man", "Israel", "Italy",  "Japan",
        "Qatar", "Republic of Montenegro", "Republic of Serbia", "Reunion",
"Romania", "Russia", "Rwanda",
        "St Helena", "St Kitts-Nevis", "St Lucia", "St Maarten", "Saipan",
"Samoa", "Samoa American", "San Marino", "Saudi Arabia", "Scotland",
        "Senegal", "Serbia", "Seychelles", "Sierra Leone", "Singapore",
"Slovakia", "Slovenia", "Solomon Islands", "Somalia", "South Africa",
        "Spain", "Sri Lanka", "Sudan", "Suriname", "Swaziland", "Sweden",
"Switzerland",
        "Thailand", "Turkey", "Turkmenistan", "Tuvalu", "Uganda",
        "Ukraine", "United Arab Emirates", "United Kingdom", "United States of
America", "Uruguay", "Uzbekistan", "Vanuatu", "Vatican City State",
        "Venezuela", "Vietnam", "Zimbabwe"
            };
        }
    }
```

Back to Top

Step 3: Add a MultiAutoComplete Control

Steps to add a MultiAutoComplete control to the application, are as follows:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (For example: `MultiAutoCompleteController`).
    3. Click **Add**.
4. Include the following references as shown below.

C#

```
using <ApplicationName>.Models;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web;
using System.Web.Mvc;
```

5. Replace the **Index()** method with the following method.

MultiAutoCompleteController.cs

```
public ActionResult Index()
{
   return View(Models.Countries.GetCountries());
}
```

Add a View for the Controller

In the view, we create an instance of MultiAutoComplete control and bind it to a data source using **.Bind** property. We have also defined the maximum numbers of items that can be selected using the MaxSelectedItems property.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `MultiAutoCompleteController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller, and then copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml copyCode

```
@model List<string>

@section Styles{
    <style>
        .highlight {
            background-color: #ff0;
            color: #000;
        }
    </style>
}
<br />
<p>Type in a country name</p>
@(Html.C1().MultiAutoComplete().Bind(Model).CssMatch("highlight")
    .SelectedIndexes(1)
    .Width(300)
    //Set the maximum selection value to 4
    .MaxSelectedItems(4))
```

Tag Helpers

Index.cshtml copyCode

```
@model List<string>

<head>
```

```
    <style>
        .highlight {
            background-color: #ff0;
            color: #000;
        }
    </style>
</head>

<p>Type in a country name</p>
@*Set the maximum selection value to 4*@
<c1-multi-auto-complete css-match="highlight" max-selected-items="4"
    selected-indexes="new List<int> { 1 }">
    <c1-items-source source-collection="@Model">
            </c1-items-source>
</c1-multi-auto-complete>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**MultiAutoComplete/Index**) in the address bar of the browser to see the view.

Back to Top

Features

Learn about concepts that help you to understand how best to use TreeView control.

This section contains information about

Incremental Search
: Learn how to implement incremental search using custom action in MultiAutoComplete.

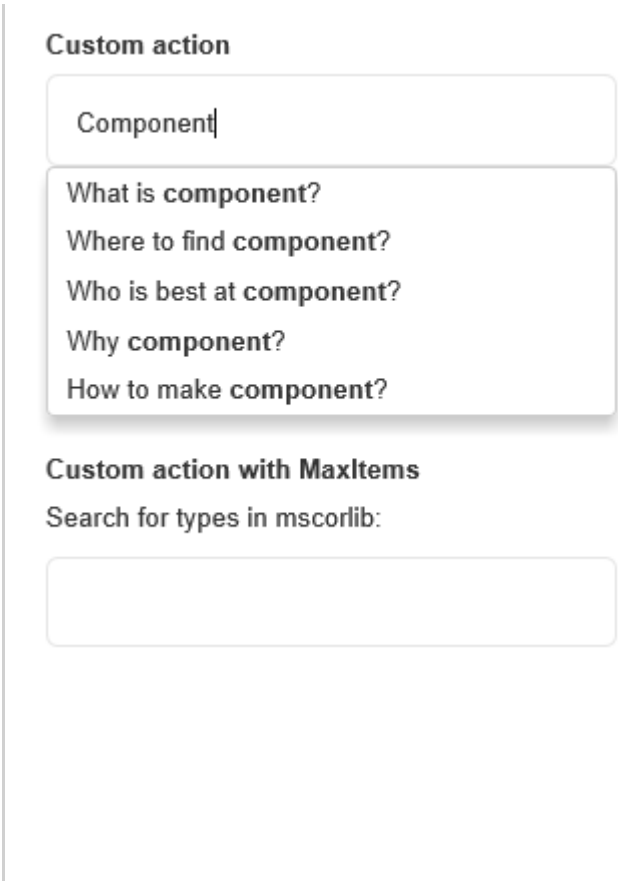
Complex Type
: Learn how to use complex type data in MultiAutoComplete.

Incremental Search

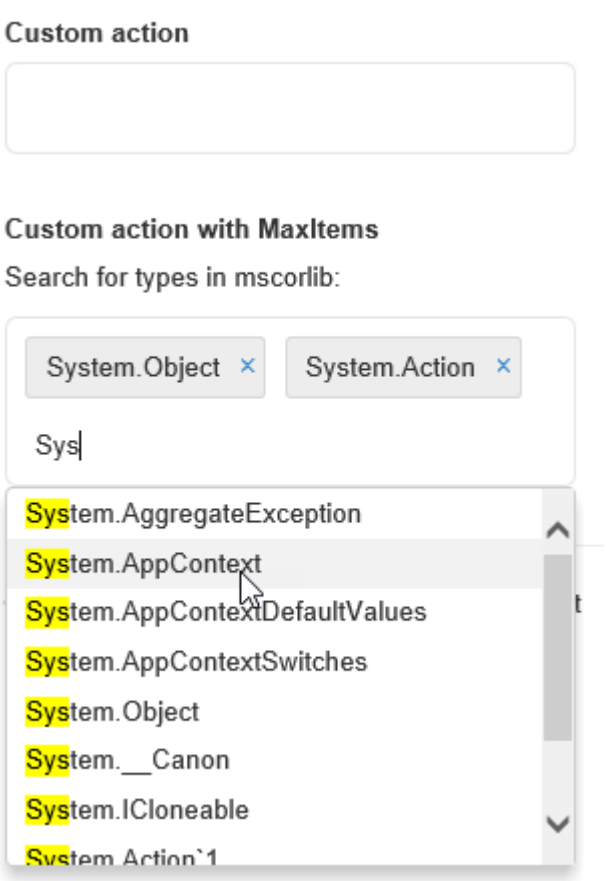
Incremental search or real-time suggestions help the user to progressively search and filter through text. Once you enter the text, one or more possible matches for the text are found and instantly presented to the user. The user may choose a closely related option from the presented list. MultiAutoComplete provides a way to create a custom **ActionResult** for the controllers, which is used to implement incremental search.

Specify the custom action that you want to perform in the **ActionResult**. For example, in the code below we are creating custom action to filter the data after a user types some text and custom action to get the object model namespaces of mscorlib.dll

The following image shows how MultiAutoComplete with custom action method.



Custom Action



Custom Action with MaxItems

The following code examples demonstrate how to use custom action in MultiAutoComplete.

In Code

MultiAutoCompleteController.cs

Razor

```
using System.Linq;
using System.Web.Mvc;
using C1.Web.Mvc.Serialization;

namespace MultiAutoComplete.Controllers
{
    partial class MultiAutoCompleteController
    {
        public ActionResult CustomAction()
        {
            return View();
        }

        public ActionResult Heuristic(string query, int max)
        {
            var prefix = new[] { "What is ", "Where to find ", "Who is best at ",
"Why ",
                        "How to make " };
```

```
            return this.C1Json(prefix.Select(f => f + query + "?").ToList(),
                behavior: JsonRequestBehavior.AllowGet);
        }

        public ActionResult TypesInMscorlib(string query, int max)
        {
            var types = typeof(object).Assembly.GetTypes();
            return this.C1Json(types
                .Where(t => t.FullName.ToUpper().Contains(query.ToUpper()))
                .Select(t => t.FullName)
                .Take(max).ToList(),
                behavior: JsonRequestBehavior.AllowGet);
        }
    }
}
```

MultiAutoComplete.cshtml

HTML Helpers

Razor

```
@section Styles{
    <style>
        .highlight {
            background-color: #ff0;
            color: #000;
        }
    </style>
}
<br />
<div>
    <label>Custom action</label><br />
    @(Html.C1().MultiAutoComplete().Width(300)
        .ItemsSourceAction(Url.Action("Heuristic"))
    )
</div>
<br />
<div>
    <label>Custom action with MaxItems</label><br />
    <p>Search for types in mscorlib:</p>
    @(Html.C1().MultiAutoComplete().Width(300)
        .MaxItems(10).CssMatch("highlight")
        .ItemsSourceAction(Url.Action("TypesInMscorlib"))
    )
</div>
```

Tag Helpers

Razor

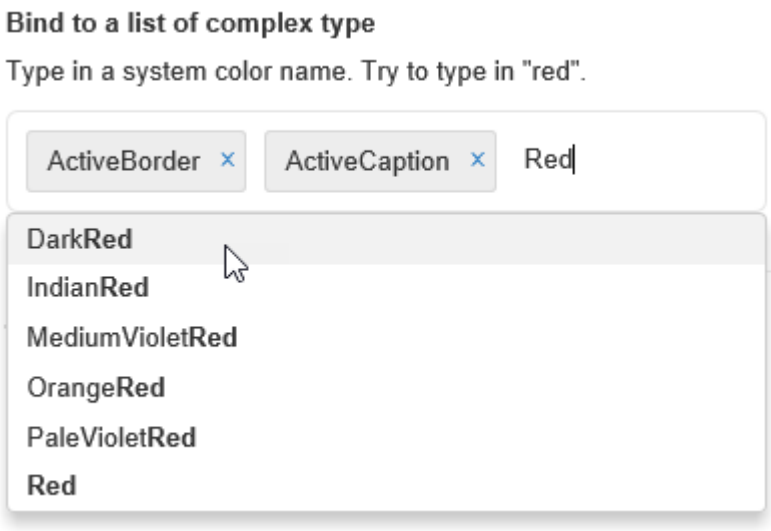
```
@section Styles{
<style>
    .highlight {
        background-color: #ff0;
        color: #000;
    }
</style>
}

<div>
    <label>Custom action</label>
    <c1-multi-auto-complete items-source-action="@Url.Action("Heuristic")">
        </c1-multi-auto-complete>
</div>
<div>
    <label>Custom action with MaxItems</label>
    <p>Search for types in mscorlib:</p>
    <c1-multi-auto-complete max-items="10" css-match="highlight"
        items-source-action="@Url.Action("TypesInMscorlib")">
        </c1-multi-auto-complete>
</div>
```

Complex Type

A complex data type or complex type is a composite of other existing data types. The main advantage that complex data types have over user-defined types is that users can access and manipulate the individual components of a complex data type. MultiAutoComplete can bind a list of ComplexType data. This can be achieved with the help of DisplayMemberPath and SelectedValuePath properties.

In the example below, MultiAutoComplete uses the **KnownColor Enumeration**, which is a part of **System.Drawing** namespace. KnownColor Enum includes a list of all the system colors that are recognized by the system. To access the Enum values, we have created a **NamedColor** model and defined two properties **Name** and **Value**. These properties are then used in the **View** code to access the Enum values.

The following image shows MultiAutoComplete control after setting the **SelectedValuePath** and **DisplayMemberPath** properties.



The following code example demonstrates how to enable complex type data binding in AutoComplete:

Create a Custom Datasource for MultiAutoComplete

1. Add a new class to the folder **Models** (for example: `NamedColor.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `NamedColor.cs` model. We are defining two string variables **Name** and **Value** to access the KnownColor Enum values.

C#

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

namespace MultiAutoComplete.Models
{
    public class NamedColor
    {
        public string Name { get; set; }
        public string Value { get; set; }
    }
}
```

Back to Top

Create a new Controller

AutoCompleteController.cs

Razor

```
using System;
using System.Drawing;
using System.Linq;
using System.Web.Mvc;
using <ApplicationName>.Models;

namespace MultiAutoComplete.Controllers
{
    partial class MultiAutoCompleteController
    {
        public ActionResult ComplexType()
        {
            var list = GetSystemColors();
            return View(list);
        }

        private static NamedColor[] GetSystemColors()
        {
            return Enum.GetValues(typeof(KnownColor))
                .Cast<KnownColor>()
                .Select(c => new NamedColor
```

```
                {
                    Name = c.ToString(),
                    Value = "#" +
Color.FromKnownColor(c).ToArgb().ToString("X8").Substring(2)
                })
                .ToArray();
        }
    }
}
```

View- AutoComplete.cshtml

HTML Helpers

Razor

```
@model IEnumerable<MvcExplorer.Models.NamedColor>

<div>
    <label>Bind to a list of complex type</label>
    <p>Type in a system color name. Try to type in "red".</p>
    @(Html.C1().MultiAutoComplete()
        .Bind(Model)
        .DisplayMemberPath("Name")
        .SelectedIndexes(0,1)
    )
</div>
```

Tag Helpers

Razor

```
@model IEnumerable<MvcExplorer.Models.NamedColor>

<div>
    <label>Bind to a list of complex type</label>
    <p>Type in a system color name. Try to type in "red".</p>
    <c1-multi-auto-complete display-member-path="Name" selected-indexes="new
List<int> { 0, 1 }">
        <c1-items-source source-collection="@Model">
        </c1-items-source>
    </c1-multi-auto-complete>
</div>
```

Back to Top

MultiAutoComplete ASP.NET Core Tags

MultiAutoComplete control supports the following ASP.NET Core Tags:

<c1-multi-auto-complete>

- max-selected-items
- selected-member-path
- selected-items-changed
- selected-indexes

<autocomple-base>

- css-match
- delay
- for
- item-source-action
- items-source-function
- max-items
- min-length
- search-member-path

Common Features

Common features comprises key features for all the **Input** controls available in ASP.NET MVC Edition. Learn about different features for input control categorized as follows.

Scaffolding in Input Controls
: Learn how to implement Scaffolding in Input Controls

AutoExpandSelection property
: Learn how to use AutoExpandSelection property in Input controls.

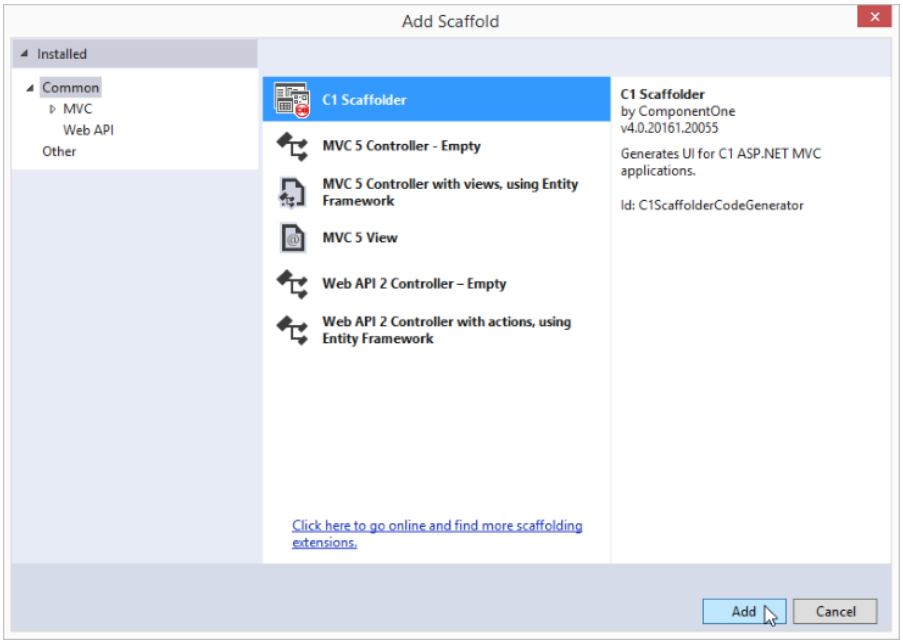
DropDownCssClass property
: Learn how to get or set a CSS class name to add to the controls drop-down element.

Unobtrusive Validation
: Learn how to use the client side unobtrusive validation support for Input Controls.

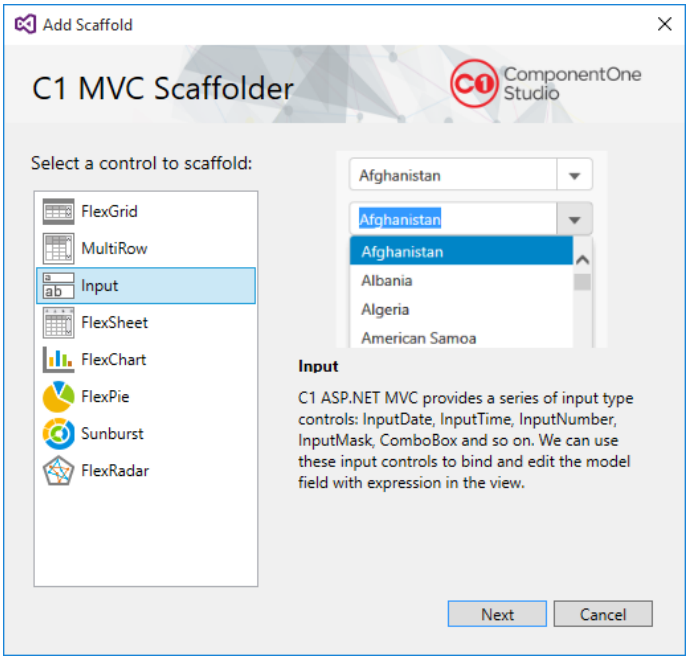
Scaffolding

The steps to scaffold **ComponentOne Input** control for ASP.NET MVC are as follows:

1. Configure the datasource. To configure data source in MVC application, see Data Source Configuration.
2. In the Solution Explorer, right-click the project name and select **Add|New Scaffolded Item**. The **Add Scaffold** wizard appears.
3. In the Add Scaffold wizard, select **Common** and then select **C1 Scaffolder** from the right pane. You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

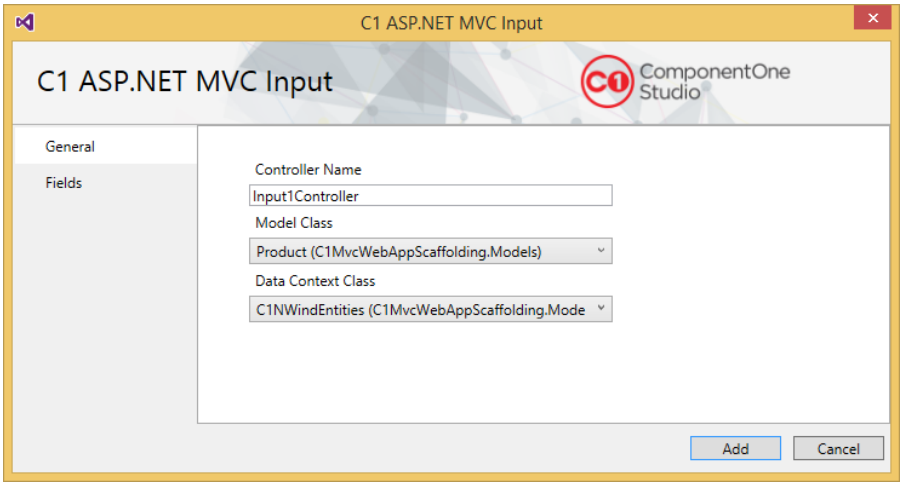


4. Click Add.
5. Select Input control and click Next.

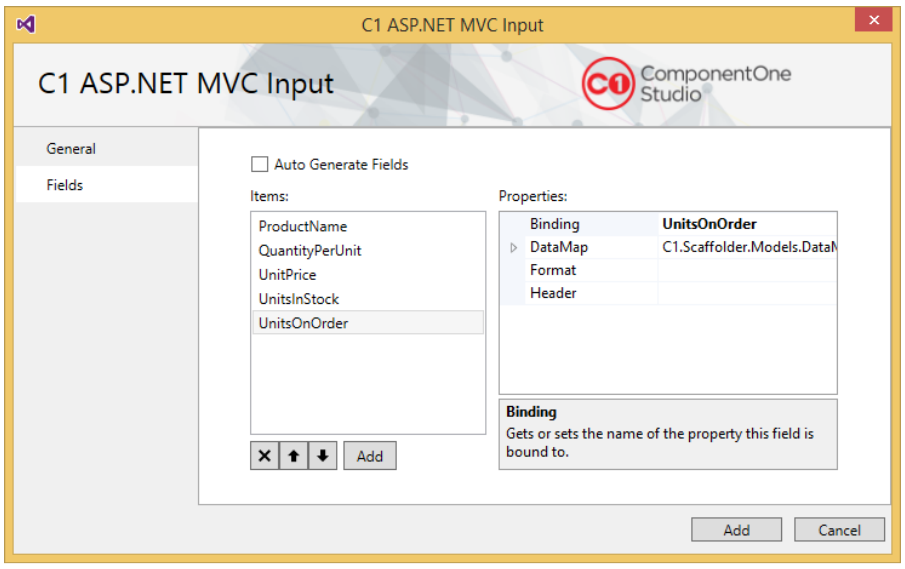


The **C1 ASP.NET MVC Input** wizard appears with the General tab selected by default.

6. In the **General** tab, specify the model as follows:
    1. Type the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the C1NWind.edmx model added in Step 1. In our case, we select Product to populate products in the Input control.
    3. Select **Data Context Class** from the drop-down list. In our case, we select C1NWindEntities.



7. Go to the **Fields** tab to specify the fields in the Input control. By default, **Auto Generate Fields** is checked; if not, then you can add, delete, or move fields upward or downward in the sequence in which they should appear in the final view. In our case, we have selected fields as shown in the following image:



8. Click Add. You will notice that a Controller and five Views - Create, Delete, Details, Edit, and Index for the selected model are added to your project. The code generated for the Controller is as follows:

Input1Controller.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using System.Web;
using System.Data;
using System.Web.Mvc;
using System.Data.Entity;
// This code was generated by C1 Scaffolder.
namespace C1MvcWebAppScaffolding.Controllers
{
    public partial class Input2Controller : Controller
    {
        private C1MvcWebAppScaffolding.Models.C1NWindEntities db = new C1MvcWebAppScaffolding.Models.C1NWindEntities();
        public ActionResult Index()
        {
            var model = db.Products;
            return View(model);
        }

        public ActionResult Details(int key1)
        {
            var model = db.Products.Find(key1);
            if (model == null)
            {
                return HttpNotFound();
            }
```

```
            return View(model);
        }
        public ActionResult Create()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include =
"ProductID,ProductName,SupplierID,CategoryID,QuantityPerUnit,UnitPrice,UnitsInStock,UnitsOnOrder,ReorderLevel,Discontinued")]
C1MvcWebAppScaffolding.Models.Product product)
        {
            if (ModelState.IsValid)
            {
                db.Products.Add(product);
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(product);
        }
        public ActionResult Edit(int key1)
        {
            var model = db.Products.Find(key1);
            if (model == null)
            {
                return HttpNotFound();
            }

            return View(model);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include =
"ProductID,ProductName,SupplierID,CategoryID,QuantityPerUnit,UnitPrice,UnitsInStock,UnitsOnOrder,ReorderLevel,Discontinued")]
C1MvcWebAppScaffolding.Models.Product product)
        {
            if (ModelState.IsValid)
            {
                db.Entry(product).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(product);
        }
        public ActionResult Delete(int key1)
        {
            var model = db.Products.Find(key1);
            if (model == null)
            {
                return HttpNotFound();
            }
            return View(model);
        }
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int key1)
        {
            var model = db.Products.Find(key1);
            db.Products.Remove(model);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

The codes generated for the five views are as follows:

HTML Helpers

Create

Create.cshtml

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@model C1MvcWebAppScaffolding.Models.Product
<script>
    function timeChanged(sender) {
        var date = sender.value;
        var hour = date.getHours();
        var min = date.getMinutes();
        var second = date.getSeconds();
        var inputDate = wijmo.Control.getControl("#inputDateFor" + sender.hostElement.id);
        inputDate.value.setHours(hour);
        inputDate.value.setMinutes(min);
        inputDate.value.setSeconds(second);
        inputDate.refresh();
    }
</script>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Product</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.ProductID, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.ProductID).Required(false)
                @Html.ValidationMessageFor(model => model.ProductID, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.ProductName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputMaskFor(model => model.ProductName)
                @Html.ValidationMessageFor(model => model.ProductName, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.QuantityPerUnit, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputMaskFor(model => model.QuantityPerUnit)
                @Html.ValidationMessageFor(model => model.QuantityPerUnit, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitPrice, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitPrice).Required(false)
                @Html.ValidationMessageFor(model => model.UnitPrice, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitsInStock, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitsInStock).Required(false)
                @Html.ValidationMessageFor(model => model.UnitsInStock, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitsOnOrder, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitsOnOrder).Required(false)
                @Html.ValidationMessageFor(model => model.UnitsOnOrder, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div>
                <input type="submit" value="Create"/>
            </div>
        </div>
    </div>
}
<div>
```

```
    @Html.ActionLink("Back to List", "Index")
</div>
```

Delete

Delete.cshtml

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@model C1MvcWebAppScaffolding.Models.Product
<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Product</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.ProductName)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.ProductName)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.QuantityPerUnit)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.QuantityPerUnit)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitPrice)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitPrice)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitsInStock)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitsInStock)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitsOnOrder)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitsOnOrder)
        </dd>
    </dl>
    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()
        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

Details

Details.cshtml

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@model C1MvcWebAppScaffolding.Models.Product
<div>
    <h4>Product</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.ProductName)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.ProductName)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.QuantityPerUnit)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.QuantityPerUnit)
```

```
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitPrice)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitPrice)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitsInStock)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitsInStock)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.UnitsOnOrder)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.UnitsOnOrder)
        </dd>
    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new {key1=Model.ProductID}) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

Edit

Edit.cshtml

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@model C1MvcWebAppScaffolding.Models.Product
<script>
    function timeChanged(sender) {
        var date = sender.value;
        var hour = date.getHours();
        var min = date.getMinutes();
        var second = date.getSeconds();
        var inputDate = wijmo.Control.getControl("#inputDateFor" + sender.hostElement.id);
        inputDate.value.setHours(hour);
        inputDate.value.setMinutes(min);
        inputDate.value.setSeconds(second);
        inputDate.refresh();
    }
</script>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Product</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.ProductID)
        <div class="form-group">
            @Html.LabelFor(model => model.ProductName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputMaskFor(model => model.ProductName)
                @Html.ValidationMessageFor(model => model.ProductName, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.QuantityPerUnit, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputMaskFor(model => model.QuantityPerUnit)
                @Html.ValidationMessageFor(model => model.QuantityPerUnit, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitPrice, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitPrice).Required(false)
                @Html.ValidationMessageFor(model => model.UnitPrice, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitsInStock, htmlAttributes: new { @class = "control-label col-md-2" })
```

```
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitsInStock).Required(false)
                @Html.ValidationMessageFor(model => model.UnitsInStock, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UnitsOnOrder, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.C1().InputNumberFor(model => model.UnitsOnOrder).Required(false)
                @Html.ValidationMessageFor(model => model.UnitsOnOrder, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div>
                <input type="submit" value="Save"/>
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

Index

Index.cshtml

```
@using C1.Web.Mvc
@using C1.Web.Mvc.Fluent
@model IEnumerable<C1MvcWebAppScaffolding.Models.Product>
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.ProductName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.QuantityPerUnit)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.UnitPrice)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.UnitsInStock)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.UnitsOnOrder)
        </th>
        <th></th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(model => item.ProductName)
        </td>
        <td>
            @Html.DisplayFor(model => item.QuantityPerUnit)
        </td>
        <td>
            @Html.DisplayFor(model => item.UnitPrice)
        </td>
        <td>
            @Html.DisplayFor(model => item.UnitsInStock)
        </td>
        <td>
            @Html.DisplayFor(model => item.UnitsOnOrder)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new {key1=item.ProductID}) |
            @Html.ActionLink("Details", "Details", new {key1=item.ProductID}) |
            @Html.ActionLink("Delete", "Delete", new {key1=item.ProductID})
        </td>
    </tr>
}
</table>
```

Tag Helpers

Create

Create.cshtml

```
@using C1.Web.Mvc
@addTagHelper *, C1.AspNetCore.Mvc
@model C1MvcApplication1.Models.Product
<script>
    function timeChanged(sender) {
        var date = sender.value;
        var hour = date.getHours();
        var min = date.getMinutes();
        var second = date.getSeconds();
        var inputDate = wijmo.Control.getControl("#inputDateFor" + sender.hostElement.id);
        inputDate.value.setHours(hour);
        inputDate.value.setMinutes(min);
        inputDate.value.setSeconds(second);
        inputDate.refresh();
    }
</script>
<form asp-action="Create">
    <div class="form-horizontal">
        <h4>Product</h4>
        <hr />
        <div asp-validation-summary="ValidationSummary.ModelOnly" class="text-danger"></div>
        <div class="form-group">
            <label asp-for="ProductName" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-mask for="@Model.ProductName">
                </c1-input-mask>
                <span asp-validation-for="ProductName" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="QuantityPerUnit" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-mask for="@Model.QuantityPerUnit">
                </c1-input-mask>
                <span asp-validation-for="QuantityPerUnit" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitPrice" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitPrice" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitPrice" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitsInStock" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitsInStock" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitsInStock" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitsOnOrder" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitsOnOrder" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitsOnOrder" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
</form>
<div>
    <a asp-action="Index">Back to List</a>
</div>
```

Delete

Delete.cshtml

```
@using C1.Web.Mvc
@addTagHelper "*, C1.Web.Mvc"
@model C1MvcApplication1.Models.Product
<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Product</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            ProductName
        </dt>
        <dd>
            @Model.ProductName
        </dd>
        <dt>
            QuantityPerUnit
        </dt>
        <dd>
            @Model.QuantityPerUnit
        </dd>
        <dt>
            UnitPrice
        </dt>
        <dd>
            @Model.UnitPrice
        </dd>
        <dt>
            UnitsInStock
        </dt>
        <dd>
            @Model.UnitsInStock
        </dd>
        <dt>
            UnitsOnOrder
        </dt>
        <dd>
            @Model.UnitsOnOrder
        </dd>
    </dl>
    <form asp-action="Delete" asp-route-key1="@Model.ProductID">
        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            <a asp-action="Index">Back to List</a>
        </div>
    </form>
</div>
```

Details

Details.cshtml

```
@using C1.Web.Mvc
@addTagHelper "*, C1.Web.Mvc"
@model C1MvcApplication1.Models.Product
<div>
    <h4>Product</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            ProductName
        </dt>
        <dd>
            @Model.ProductName
        </dd>
        <dt>
            QuantityPerUnit
        </dt>
        <dd>
            @Model.QuantityPerUnit
        </dd>
        <dt>
            UnitPrice
```

```
        </dt>
        <dd>
            @Model.UnitPrice
        </dd>
        <dt>
            UnitsInStock
        </dt>
        <dd>
            @Model.UnitsInStock
        </dd>
        <dt>
            UnitsOnOrder
        </dt>
        <dd>
            @Model.UnitsOnOrder
        </dd>
    </dl>
</div>
<p>
    <a asp-action="Edit" asp-route-key1="@Model.ProductID">Edit</a> |
    <a asp-action="Index">Back to List</a>
</p>
```

Edit

Edit.cshtml

```
@using C1.Web.Mvc
@addTagHelper "*, C1.Web.Mvc"
@model C1MvcApplication1.Models.Product
<script>
    function timeChanged(sender) {
        var date = sender.value;
        var hour = date.getHours();
        var min = date.getMinutes();
        var second = date.getSeconds();
        var inputDate = wijmo.Control.getControl("#inputDateFor" + sender.hostElement.id);
        inputDate.value.setHours(hour);
        inputDate.value.setMinutes(min);
        inputDate.value.setSeconds(second);
        inputDate.refresh();
    }
</script>
<form asp-action="Edit">
    <div class="form-horizontal">
        <h4>Product</h4>
        <hr />
        <div asp-validation-summary="ValidationSummary.ModelOnly" class="text-danger"></div>
        <div class="form-group">
            <label asp-for="ProductName" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-mask for="@Model.ProductName">
                </c1-input-mask>
                <span asp-validation-for="ProductName" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="QuantityPerUnit" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-mask for="@Model.QuantityPerUnit">
                </c1-input-mask>
                <span asp-validation-for="QuantityPerUnit" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitPrice" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitPrice" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitPrice" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitsInStock" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitsInStock" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitsInStock" class="text-danger"></span>
```

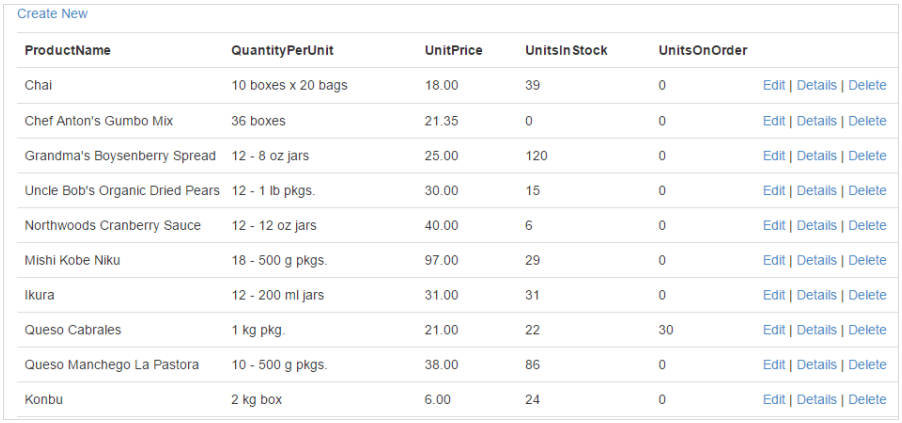
```
            </div>
        </div>
        <div class="form-group">
            <label asp-for="UnitsOnOrder" class="control-label col-md-2"></label>
            <div class="col-md-10">
                <c1-input-number for="@Model.UnitsOnOrder" required="false">
                </c1-input-number>
                <span asp-validation-for="UnitsOnOrder" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
</form>
<div>
    <a asp-action="Index">Back to List</a>
</div>
```

Index

Index.cshtml

```
@using C1.Web.Mvc
@addTagHelper "*, C1.Web.Mvc"
@model IEnumerable<C1MvcApplication1.Models.Product>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <tr>
        <th>ProductName</th>
        <th>QuantityPerUnit</th>
        <th>UnitPrice</th>
        <th>UnitsInStock</th>
        <th>UnitsOnOrder</th>
        <th></th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>
            @item.ProductName
        </td>
        <td>
            @item.QuantityPerUnit
        </td>
        <td>
            @item.UnitPrice
        </td>
        <td>
            @item.UnitsInStock
        </td>
        <td>
            @item.UnitsOnOrder
        </td>
        <td>
            <a asp-action="Edit" asp-route-key1="@item.ProductID">Edit</a> |
            <a asp-action="Details" asp-route-key1="@item.ProductID">Details</a> |
            <a asp-action="Delete" asp-route-key1="@item.ProductID">Delete</a>
        </td>
    </tr>
}
</table>
```

9. Run the project.


Create New

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	
Chai	10 boxes x 20 bags	18.00	39	0	Edit | Details | Delete
Chef Anton's Gumbo Mix	36 boxes	21.35	0	0	Edit | Details | Delete
Grandma's Boysenberry Spread	12 - 8 oz jars	25.00	120	0	Edit | Details | Delete
Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	30.00	15	0	Edit | Details | Delete
Northwoods Cranberry Sauce	12 - 12 oz jars	40.00	6	0	Edit | Details | Delete
Mishi Kobe Niku	18 - 500 g pkgs.	97.00	29	0	Edit | Details | Delete
Ikura	12 - 200 ml jars	31.00	31	0	Edit | Details | Delete
Queso Cabrales	1 kg pkg.	21.00	22	30	Edit | Details | Delete
Queso Manchego La Pastora	10 - 500 g pkgs.	38.00	86	0	Edit | Details | Delete
Konbu	2 kg box	6.00	24	0	Edit | Details | Delete

Unobtrusive Validation

Unobtrusive Validation support for Input controls helps you to validate any C1 input control using client side validation. Unobtrusive validation can implement simple client-side validation without writing a bulk of validation code, and improves the user experience simply by adding the right attributes and including the script files.

In a common validation scenario, when we use a validation to validate any control and use client side validation, JavaScript code is generated and rendered as HTML on the web browser. However, with unobtrusive validation inline JavaScript is not generated for rendering to handle client side validation. Instead, it uses HTML5 data-* attributes for client side validations.

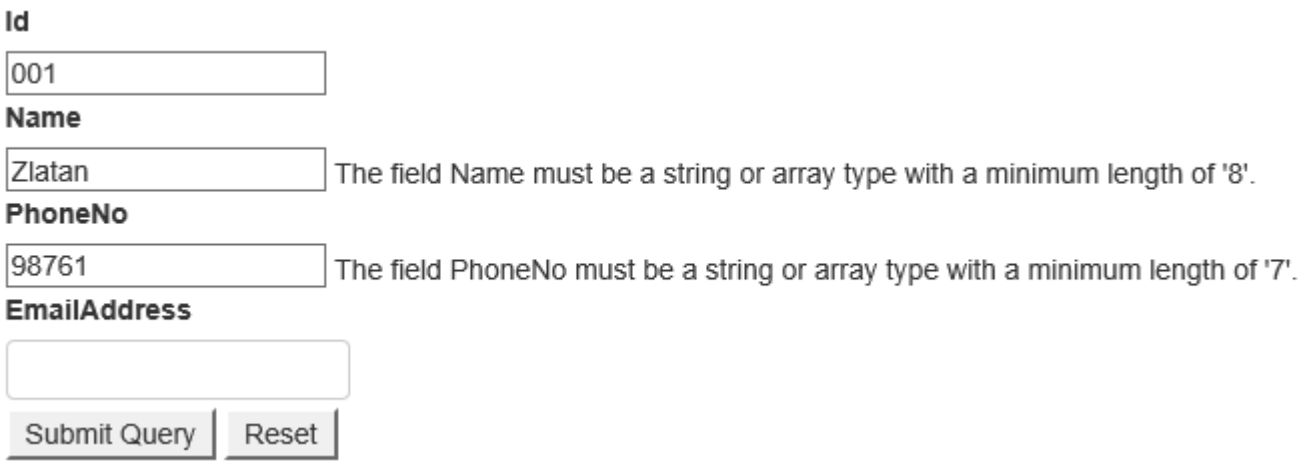
Unobtrusive Validation support is available in the following Input controls;

- InputColor
- InputDate
- InputDateTime
- InputMask
- InputNumber
- AutoComplete
- ComboBox
- InputTime
- MultiSelect

Before implementing the below steps, you need to create a new MVC application using ComponentOne template or Visual Studio template.

This topic comprises of four steps:

- **Step 1: Configure your MVC application**
- **Step 2: Create Validations for Input controls**
- **Step 3: Add an Input control**
- **Step 4: Build and Run the Project**



Step 1: Configure your MVC application

Using HTML Helpers

Complete the following steps to add the js file references to your application.

1. From the **Solution Explorer**, open the folders **Views | Shared**.
2. Double click **_Layout.cshtml** to open it.
3. Add the following code between the <head></head> tags.

Razor

```
<script src="~/Scripts/jquery-1.10.2.js"></script>
<script src="~/Scripts/jquery.validate.js"></script>
<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
```

Using Tag Helpers

In case of tag helpers, references to the script files are automatically added to your application.

Step 2: Create Validations for Input controls.

Model - Form.cs

Razor

```
using System.ComponentModel.DataAnnotations;
namespace UnobtrusiveValidationSample.Models
{
    public class Form
    {
        [Required]

        public string Id { get; set; }

        [Required]
        [DataType(DataType.Text)]
```

```
        [MinLength(8)]
        [MaxLength(30)]

        public string Name { get; set; }

        [Required]
        [DataType(DataType.PhoneNumber)]
        [Phone]
        [Range(0, 10000000000)]
        [StringLength(11)]
        [MinLength(7)]
        [MaxLength(10)]

        public string PhoneNo { get; set; }

        [Required]
        [DataType(DataType.EmailAddress)]
        [EmailAddress]
        [MinLength(10)]
        [MaxLength(50)]

        public string EmailAddress { get; set; }

    }
}
```

Step 3: Add an Input control

View - Index.cshtml

HTML Helpers

Razor

```
@using UnObtrusiveValidationSample.Models;
@model Form
@using (Html.BeginForm())
{
    <br />
    @Html.LabelFor(m => m.Id)
    <br />
    @Html.EditorFor(m => m.Id)
    @Html.ValidationMessageFor(m => m.Id)
    <br />

    @Html.LabelFor(m => m.Name)
    <br />
    @Html.EditorFor(m => m.Name)
    @Html.ValidationMessageFor(m => m.Name)
    <br />
```

```
    @Html.LabelFor(m => m.PhoneNo)
    <br />
    @Html.EditorFor(m => m.PhoneNo)
    @Html.ValidationMessageFor(m => m.PhoneNo)
    <br />

    @Html.LabelFor(m => m.EmailAddress)
    <br />
    @Html.C1().InputMaskFor(m => m.EmailAddress)
    @Html.ValidationMessageFor(m => m.EmailAddress)
    <br />
    <input type="submit" />
    <input type="reset" />
}
```

Tag Helpers

Razor

```
@model Form
@section Scripts{
@{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
<form>
    <br />
    <label asp-for="Id"></label>
    <br />
    <input asp-for="Id" />
    <span asp-validation-for="Id"></span>
    <br />

    <label asp-for="Name"></label>
    <br />
    <input asp-for="Name" />
    <span asp-validation-for="Name"></span>
    <br />

    <label asp-for="PhoneNo"></label>
    <br />
    <input asp-for="PhoneNo" />
    <span asp-validation-for="PhoneNo"></span>
    <br />

    <label asp-for="EmailAddress"></label>
    <br />
    <c1-input-mask for="EmailAddress"></c1-input-mask>
    <span asp-validation-for="EmailAddress"></span>
    <br />

    <input type="submit" />
    <input type="reset" />
```

```
</form>
```

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.

Back to Top

AutoExpandSelection

You can use the AutoExpandSelection property to specify whether DropDown control automatically fills the text box portion of a combo box with a value from the combo box list that matches the characters you enter as you type in the combo box.

> **Note:** In case of **InputColor** and **InputDate** control, the default value of AutoExpandSelection property is set to **True**. For all the other Input controls the default value is False.

AutoExpandSelection property is available in the following Input DropDown controls;

- InputColor
- InputDate
- AutoComplete, ComboBox
- InputTime

The following code example demonstrates how to use AutoExpandSelection in the Input control.

IndexController.cs

Razor

```
public ActionResult Index()
  {
ViewBag.Cities = Models.Cities.GetCities();
return View();
   }
```

Index.cshtml

HTML Helpers

Razor

```
@using C1.Web.Mvc;
@{

List<string> cities = ViewBag.Cities;

}
@(Html.C1().ComboBox().Bind(cities).SelectedIndex(0)
.AutoExpandSelection(true))
```

Tag Helpers

Razor

```
@{

List<string> cities = ViewBag.Cities;

}
<div>

    <c1-combo-box selected-index=0 auto-expand-selection="true">
        <c1-items-source source-collection=@cities></c1-items-source>
    </c1-combo-box>
</div>
```

DropDownCssClass

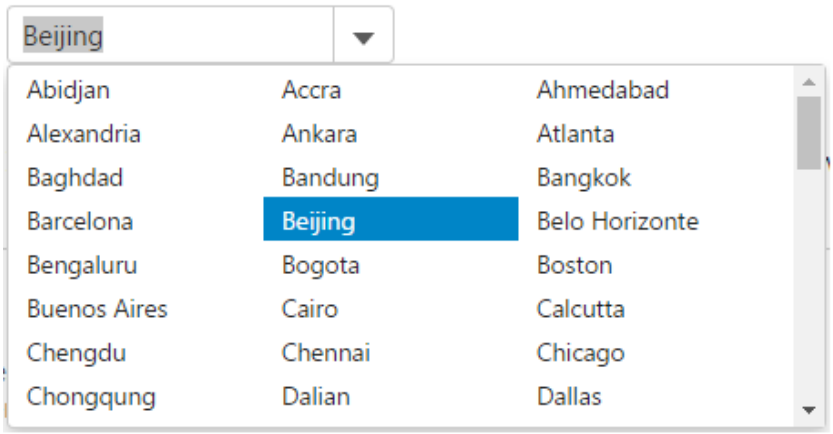
You can get or set a CSS class name to add to the controls drop-down element. The property is useful when you are styling the drop-down element, because it is shown as a child of the document body rather than as a child of the control itself, which prevents using CSS selectors based on the parent control. You can use this property to display multiple columns in a ComboBox dropdown.

The following code example demonstrates how to enable multi-column using DropDownCssClass property. This example uses the sample created in the Quick Start section.

DropDownCssClass property is available in the following Input DropDown controls;

- InputColor
- InputDate
- AutoComplete, ComboBox
- InputTime
- MultiSelect
- Menu

The following image shows how the Menu appears after applying the **DropDownCssClass** property:



The following code example demonstrates how to use DropDownCssClass property.

IndexController.cs

Razor

```
public ActionResult Index()
        {
        ViewBag.Cities = Models.Cities.GetCities();
        return View();
        }
```

ComboBox.cshtml

HTML Helpers

Razor

```
<style>
    .multi-column
    {   display: flex;
        flex-wrap: wrap;
        max-width: 450px;
    }
</style>
@{ List<string> cities = ViewBag.Cities;}
@Html.C1().ComboBox().Bind(cities).SelectedIndex(0).IsEditable(false).DropDownCssClass("multi-column")
```

Tag Helpers

Razor

```
<style>
    .multi-column
    {   display: flex;
        flex-wrap: wrap;
        max-width: 450px;
    }
</style>
@{ List<string> cities = ViewBag.Cities;}
   <c1-combo-box selected-index="0" is-editable="false" drop-down-css-class="multi-column">
    <c1-items-source source-collection="cities"></c1-items-source>
</c1-combo-box>
```

MultiRow

The **MultiRow** control provides a powerful and flexible way of using multiple rows to represent each data item in tabular format. MultiRow control allows the users to view and edit data in a tabular form, just like any other conventional grids. But, MultiRow is different from these grids in a way that it allows you to bind each data item to multiple rows, creating form-like interfaces that can display a large number of columns with minimal horizontal scrolling.

	ID	Ordered	Customer	Customer Email	
	Amount	Shipped	Address	City	State
	0	7/2/2010	Joe Johnson	Joe.Johnson@gmail.com	
	$2,740.00	7/6/2010	3709 Johnson St.	Sidney	RS
	1	6/14/2013	Bill Peters	Bill.Peters@gmail.com	
	$4,678.00	6/18/2013	5546 Smith St.	Paris	RT
	2	8/20/2012	Aaron Johnson	Aaron.Johnson@gmail.com	
	$2,270.00	8/23/2012	1367 Adams St.	Cairo	RS
	3	5/27/2012	Chris Bannon	Chris.Bannon@gmail.com	
	$4,821.00	5/30/2012	4841 Richards St.	Florence	SP
	4	5/11/2012	Paul Smith	Paul.Smith@gmail.com	
	$204.00	5/15/2012	9630 Johnson St.	Sidney	SP
	5	5/21/2011	Chris Peters	Chris.Peters@gmail.com	
	$1,856.00	5/22/2011	3865 Wong St.	Cairo	RS
	6	12/2/2015	Tony Brown	Tony.Brown@gmail.com	
	$2,895.00	12/6/2015	2621 White St.	Cairo	SC
	7	9/4/2015	Joe Richards	Joe.Richards@gmail.com	
	$3,051.00	9/7/2015	4471 Peters St.	York	SC

To work with MultiRow control, you need to create an instance of **MultiRow** class, which is a part of **C1.Web.Mvc.MultiRow** namespace. MultiRow class extends from the **FlexGrid** class. Users already working with FlexGrid control, it will be easier for them to understand and implement MultiRow control. For more information, see QuickStart: Add Data to MultiRow.

Key Features

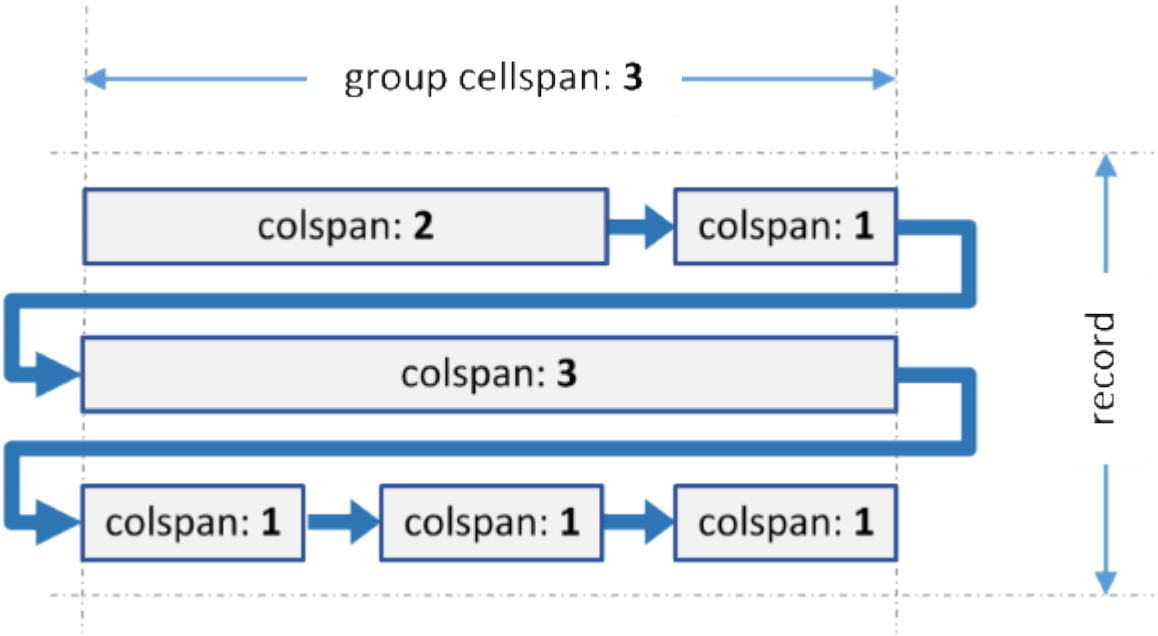
- **Multiple Layout Definition:** MultiRow allows you to set the LayoutDefinition property, which takes an object that describes the layout of the grid rows and cells. The LayoutDefinition property specifies the layout of the cells in the MultiRow control.
- **Fully Expanded Column Headers:** MultiRow allows you to set the CollapsedHeaders property to true which then enables the grid to show a single row of column headers containing the group names. Setting it to false cause it to show a group of rows with the same layout as the data, and column names in the cells.
- **Collapsible Column Headers**: MultiRow allows you to set the CollapsedHeaders property to true which then allows you to collapse the column headers to a single line, showing only the group names rather than individual cells. This saves space at the expense of having individual cell headers.
- **FlexGrid Features**: MultiRow extends from the FlexGrid class. It is a full-featured grid just like the FlexGrid control, providing various features including several selection modes, sorting, filtering, row and column freezing, custom cells, data mapping and more.

Layout Definition

The main property in **MultiRow** control is LayoutDefinition, which takes an object that describes the layout of the grid rows and cells. The LayoutDefinition property specifies the layout of the cells in the grid. It contains an array of cell group objects. Each cell group specifies how many columns the group should span, and the cells that make up each group.

The LayoutDefinition property contains an array of cell group objects with the following properties.

- **header** - Group Name
- **colspan** - Number of columns spanned by the group
- **cells** - Array of objects representing the cells in the group. Cells are columns with an additional colspan property.



The group spans three grid columns and contains six cells with different spans. When generating the layout, the grid fits as many cells as possible in each row, and wraps to the next row when the group span is reached. The last cell in each row is automatically expanded to fill colspan of the group. The process is similar to wrapping of text to create a paragraph. The same process is applied to every group in the layoutDefinition object.

Quick Start: Add Data to Multi Row

The topic describes how to add MultiRow control to your MVC web application and add data to it using model binding.

This topic comprises following four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for MultiRow**
- **Step 3: Add a MultiRow Control**
- **Step 4: Build and Run the Project**

The following image shows MultiRow control with data after completing the steps:

	ID	Ordered	Customer	Customer Email	
	Amount	Shipped	Address	City	State
	0	7/2/2010	Joe Johnson	Joe.Johnson@gmail.com	
	$2,740.00	7/6/2010	3709 Johnson St.	Sidney	RS
	1	6/14/2013	Bill Peters	Bill.Peters@gmail.com	
	$4,678.00	6/18/2013	5546 Smith St.	Paris	RT
	2	8/20/2012	Aaron Johnson	Aaron.Johnson@gmail.com	
	$2,270.00	8/23/2012	1367 Adams St.	Cairo	RS
	3	5/27/2012	Chris Bannon	Chris.Bannon@gmail.com	
	$4,821.00	5/30/2012	4841 Richards St.	Florence	SP
	4	5/11/2012	Paul Smith	Paul.Smith@gmail.com	
	$204.00	5/15/2012	9630 Johnson St.	Sidney	SP
	5	5/21/2011	Chris Peters	Chris.Peters@gmail.com	
	$1,856.00	5/22/2011	3865 Wong St.	Cairo	RS
	6	12/2/2015	Tony Brown	Tony.Brown@gmail.com	
	$2,895.00	12/6/2015	2621 White St.	Cairo	SC
	7	9/4/2015	Joe Richards	Joe.Richards@gmail.com	
	$3,051.00	9/7/2015	4471 Peters St.	York	SC

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for MultiRow

1. Add a new class to the **Models** folder (for example: `Orders.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `Orders.cs` model. We are using **`Orders`** class to represent sales order data in the database. Each instance of **Orders** object will correspond to a record on MultiRow control.

C#

Orders.cs copyCode

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace MultiRow.Models
{
    public class Orders
    {
        private static object _lockObj = new object();
        private static Random Rand = new Random();
        private static IList<Order> _orders;
        private static IList<string> _cities;
```

```
        private static IList<Customer> _customers;
        public class Customer
        {
            public int Id { get; set; }
            public string Name { get; set; }
            public string Address { get; set; }
            public string City { get; set; }
            public string State { get; set; }
            public string Email { get; set; }
        }
        public class Order
        {
            public int Id { get; set; }
            public DateTime Date { get; set; }
            public DateTime ShippedDate { get; set; }
            public double Amount { get; set; }
            public Customer Customer { get; set; }
        }

        public static IList<Order> GetOrders()
        {
               if (_orders == null)
                    {
                        var today = DateTime.Now.Date;
                        var customers = GetCustomers();
                        _orders = new List<Order>();
                        for (int i = 0; i < 8; i++)
                        {
                            var shipped = today.AddDays(-Rand.Next(-1, 3000));
                            var order = new Order
                            {
                                Id = i,
                                Date = shipped.AddDays(-Rand.Next(1, 5)),
                                ShippedDate = shipped,
                                Amount = Rand.Next(10000, 500000) / 100,
                                Customer = customers[Rand.Next(0, customers.Count - 1)],
                            };
                            _orders.Add(order);       }
            }
            return _orders;
        }
        public static IList<Customer> GetCustomers()
        {
            if (_customers == null)
                    {
                        var firstNames = new[] { "Aaron", "Paul", "John", "Mark", "Sue",
"Tom", "Bill", "Joe", "Tony", "Brad", "Frank", "Chris", "Pat" };
                        var lastNames = new[] { "Smith", "Johnson", "Richards",
"Bannon", "Wong", "Peters", "White", "Brown", "Adams", "Jennings" };
                        var cities = GetCities();
                        var states = new[] { "SP", "RS", "RN", "SC", "CS", "RT", "BC" };

                        _customers = new List<Customer>();
```

```
                        for (int i = 0; i < 50; i++)
                        {
                            var first = firstNames[Rand.Next(0, firstNames.Length - 1)];
                            var last = lastNames[Rand.Next(0, lastNames.Length - 1)];
                            var customer = new Customer
                            {
                                Id = i,
                                Name = first + " " + last,
                                Address = Rand.Next(100, 10000) + " " +
lastNames[Rand.Next(0, lastNames.Length - 1)] + " St.",
                                City = cities[Rand.Next(0, cities.Count - 1)],
                                State = states[Rand.Next(0, states.Length - 1)],
                                Email = first + "." + last + "@gmail.com",
                                                                    };
                            _customers.Add(customer);
                        }
            }
            return _customers;
        }
        public static IList<string> GetCities()
        {
          {
                if (_cities == null)
                    {
                        _cities = new[] { "York", "Paris", "Rome", "Cairo", "Florence",
"Sidney", "Hamburg", "Vancouver" };
                    }
                }
            return _cities;
        }     }
}
```

Back to Top

Step 3: Add a MultiRow Control

To add a MultiRow control to the application, follow these steps:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (for example: `MultiRowController`).
    3. Click **Add**.
4. Include the following MVC references as shown below.

    C#

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

5. Replace the method **Index()** with the following method.

C#

MultiRowController.cs

```
public ActionResult Index()
        {
            var model = Orders.GetOrders();
            return View(model);
        }
```

Add a View for the Controller

In the view, we create an instance of MultiRow control and bind it to a data source using **.Bind** property. The layout definition property helps us to define the column and row layout of the control.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `MultiRowController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml copyCode

```
@using MultiRow.Models
@model IEnumerable<Orders.Order>

<br />
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
                cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
                .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
                .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
                .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
                .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
                .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
                .Add(cell =>
```

```
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
                .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
}))
```

Tag Helpers

Index.cshtml copyCode

```
@using MultiRowNetCore.Models
@model IEnumerable<Orders.Order>

<c1-multi-row id="ovMultiRowCompact" class="multirow">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c"
class="amount" />
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity"
header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**MultiRow/Index**) in the address bar of the browser to see the view.

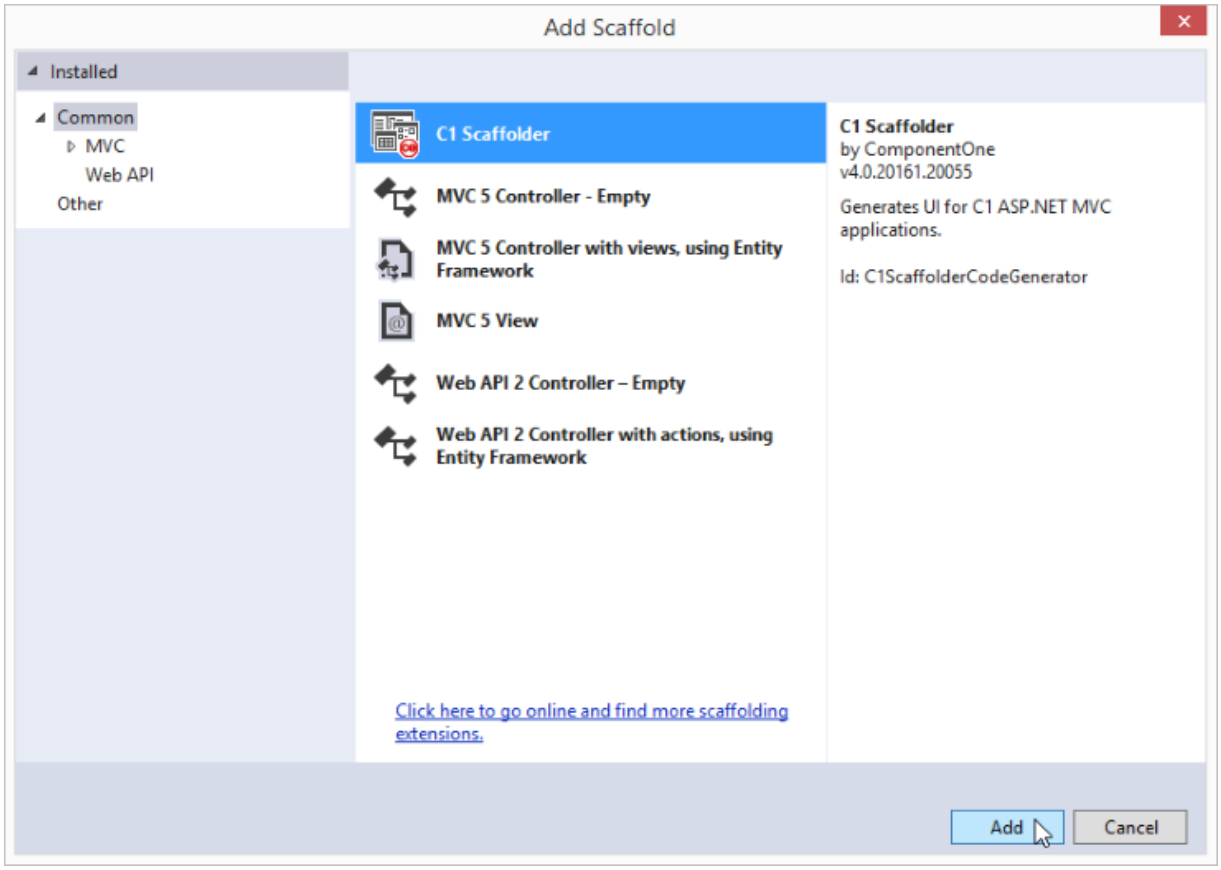
Back to Top

Features

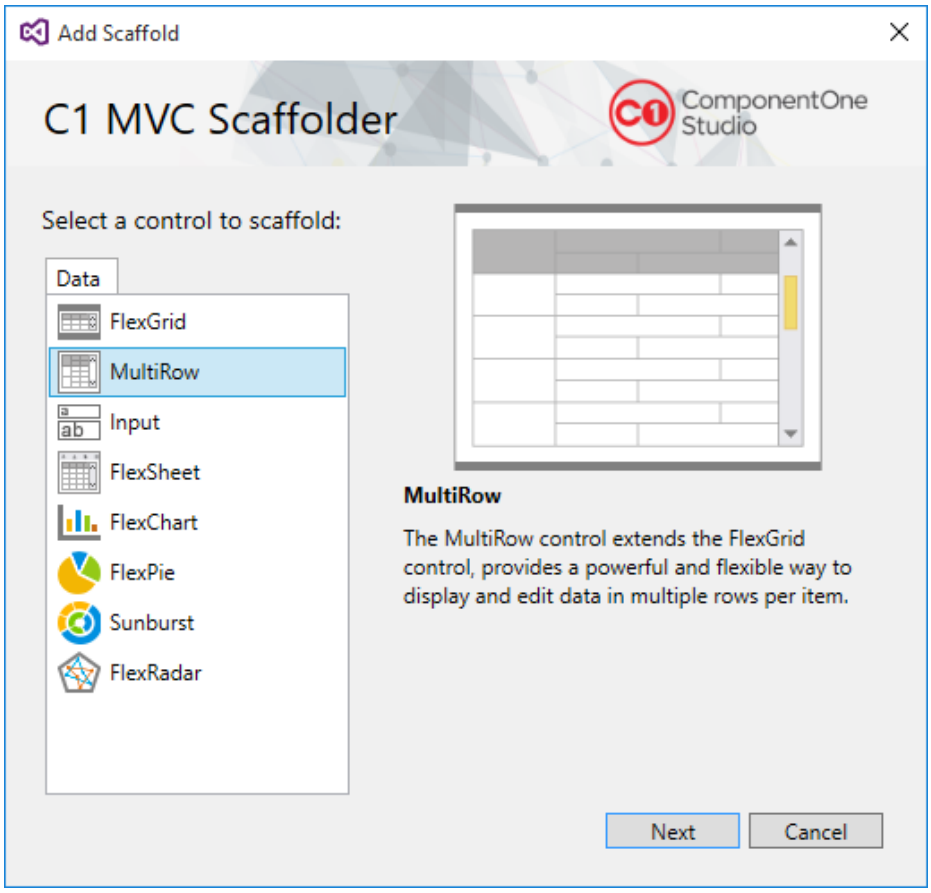
Scaffolding

The steps to scaffold **ComponentOne MultiRow** control for ASP.NET MVC are as follows:

1. Configure a datasource in your application. For more information on configuring a datasource, see Data Source Configuration topic.
2. In the **Solution Explorer**, right-click the project name and select **Add|New Scaffolded Item**. The Add Scaffold wizard appears.
3. In the **Add Scaffold** wizard, select **Common** and then select **C1 Scaffolder** from the right pane. You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.



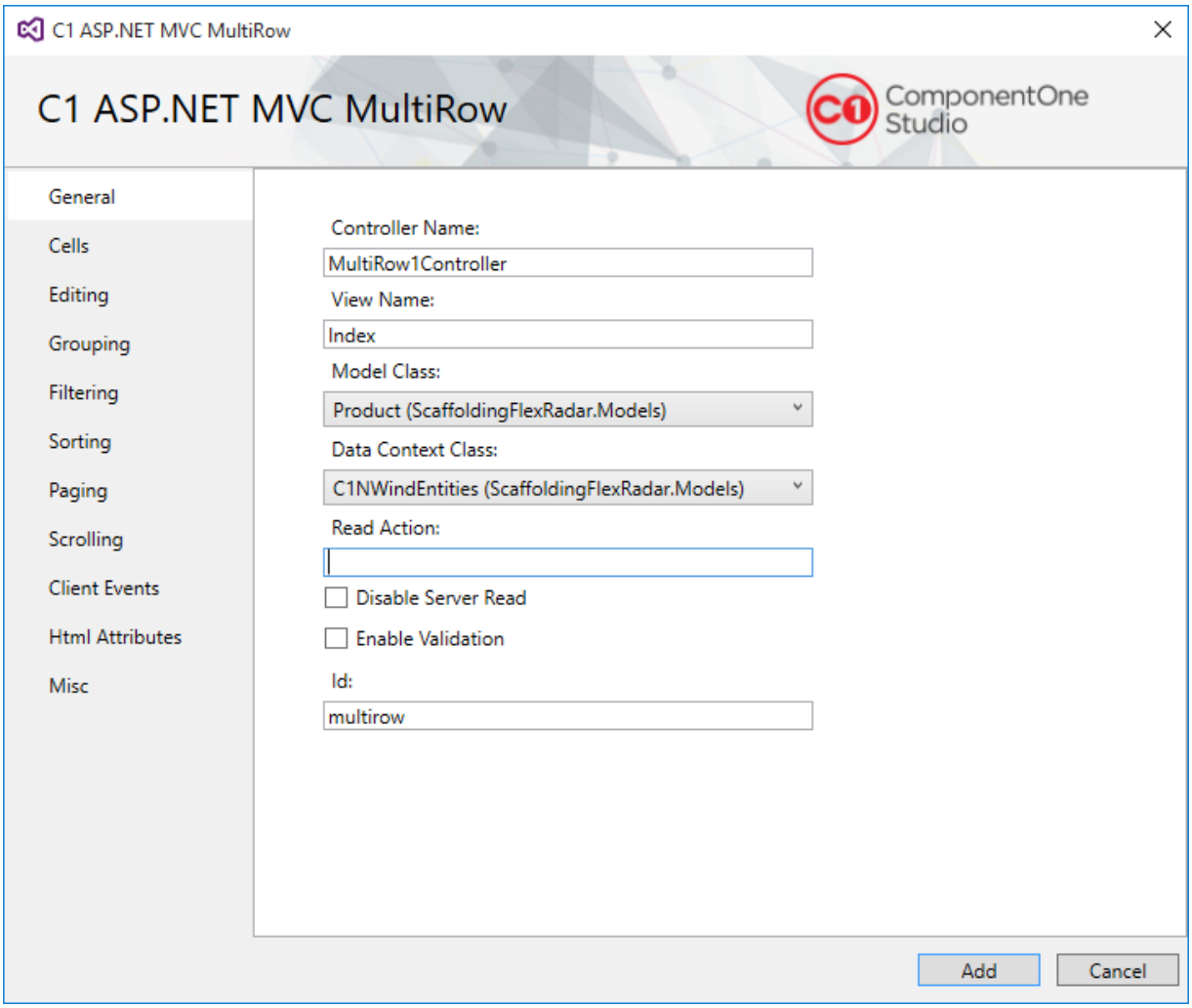
4. Click **Add**.
5. In the **Add Scaffold** dialog, select MultiRow control, and the click **Next**.



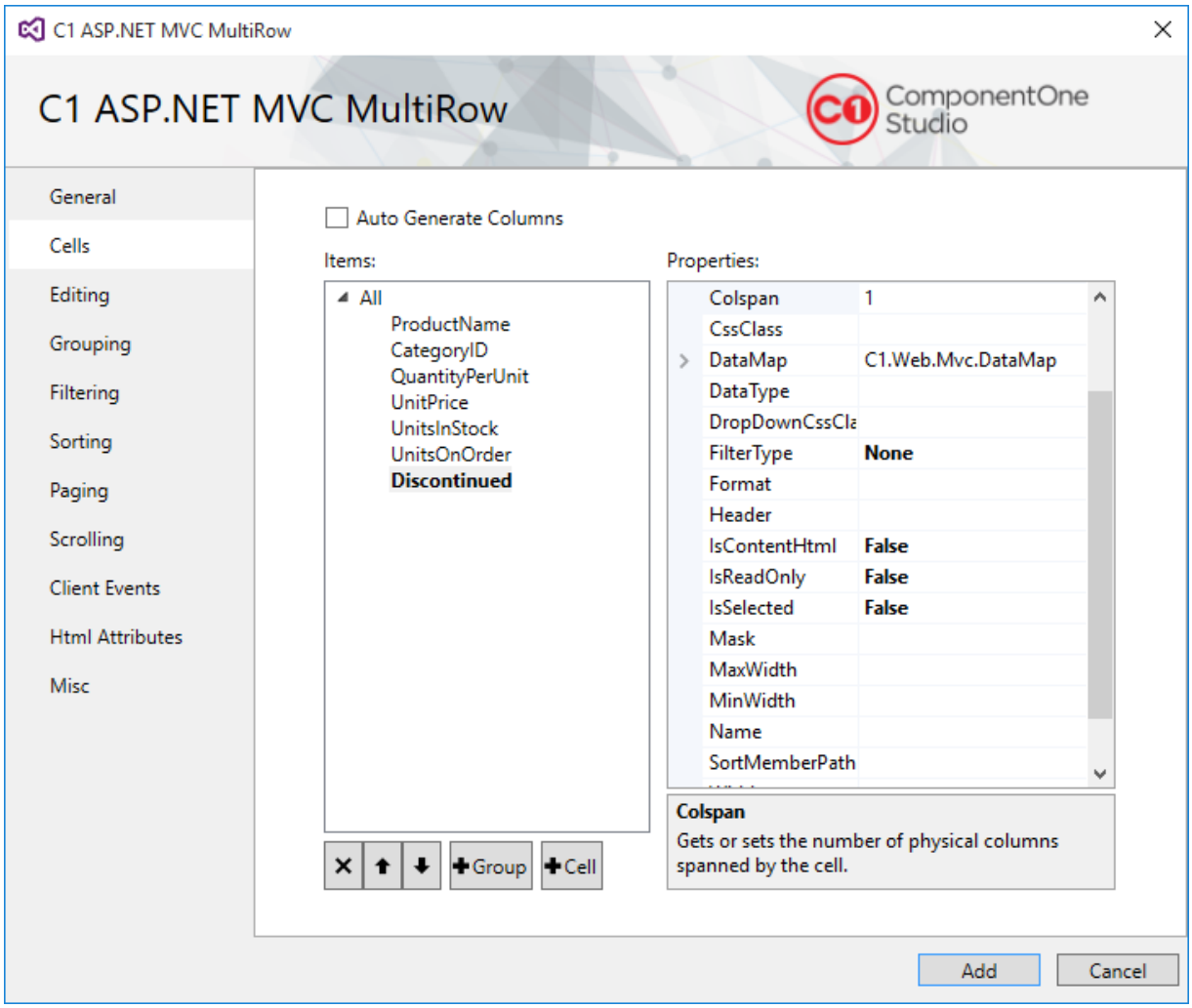
The **C1 ASP.NET MVC MultiRow** wizard appears with the General tab selected by default.

6. In the General tab, specify the model details as follows:

    1. Enter the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. The list shows all the available model types in the application in addition to the **C1NWind.edmx** model added in **Step 1**. In our case, we select **Product** to populate data in MultiRow.
    3. Select Data Context Class from the drop-down list. In our case, we select **C1NWindEntities.**



7. In the **Cells** tab on the left, un-check the Auto Generate Columns option to add, delete, or move columns upward or downward in the sequence according to your requirements. In our case, we have selected columns as shown in the image below.



8. In the **Editing** tab, check the **Allow Edit** and **Allow Delete** options.
9. In the **Grouping** tab, under the **Group Settings**, check **CategoryID** from Group Descriptions section.
10. In the **Filtering** tab, check the **Allow Filtering** option.
11. Click **Add**. You will notice that the Controller and View for the selected model are added to your project. Once the code is generated, you can run the project using the **F5** button.

Data Binding

To use MultiRow control and its features, you must bind data to your MultiRow. Once you bind the data to MultiRow control, you can use many different features such as filtering, sorting, selection mode, detail row and grouping.

This section contains information about

Remote Data Binding
: Learn how to remotely bind data in MultiRow control using C1JSONRequest.

Model Binding
: Learn how to add data to MultiRow control using basic Model binding.

Remote Binding

The MultiRow control allows you to retrieve data directly using **C1JSONRequest**. This specifies remote data

URLs, which include the server, table and columns. The arrays returned are used as data sources for **CollectionView** objects.

`CollectionViewHelper` is a static class that enables collections to have editing, filtering, grouping, and sorting services. This class also includes the following methods:

- `Read()`: Retrieves data from the collection.
- `Edit()`: Enables excel-style editing in MultiRow.
- `BatchEdit()`: Allows editing multiple items at a time.

The **`Bind`** property in MultiRow is used to bind it to a collection by passing an action URL method to carry out a specific operation.

This topic demonstrates how to retrieve data from an existing data source remotely. This is useful for developing data-intensive applications and scenarios for representing data as dashboards. The following image shows how MultiRow control appears after making C1JSON request to fetch data from a model. This example uses the sample created in the Quick Start topic.

	ID	Ordered	Customer	Customer Email	
	Amount	Shipped	Address	City	State
	0	7/2/2010	Joe Johnson	Joe.Johnson@gmail.com	
	$2,740.00	7/6/2010	3709 Johnson St.	Sidney	RS
	1	6/14/2013	Bill Peters	Bill.Peters@gmail.com	
	$4,678.00	6/18/2013	5546 Smith St.	Paris	RT
	2	8/20/2012	Aaron Johnson	Aaron.Johnson@gmail.com	
	$2,270.00	8/23/2012	1367 Adams St.	Cairo	RS
	3	5/27/2012	Chris Bannon	Chris.Bannon@gmail.com	
	$4,821.00	5/30/2012	4841 Richards St.	Florence	SP
	4	5/11/2012	Paul Smith	Paul.Smith@gmail.com	
	$204.00	5/15/2012	9630 Johnson St.	Sidney	SP
	5	5/21/2011	Chris Peters	Chris.Peters@gmail.com	
	$1,856.00	5/22/2011	3865 Wong St.	Cairo	RS
	6	12/2/2015	Tony Brown	Tony.Brown@gmail.com	
	$2,895.00	12/6/2015	2621 White St.	Cairo	SC
	7	9/4/2015	Joe Richards	Joe.Richards@gmail.com	
	$3,051.00	9/7/2015	4471 Peters St.	York	SC

In Code

Include the following MVC references in RemoteBindController.cs

C#

```
using <ApplicationName>.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serialization;
```

RemoteBindController.cs

C#

```
public partial class MultiRowController : Controller
 {
     // GET: MultiRow
     public ActionResult RemoteBind_Read([C1JsonRequest]
CollectionViewRequest<Orders.Order> requestData)
     {
     return this.C1Json(CollectionViewHelper.Read(requestData, Orders.GetOrders()));
     }
        public ActionResult Index()
     {
        return View();
     }
    }
```

RemoteBind.cshtml

HTML Helpers

Razor

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(Url.Action("RemoteBind_Read"))
    .AllowSorting(true)
    .IsReadOnly(true)
    .CssClass("multirow")
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
        cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
        .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
        .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
        .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
        .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
        .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
```

```
        .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
        .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
    }))
```

Tag Helpers

HTML

```
<c1-multi-row id="ovMultiRowCompact" is-read-only="true" class="multirow">
<c1-items-source read-action-url="@Url.Action("RemoteBind_Read")"></c1-items-source>
<c1-multi-row-cell-group header="Order" colspan="2">
<c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
<c1-multi-row-cell binding="Date" header="Ordered" width="150" />
<c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount" />
<c1-multi-row-cell binding="ShippedDate" header="Shipped" />
</c1-multi-row-cell-group>
<c1-multi-row-cell-group header="Customer" colspan="3">
<c1-multi-row-cell binding="Customer.Name" name="CustomerName" header="Cutomer"
width="200" />
<c1-multi-row-cell binding="Customer.Email" name="CustomerEmail" header="Customer
Email" class="email" colspan="2" />
<c1-multi-row-cell binding="Customer.Address" name="CustomerAddress" header="Address"
/>
<c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
</c1-multi-row-cell>
<c1-multi-row-cell binding="Customer.State" name="CustomerState" header="State" />
</c1-multi-row-cell-group>
</c1-multi-row>
```

Model Binding

This topic describes how to add a MultiRow control to your MVC application and populate data in it. The below example demonstrates how to achieve local model binding in MultiRow control. You can also perform remote data binding in MultiRow control, for more information, see Remote Data Binding.

This topic comprises following three steps:

- **Step 1: Create a Datasource for MultiRow**
- **Step 2: Add a MultiRow Control**
- **Step 3: Build and Run the Project**

The following image shows how MultiRow control appears after completing the steps:

	ID	Ordered	Customer	Customer Email	
	Amount	Shipped	Address	City	State
	0	7/2/2010	Joe Johnson	Joe.Johnson@gmail.com	
	$2,740.00	7/6/2010	3709 Johnson St.	Sidney	RS
	1	6/14/2013	Bill Peters	Bill.Peters@gmail.com	
	$4,678.00	6/18/2013	5546 Smith St.	Paris	RT
	2	8/20/2012	Aaron Johnson	Aaron.Johnson@gmail.com	
	$2,270.00	8/23/2012	1367 Adams St.	Cairo	RS
	3	5/27/2012	Chris Bannon	Chris.Bannon@gmail.com	
	$4,821.00	5/30/2012	4841 Richards St.	Florence	SP
	4	5/11/2012	Paul Smith	Paul.Smith@gmail.com	
	$204.00	5/15/2012	9630 Johnson St.	Sidney	SP
	5	5/21/2011	Chris Peters	Chris.Peters@gmail.com	
	$1,856.00	5/22/2011	3865 Wong St.	Cairo	RS
	6	12/2/2015	Tony Brown	Tony.Brown@gmail.com	
	$2,895.00	12/6/2015	2621 White St.	Cairo	SC
	7	9/4/2015	Joe Richards	Joe.Richards@gmail.com	
	$3,051.00	9/7/2015	4471 Peters St.	York	SC

Step 1: Create a Datasource for MultiRow

1. Add a new class to the **Models** folder (for example: `Orders.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `Orders.cs` model. We are using **`Orders`** class to represent sales order data in the database. Each instance of **Orders** object will correspond to a record in MultiRow control.

C#

Orders.cs copyCode

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace MultiRow.Models
{
    public class Orders
    {
        private static object _lockObj = new object();
        private static Random Rand = new Random();
        private static IList<Order> _orders;
        private static IList<string> _cities;
        private static IList<Customer> _customers;
        public class Customer
        {
            public int Id { get; set; }
```

```
            public string Name { get; set; }
            public string Address { get; set; }
            public string City { get; set; }
            public string State { get; set; }
            public string Email { get; set; }
        }
        public class Order
        {
            public int Id { get; set; }
            public DateTime Date { get; set; }
            public DateTime ShippedDate { get; set; }
            public double Amount { get; set; }
            public Customer Customer { get; set; }
        }

        public static IList<Order> GetOrders()
        {
               if (_orders == null)
                    {
                        var today = DateTime.Now.Date;
                        var customers = GetCustomers();
                        _orders = new List<Order>();
                        for (int i = 0; i < 8; i++)
                        {
                            var shipped = today.AddDays(-Rand.Next(-1, 3000));
                            var order = new Order
                            {
                                Id = i,
                                Date = shipped.AddDays(-Rand.Next(1, 5)),
                                ShippedDate = shipped,
                                Amount = Rand.Next(10000, 500000) / 100,
                                Customer = customers[Rand.Next(0, customers.Count - 1)],
                            };
                            _orders.Add(order);       }
            }
            return _orders;
        }
        public static IList<Customer> GetCustomers()
        {
            if (_customers == null)
                    {
                        var firstNames = new[] { "Aaron", "Paul", "John", "Mark", "Sue",
"Tom", "Bill", "Joe", "Tony", "Brad", "Frank", "Chris", "Pat" };
                        var lastNames = new[] { "Smith", "Johnson", "Richards",
"Bannon", "Wong", "Peters", "White", "Brown", "Adams", "Jennings" };
                        var cities = GetCities();
                        var states = new[] { "SP", "RS", "RN", "SC", "CS", "RT", "BC" };

                        _customers = new List<Customer>();
                        for (int i = 0; i < 50; i++)
                        {
                            var first = firstNames[Rand.Next(0, firstNames.Length - 1)];
                            var last = lastNames[Rand.Next(0, lastNames.Length - 1)];
```

```
                            var customer = new Customer
                            {
                                Id = i,
                                Name = first + " " + last,
                                Address = Rand.Next(100, 10000) + " " +
lastNames[Rand.Next(0, lastNames.Length - 1)] + " St.",
                                City = cities[Rand.Next(0, cities.Count - 1)],
                                State = states[Rand.Next(0, states.Length - 1)],
                                Email = first + "." + last + "@gmail.com",
                                                                    };
                            _customers.Add(customer);
                        }
            }
            return _customers;
        }
        public static IList<string> GetCities()
        {
          {
                if (_cities == null)
                    {
                        _cities = new[] { "York", "Paris", "Rome", "Cairo", "Florence",
"Sidney", "Hamburg", "Vancouver" };
                    }
                }
            return _cities;
        }     }
}
```

Back to Top

Step 2: Add a MultiRow Control

To add a MultiRow control to the application, follow these steps:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (for example: `MultiRowController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

    C#

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

5. Replace the method **Index()** with the following method.

    C#

MultiRowController.cs

```
public ActionResult Index()
        {
            var model = Orders.GetOrders();
            return View(model);
        }
```

Add a View for the Controller

In the view, we create an instance of MultiRow control and bind it to a data source using **.Bind** property. The layout definition property helps us to define the column and row layout of the control.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `MultiRowController.`
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller. Copy the following code and paste it inside **Index.cshtml.**

HTML Helpers

Index.cshtml copyCode

```
@using MultiRow.Models
@model IEnumerable<Orders.Order>

<br />
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
                cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
                .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
                .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
                .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
                .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
                .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
                .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
                .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
```

```
}))
```

Tag Helpers

Index.cshtml copyCode

```
@using MultiRowNetCore.Models
@model IEnumerable<Orders.Order>

<c1-multi-row id="ovMultiRowCompact" class="multirow">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c"
class="amount" />
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity"
header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**MultiRow/Index**) in the address bar of the browser to see the view.

Back to Top

Export

Excel Export

MultiRow control allows you to export its data to an Excel file (.xlsx) format. To export the MultiRow content in Excel format, you need to use the **FlexGridXlsxConverter.save** method. This method generates xlsx file content, which can be saved to your file system or shared on a server.

The following image shows how to export the MultiRow content in Excel format.

	ID	Start	Country		Amount	Amount2
	Active	End	Product	Color		Discount
	◢ Product: **Gadget** (5 items)					
	◢ Country: **German** (3 items)					
	1	1/25/2017	German		581.61	-2,939.67
	☑	1/25/2017	Gadget	Green		0.14
	4	4/25/2017	German		1,248.66	-2,815.93
	☐	4/25/2017	Gadget	Red		0.22
	5	5/25/2017	German		4,051.76	-3,108.76
	☑	5/25/2017	Gadget	Black		0.12
	◢ Country: **Canada** (1 items)					
	2	2/25/2017	Canada		4,919.02	-4,673.75
	☐	2/25/2017	Gadget	Green		0.17
	◢ Country: **Japan** (1 items)					
	3	3/25/2017	Japan		2,159.73	-3,810.42
	☐	3/25/2017	Gadget	Red		0.07

Export

View > Shared > _Layout.cshtml

Before implementing the code below, you need to add the `jszip.min.js` JavaScript library in your application to export MultiRow content to an Excel xlsx file.

HTML

```
<!-- SheetJS library -->
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.2.1/jszip.min.js">
</script>
```

Model - Sale.cs

We are using **`Sale`** class to represent sales order data in the database. Each instance of **Sale** object will correspond to a record on MultiRow control.

C#

```
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 int Rank { get; set; }
        private static List<string> COUNTRIES = new List<string> { "US", "UK",
"Canada", "Japan", "China", "France", "German", "Italy", "Korea", "Australia" };
        private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget",
"Doohickey" };
        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 startDate = new DateTime(dt.Year, i % 12 + 1, 25);
                var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60, i
% 60);
                return new Sale
                {
                    ID = i + 1,
                    Start = startDate,
                    End = endDate,
                    Country = country,
                    Product = product,
                    Color = color,
                    Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Discount = Math.Round(rand.NextDouble() / 4, 2),
                    Active = (i % 4 == 0),
                    Rank = rand.Next(1, 6)
                };
            });
            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;
        }
    }
```

Controller - ExcelExportController.cs

C#

```
public ActionResult Index()
{
         return View(Sale.GetData(5));
}
```

View - Index.cshtml

In the view, we create an instance of MultiRow control and add an Export button. The layout definition property helps us to define the column and row layout of the control. The JavaScript will help to export the MultiRow control to Excel file using FlexGridXlsxConverter.

HTML Helpers

Razor

```
@model IEnumerable<Sale>
@using MultiRowExport.Models;
@section Scripts{
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
<script>
        var multiRow, colHeaderCheckBox;
        c1.documentReady(function () {
            multiRow = wijmo.Control.getControl("#excelExportMultiRow");
        });

        function exportXlsx() {
            if (multiRow) {
                wijmo.grid.xlsx.FlexGridXlsxConverter.save(multiRow, {
includeCellStyles: false, includeColumnHeaders: true }, 'MultiRow.xlsx');
            }
        }
</script>}
@(Html.C1().MultiRow<Sale>().Id("excelExportMultiRow")
    .Bind(Model)
    .ShowGroups(true)
    .GroupBy("Product", "Country")
    .IsReadOnly(true)
    .CssClass("multirow")
    .LayoutDefinition(ld =>
        {
              ld.Add().Cells(cells =>
              {
              cells.Add(cell => cell.Binding("ID").Header("ID"));
              cells.Add(cell => cell.Binding("Active").Header("Active"));
              });
              ld.Add().Cells(cells =>
              {
              cells.Add(cell => cell.Binding("Start").Header("Start"));
```

```
                cells.Add(cell => cell.Binding("End").Header("End"));
                });
                ld.Add().Colspan(2).Cells(cells =>
                {
                cells.Add(cell =>
cell.Binding("Country").Header("Country").Colspan(2));
                cells.Add(cell => cell.Binding("Product").Header("Product"));
                cells.Add(cell => cell.Binding("Color").Header("Color"));
                });
                ld.Add().Colspan(2).Cells(cells =>
                {
                cells.Add(cell => cell.Binding("Amount").Header("Amount"));
                cells.Add(cell => cell.Binding("Amount2").Header("Amount2"));
                cells.Add(cell =>
cell.Binding("Discount").Header("Discount").Colspan(2));
        });
}))
<a download="MultiRow.xlsx" class="btn btn-default" id="exportBtn"
onclick="exportXlsx();">Export</a>
```

Tag Helpers

HTML

```
@model IEnumerable<Sale>
@using ExcelExportCore.Models

@section Scripts{
<script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
<script>
        var multiRow;
        c1.documentReady(function () {
            multiRow = wijmo.Control.getControl("#excelExportMultiRow");
         });
        function exportXlsx() {
            if (multiRow) {
                wijmo.grid.xlsx.FlexGridXlsxConverter.save(multiRow, {
includeCellStyles: false, includeColumnHeaders: true }, 'MultiRow.xlsx');
            }
        }
</script>
}
<c1-multi-row id="excelExportMultiRow" class="multirow" is-read-only="true"
                show-groups="true" group-by="Product,Country">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group>
        <c1-multi-row-cell binding="ID"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Active"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group>
```

```
        <c1-multi-row-cell binding="Start"></c1-multi-row-cell>
        <c1-multi-row-cell binding="End"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group colspan="2">
        <c1-multi-row-cell binding="Country" colspan="2"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Product"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Color"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group colspan="2">
        <c1-multi-row-cell binding="Amount"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Amount2"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Discount" colspan="2"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
</c1-multi-row>
<a download="MultiRow.xlsx" class="btn btn-default" id="exportBtn"
onclick="exportXlsx();">Export</a>
```

PDF Export

When using MultiRow control, you may need to export your data to PDF format in order to make it accessible for the peers or share the file on a server. The MultiRow control provides PDF export functionality on the client side which uses the **FlexGridPdfConverter**, a PDFKit-based JavaScript library, to export MultiRow to PDF (Portable Document Format) without using any server-side code. The MultiRow control uses the current column order, visibility, and dimensions to generate the PDF file output.

The following image shows how to export the MultiRow content in PDF file format.

	ID	Start	Country		Amount	Amount2
	Active	End	Product	Color		Discount
	◢ Product: **Gadget** (5 items)					
	◢ Country: **German** (3 items)					
	1	1/25/2017	German		581.61	-2,939.67
	☑	1/25/2017	Gadget	Green		0.14
	4	4/25/2017	German		1,248.66	-2,815.93
	☐	4/25/2017	Gadget	Red		0.22
	5	5/25/2017	German		4,051.76	-3,108.76
	☑	5/25/2017	Gadget	Black		0.12
	◢ Country: **Canada** (1 items)					
	2	2/25/2017	Canada		4,919.02	-4,673.75
	☐	2/25/2017	Gadget	Green		0.17
	◢ Country: **Japan** (1 items)					
	3	3/25/2017	Japan		2,159.73	-3,810.42
	☐	3/25/2017	Gadget	Red		0.07

Export

Model - Sale.cs

We are using **`Sale`** class to represent sales order data in the database. Each instance of **Sale** object will correspond to a record on MultiRow control.

C#

```
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 int Rank { get; set; }
        private static List<string> COUNTRIES = new List<string> { "US", "UK",
"Canada", "Japan", "China", "France", "German", "Italy", "Korea", "Australia" };
        private static List<string> PRODUCTS = new List<string> { "Widget", "Gadget",
"Doohickey" };
        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 startDate = new DateTime(dt.Year, i % 12 + 1, 25);
                var endDate = new DateTime(dt.Year, i % 12 + 1, 25, i % 24, i % 60, i
% 60);
                return new Sale
                {
                    ID = i + 1,
                    Start = startDate,
                    End = endDate,
                    Country = country,
                    Product = product,
                    Color = color,
                    Amount = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Amount2 = Math.Round(rand.NextDouble() * 10000 - 5000, 2),
                    Discount = Math.Round(rand.NextDouble() / 4, 2),
                    Active = (i % 4 == 0),
                    Rank = rand.Next(1, 6)
                };
            });
            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;
        }
    }
```

Controller - PDFExportController.cs

C#

```
public ActionResult Index()
{
         return View(Sale.GetData(5));
}
```

View - Index.cshtml

In the view, we create an instance of MultiRow control and add an Export button. The layout definition property helps us to define the column and row layout of the control. The JavaScript will help to export the MultiRow control to PDF file using FlexGridPdfConverter.

HTML Helpers

Razor

```
@model IEnumerable<Sale>
@using C1.Web.Mvc
@using MultiRowPdfExport.Models

@section Scripts{
    <script>
        var multiRow;
            c1.documentReady(function () {
            multiRow = wijmo.Control.getControl("#exportPdfMultiRow");
        });

        function exportPdf() {
           wijmo.grid.pdf.FlexGridPdfConverter.export(multiRow, 'MultiRow.pdf',
{maxPages: 10});
        }
    </script>
    }
<br />
@(Html.C1().MultiRow<Sale>().Id("exportPdfMultiRow")
    .Bind(Model)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.ListBox)
    .HeadersVisibility(C1.Web.Mvc.Grid.HeadersVisibility.All)
    .ShowGroups(true)
    .GroupBy("Product", "Country")
    .CssClass("multirow custom-multi-row")
    .LayoutDefinition(ld =>
    {
        ld.Add().Cells(cells =>
        {
            cells.Add(cell => cell.Binding("ID").Header("ID"));
            cells.Add(cell => cell.Binding("Active").Header("Active"));
        });
        ld.Add().Cells(cells =>
        {
            cells.Add(cell => cell.Binding("Start").Header("Start"));
            cells.Add(cell => cell.Binding("End").Header("End"));
        });
        ld.Add().Colspan(2).Cells(cells =>
        {
            cells.Add(cell => cell.Binding("Country").Header("Country").Colspan(2));
            cells.Add(cell => cell.Binding("Product").Header("Product"));
            cells.Add(cell => cell.Binding("Color").Header("Color"));
        });
```

```
        ld.Add().Colspan(2).Cells(cells =>
        {
            cells.Add(cell => cell.Binding("Amount").Header("Amount"));
            cells.Add(cell => cell.Binding("Amount2").Header("Amount2"));
            cells.Add(cell =>
cell.Binding("Discount").Header("Discount").Colspan(2));
        });
    }))

<button class="btn btn-default" onclick="exportPdf()">Export</button>
```

Tag Helpers

HTML

```
@model IEnumerable<Sale>
@using C1.Web.Mvc
@section Scripts{
    <script>
        var multiRow;
            c1.documentReady(function () {
            multiRow = wijmo.Control.getControl("#exportPdfMultiRow");
        });

        function exportPdf() {
           wijmo.grid.pdf.FlexGridPdfConverter.export(multiRow, 'MultiRow.pdf',
{maxPages: 10});
        }
    </script>}
<br />
<c1-multi-row id="exportPdfMultiRow" class="multirow custom-multi-row" is-read-
only="true"
               show-groups="true" group-by="Product,Country" selection-mode="ListBox">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group>
        <c1-multi-row-cell binding="ID"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Active"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group>
        <c1-multi-row-cell binding="Start"></c1-multi-row-cell>
        <c1-multi-row-cell binding="End"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group colspan="2">
        <c1-multi-row-cell binding="Country" colspan="2"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Product"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Color"></c1-multi-row-cell>
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group colspan="2">
        <c1-multi-row-cell binding="Amount"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Amount2"></c1-multi-row-cell>
        <c1-multi-row-cell binding="Discount" colspan="2"></c1-multi-row-cell>
```

```
    </c1-multi-row-cell-group>
</c1-multi-row>

<button class="btn btn-default" onclick="exportPdf()">Export</button>
```

Grouping

MultiRow supports column-wise grouping of grid data, using CollectionView class. You can configure group by using the GroupBy property in view. You can also group the data using JavaScript by adding GroupDescription objects to the GroupDescriptions property.

MultiRow also allows you to customize the text displayed in group header rows using the GroupHeaderFormat property. By default, it displays the name of the group, followed by the current group and the number of items in the group. To format the aggregated data in the group header for a particular column, you can set the format property on each Column object.

The following image shows the MultiRow control with data grouped based on state. This example uses the sample created in the Quick Start topic.

ID	Ordered	Customer	Customer Email	
Amount	Shipped	Address	City	State
◢ State: **RS** (2 items)				
0	11/5/2009	John Peters	John.Peters@gmail.com	
$2,229.00	11/8/2009	6352 Johnson St.	York	RS
4	9/26/2016	Frank Bannon	Frank.Bannon@gmail.com	
$3,977.00	9/30/2016	7217 Wong St.	Florence	RS
◢ State: **RN** (3 items)				
1	6/26/2010	John White	John.White@gmail.com	
$4,184.00	6/27/2010	144 Smith St.	Paris	RN
5	4/1/2010	Frank Peters	Frank.Peters@gmail.com	
$3,036.00	4/5/2010	3497 Adams St.	Paris	RN
7	2/5/2013	Brad Brown	Brad.Brown@gmail.com	
$955.00	2/7/2013	3513 Richards St.	York	RN
◢ State: **RT** (2 items)				
2	3/16/2009	Chris Smith	Chris.Smith@gmail.com	
$2,210.00	3/18/2009	9630 Richards St.	York	RT
6	9/19/2014	Mark White	Mark.White@gmail.com	
$507.00	9/22/2014	8988 Smith St.	York	RT

In Code

Grouping.cshtml

HTML Helpers

Grouping.cshtml

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .GroupBy("Customer.State")
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
             cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
             .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
             .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
             .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
            .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
            .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
            .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
            .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
}))
```

Tag Helpers

Grouping.cshtml

```
<c1-multi-row id="ovMultiRowCompact" class="multirow" group-by="Customer.State" show-
groups="true">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
```

```
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

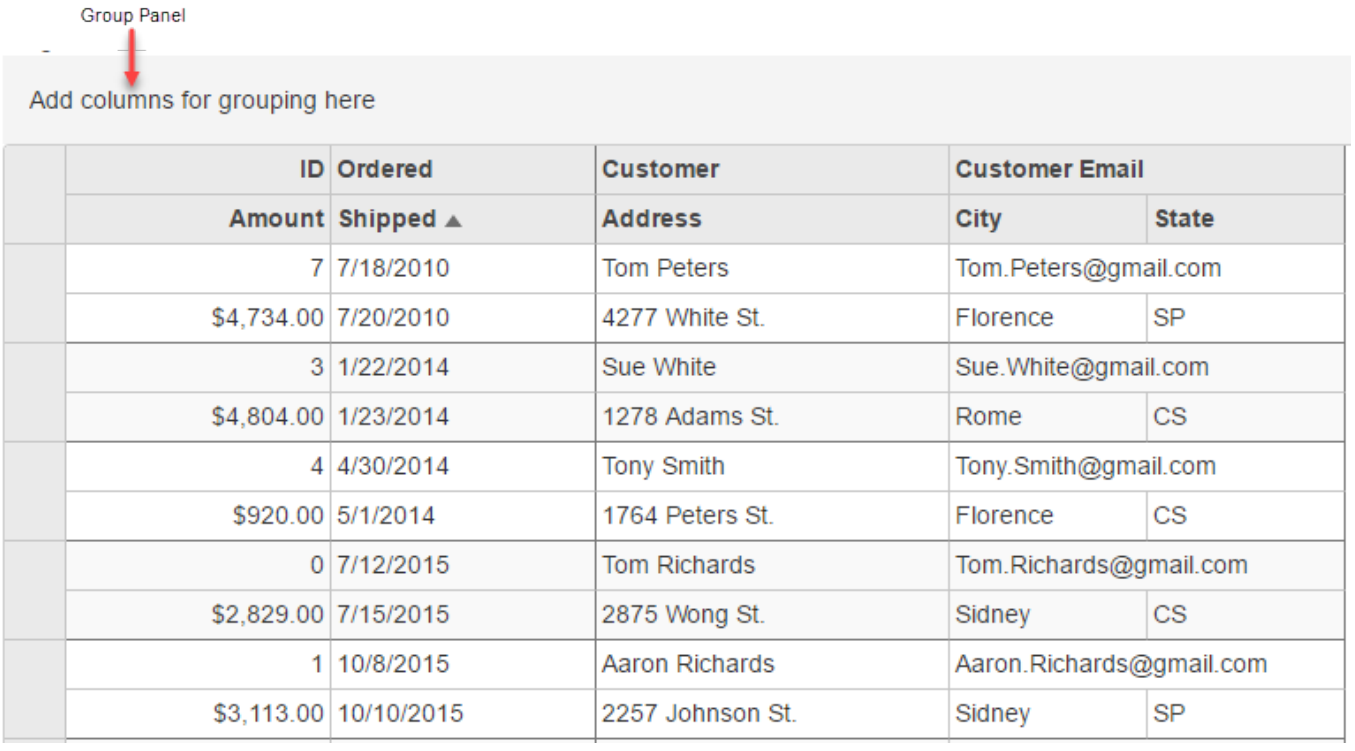
Group Panel

The MultiRow control provides the flexibility to group and ungroup grid data as per the requirement at run-time. Group panel enables you to drag and drop desired columns from the MultiRow control for grouping.

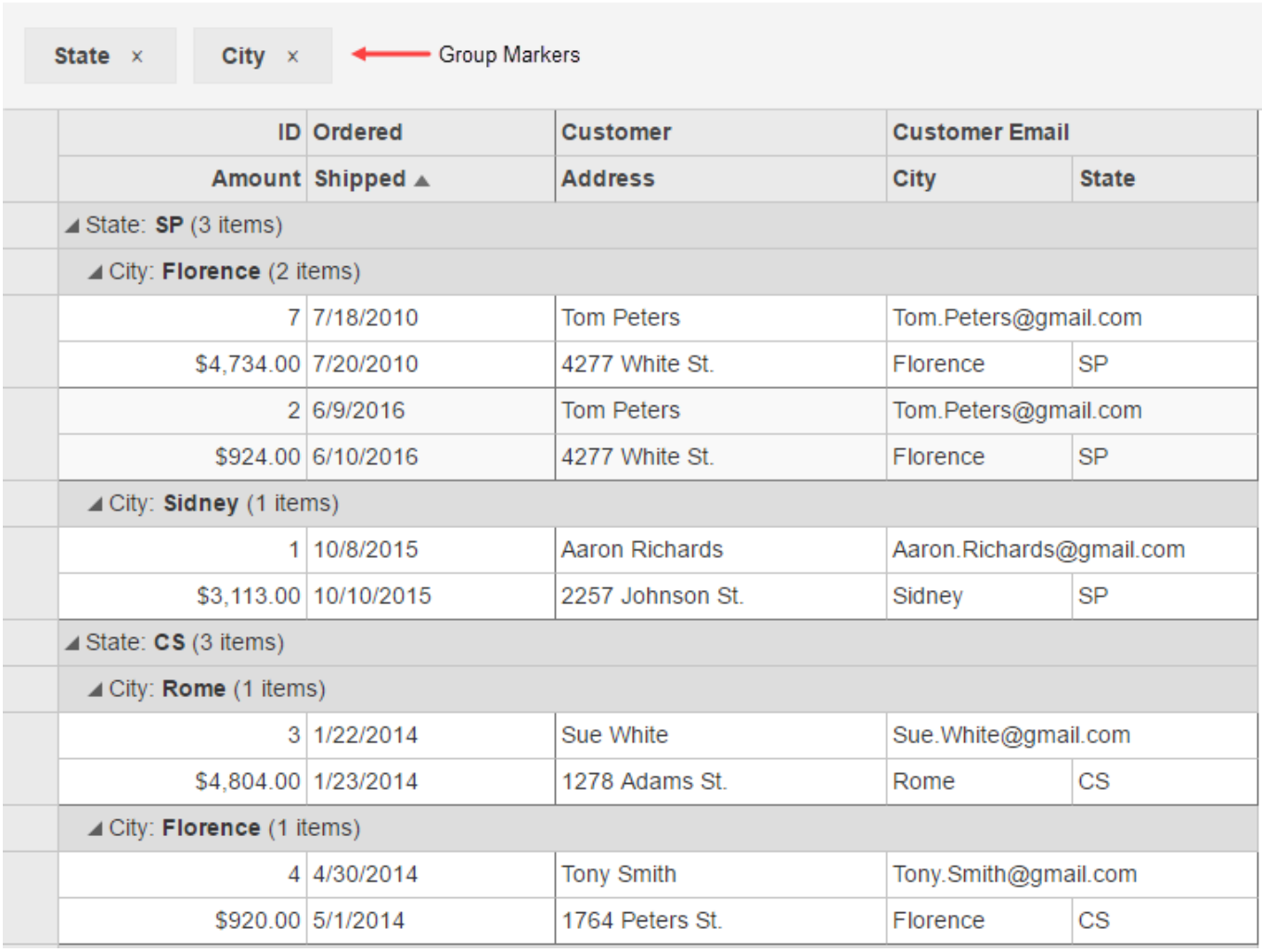
It is possible to move groups within the panel to change the grouping order of the grid data. Moreover, clicking a group marker in the panel enables sorting of the groups, based on the particular column entries.

To set the maximum number of groups to allow in group panel, set the MaxGroups property. Set the Placeholder property to the string you want to display in the panel when it contains no groups.

The following images display grouping columns using Group Panel in MultiRow. This example uses the sample created in the Quick Start topic.



	ID	Ordered	Customer	Customer Email	
	Amount	Shipped ▲	Address	City	State
	7	7/18/2010	Tom Peters	Tom.Peters@gmail.com	
	$4,734.00	7/20/2010	4277 White St.	Florence	SP
	3	1/22/2014	Sue White	Sue.White@gmail.com	
	$4,804.00	1/23/2014	1278 Adams St.	Rome	CS
	4	4/30/2014	Tony Smith	Tony.Smith@gmail.com	
	$920.00	5/1/2014	1764 Peters St.	Florence	CS
	0	7/12/2015	Tom Richards	Tom.Richards@gmail.com	
	$2,829.00	7/15/2015	2875 Wong St.	Sidney	CS
	1	10/8/2015	Aaron Richards	Aaron.Richards@gmail.com	
	$3,113.00	10/10/2015	2257 Johnson St.	Sidney	SP



ID	Ordered	Customer	Customer Email	
Amount	Shipped ▲	Address	City	State
State: **SP** (3 items)				
City: **Florence** (2 items)				
7	7/18/2010	Tom Peters	Tom.Peters@gmail.com	
$4,734.00	7/20/2010	4277 White St.	Florence	SP
2	6/9/2016	Tom Peters	Tom.Peters@gmail.com	
$924.00	6/10/2016	4277 White St.	Florence	SP
City: **Sidney** (1 items)				
1	10/8/2015	Aaron Richards	Aaron.Richards@gmail.com	
$3,113.00	10/10/2015	2257 Johnson St.	Sidney	SP
State: **CS** (3 items)				
City: **Rome** (1 items)				
3	1/22/2014	Sue White	Sue.White@gmail.com	
$4,804.00	1/23/2014	1278 Adams St.	Rome	CS
City: **Florence** (1 items)				
4	4/30/2014	Tony Smith	Tony.Smith@gmail.com	
$920.00	5/1/2014	1764 Peters St.	Florence	CS

In Code

HTML Helpers

GroupPanel.cshtml

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .ShowGroupPanel(s => s
        .MaxGroups(4)
        .Placeholder("Add columns for grouping here"))
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
            .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
            .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
            .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
```

```
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
            .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
            .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
            .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
            .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
}))
```

Tag Helpers

GroupPanel.cshtml

```
<c1-multi-row id="ovMultiRowCompact" class="multirow" >
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-grid-group-panel max-groups="4"
     placeholder="Add columns for grouping here">
    </c1-flex-grid-group-panel>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

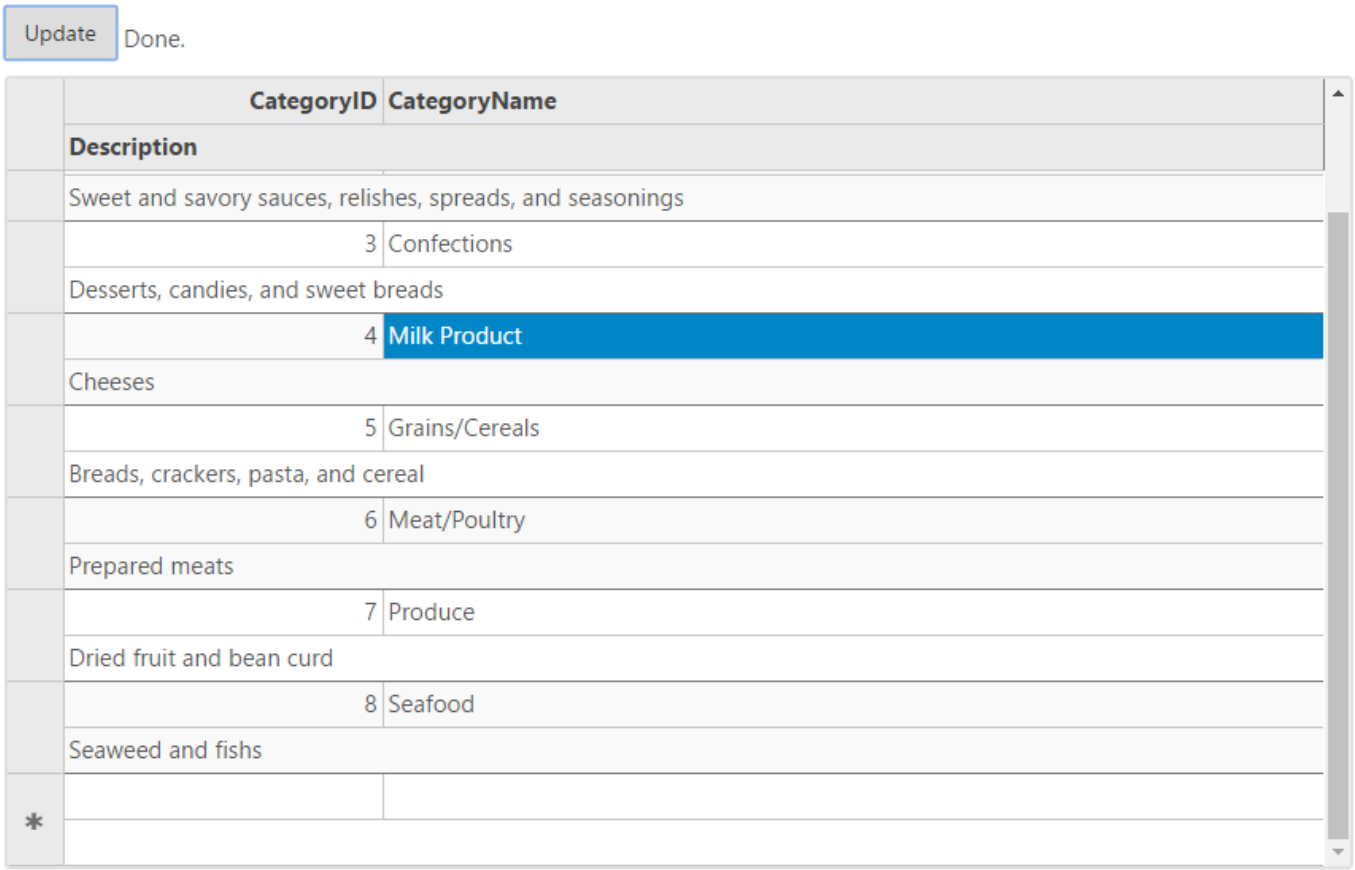
Batch Editing

The MultiRow control supports batch editing, which allows the user to update, create or remove multiple items, and commit data item to the database at run-time. The user can also perform multiple modifications on the MultiRow

control using sorting, paging or filtering. To access the MultiRow control in BatchEditing mode, you need to set the BatchEditActionUrl property.

> **Note**: To disable data update while performing sorting, filtering or page operations, set the DisableServerRead property of MultiRow ItemsSource to true. To submit the data, you can call collectionView commit method explicitly from client-side.

The following image shows the MultiRow control along with the **Update** button to commit the data using **BatchEditActionUrl** property. The example provides two ways to commit data to the server, one by pressing the Enter key after updating the content of a cell, and other by clicking the Update button.



InCode

BatchEditingController.cs

BatchEditingController.cs

```
// GET: BatchEdit
private C1NWindEntities db = new C1NWindEntities();
public ActionResult BatchEditing(CollectionViewRequest<Category> requestData)
        {
            return View(db.Categories.ToList());
        }

public ActionResult
MultiRowBatchEdit([C1JsonRequest]CollectionViewBatchEditRequest<Category>
requestData)
        {
            return this.C1Json(CollectionViewHelper.BatchEdit(requestData, batchData
```

```
=>
            {
                var itemresults = new List<CollectionViewItemResult<Category>>();
                string error = string.Empty;
                bool success = true;
                try
                {
                    if (batchData.ItemsCreated != null)
                    {
                        batchData.ItemsCreated.ToList().ForEach(st =>
                        {
                            db.Categories.Add(st);
                            itemresults.Add(new CollectionViewItemResult<Category>
                            {
                                Error = "",
                                Success = ModelState.IsValid,
                                Data = st
                            });
                        });
                    }
                    if (batchData.ItemsDeleted != null)
                    {
                        batchData.ItemsDeleted.ToList().ForEach(category =>
                        {
                            var fCategory = db.Categories.Find(category.CategoryID);
                            db.Categories.Remove(fCategory);
                            itemresults.Add(new CollectionViewItemResult<Category>
                            {
                                Error = "",
                                Success = ModelState.IsValid,
                                Data = category
                            });
                        });
                    }
                    if (batchData.ItemsUpdated != null)
                    {
                        batchData.ItemsUpdated.ToList().ForEach(category =>
                        {
                            db.Entry(category).State = EntityState.Modified;
                            itemresults.Add(new CollectionViewItemResult<Category>
                            {
                                Error = "",
                                Success = ModelState.IsValid,
                                Data = category
                            });
                        });
                    }
                    db.SaveChanges();
                }
                catch (DbEntityValidationException e)
                {
```

```
                    error = string.Join(",", e.EntityValidationErrors.SelectMany(i =>
i.ValidationErrors).Select(i => i.ErrorMessage));
                    success = false;
                }
                catch (Exception e)
                {
                    error = e.Message;
                    success = false;
                }

                return new CollectionViewResponse<Category>
                {
                    Error = error,
                    Success = success,
                    OperatedItemResults = itemresults
                };
            }, () => db.Categories.ToList()));
        }
```

BatchEditing.cshtml

HTML Helpers

BatchEditing.cshtml

```
@using MultiRow.Models
@model IEnumerable<Category>

<script type="text/javascript">
        function batchUpdate() {
            var batchEditMultiRow = wijmo.Control.getControl('#batchEditMultiRow'),
                cv = batchEditMultiRow.collectionView;
            cv.commit();
        }
</script>
}
<input type="button" value="Update" class="btn" onclick="batchUpdate()" />

@(Html.C1().MultiRow<Category>()
    .Id("batchEditMultiRow")
    .AllowAddNew(true)
    .AllowDelete(true)
    .LayoutDefinition(ld =>
    {
        ld.Add().Colspan(2).Cells(columns =>
        {
            columns.Add(c => c.Binding("CategoryID").IsReadOnly(true).Format("d"))
        .Add(c => c.Binding("CategoryName").Width("*"))
        .Add(c => c.Binding("Description"));
        });
```

```
    })
    .Bind(ib =>
ib.DisableServerRead(true).Bind(Model).BatchEdit(Url.Action("MultiRowBatchEdit"))
    ))
```

Tag Helpers

BatchEditing.cshtml

```
<script type="text/javascript">
        function batchUpdate() {
            var batchEditMultiRow = wijmo.Control.getControl('#batchEditMultiRow'),
                cv = batchEditMultiRow.collectionView;
            cv.commit();
        }
    </script>
}
<input type="button" value="Update" class="btn" onclick="batchUpdate()" />
<c1-multi-row id="batchEditMultiRow" allow-add-new="true" allow-delete="true"
class="multirow" >
    <c1-items-source disable-server-read="true" read-action-
url="@Url.Action("BatchEditing_Bind")"
                     batch-edit-action-url="@Url.Action("MultiRowBatchEdit")"></c1-
items-source>
    <c1-multi-row-cell-group colspan="2">
        <c1-multi-row-cell binding="CategoryID" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="CategoryName" width="150" />
        <c1-multi-row-cell binding="Description" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

Collapsible Column Headers

The MultiRow control can collapse the column headers to a single line, showing only the group names rather than individual cells. The MultiRow control, by default show column headers that span multiple rows and shows the header for each cell defined in the LayoutDefinition. This saves space at the expense of having individual cell headers.

The following image shows the MultiRow control displaying CollapsedHeaders. This example uses the sample created in the Quick Start topic.

	Order		Customer		
	0	4/5/2015	Aaron Adams	Aaron.Adams@gmail.com	
	$3,399.00	4/8/2015	2100 Richards St.	Paris	RT
	1	10/3/2011	Sue Bannon	Sue.Bannon@gmail.com	
	$4,055.00	10/4/2011	2718 Adams St.	Rome	RS
	2	9/9/2013	Bill Adams	Bill.Adams@gmail.com	
	$944.00	9/13/2013	1108 Adams St.	Cairo	SC
	3	1/24/2013	Frank Wong	Frank.Wong@gmail.com	
	$3,167.00	1/28/2013	177 Wong St.	Florence	CS
	4	2/7/2012	John Johnson	John.Johnson@gmail.com	
	$1,300.00	2/8/2012	7787 Johnson St.	York	SP

In Code

CollapsedHeaders.cshtml

To collapse the column headers, set the CollapsedHeaders property to true. In these scenarios, remember to set the Header property on the groups to avoid empty column headers.

HTML Helpers

CollapsedHeaders.cshtml

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .CollapsedHeaders(true)
    .ShowHeaderCollapseButton(true)
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
            .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
            .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
            .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
            .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
            .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
            .Add(cell =>
```

```
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
            .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
}))
```

Tag Helpers

CollapsedHeaders.cshtml

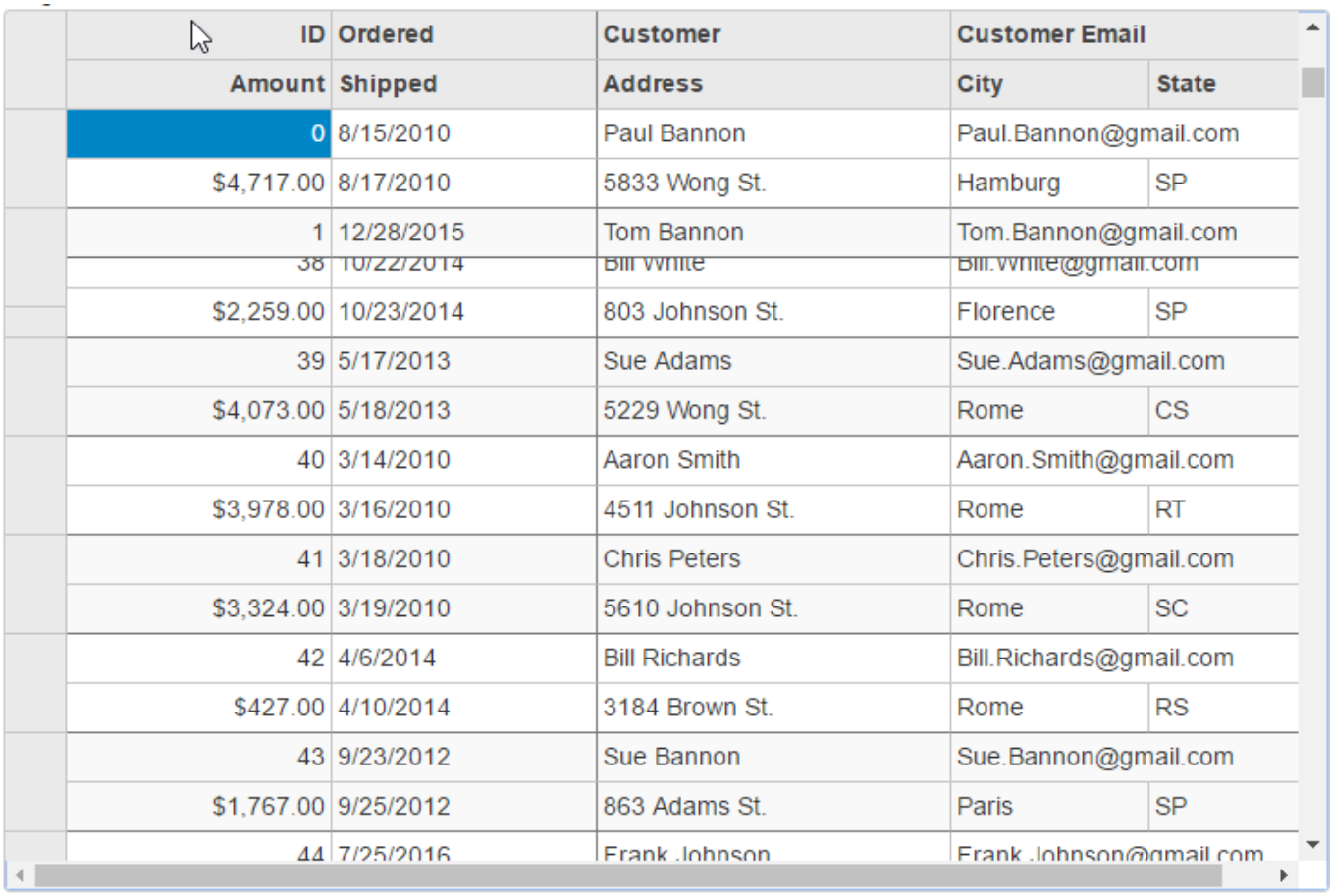
```
<c1-multi-row id="ovMultiRowCompact" class="multirow" collapsed-headers="true" show-
header-collapse-button="true">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group header="Order" colspan="2">
    <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

Row and Column Freezing

The MultiRow control allows you to freeze rows and columns so they remain in view as the user scrolls the grid. Row and column freezing is used to keep an area of a MultiRow visible while you scroll to another area of the worksheet. Frozen cells can be edited and selected as regular cells in the MultiRow control.

When you use row and column freezing, you keep specific rows or columns visible when you scroll in the MultiRow. For example, you might want to keep the row and column labels visible as you scroll.

The following image shows how the MultiRow control appears after freezing three rows and two columns using the FrozenColumns and the FrozenRows property. This example uses the sample created in the Quick Start topic.



ID	Ordered	Customer	Customer Email	
Amount	Shipped	Address	City	State
0	8/15/2010	Paul Bannon	Paul.Bannon@gmail.com	
$4,717.00	8/17/2010	5833 Wong St.	Hamburg	SP
1	12/28/2015	Tom Bannon	Tom.Bannon@gmail.com	
38	10/22/2014	Bill White	Bill.White@gmail.com	
$2,259.00	10/23/2014	803 Johnson St.	Florence	SP
39	5/17/2013	Sue Adams	Sue.Adams@gmail.com	
$4,073.00	5/18/2013	5229 Wong St.	Rome	CS
40	3/14/2010	Aaron Smith	Aaron.Smith@gmail.com	
$3,978.00	3/16/2010	4511 Johnson St.	Rome	RT
41	3/18/2010	Chris Peters	Chris.Peters@gmail.com	
$3,324.00	3/19/2010	5610 Johnson St.	Rome	SC
42	4/6/2014	Bill Richards	Bill.Richards@gmail.com	
$427.00	4/10/2014	3184 Brown St.	Rome	RS
43	9/23/2012	Sue Bannon	Sue.Bannon@gmail.com	
$1,767.00	9/25/2012	863 Adams St.	Paris	SP
44	7/25/2016	Frank Johnson	Frank.Johnson@gmail.com	

In Code

FrozenCell.cshtml

HTML Helpers

FrozenCell.cshtml

```
@(Html.C1().MultiRow()
    .Bind(bl => bl.Bind(Model))
    .Height(500)
        .Width(750)
    .FrozenColumns(2)
    .FrozenRows(3)
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
            .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
            .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
            .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
```

```
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
            .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
            .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
            .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
            .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
}))
```

Tag Helpers

FrozenCell.cshtml

```
<c1-multi-row id="ovMultiRowCompact" is-read-only="true" height="500 width="750"
class="multirow"              frozen-columns="2" frozen-rows="3">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-multi-row-cell-group header="Order" colspan="2">
    <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

Virtual Scrolling

The MultiRow control provides support for virtual scrolling when working with voluminous data. Bind MultiRow control with large data sets using models or other data sources, and experience a smooth scrolling without any flicker or delay.

The following image shows how the MultiRow control appears after applying virtual scrolling to the large data set. This example uses the sample created in the Quick Start topic.

ID	Ordered	Customer	Customer Email	
Amount	Shipped	Address	City	State
7	11/30/2011	Aaron Wong	Aaron.Wong@gmail.com	
$4,131.00	12/4/2011	4667 Brown St.	Hamburg	SP
8	2/5/2013	Sue Adams	Sue.Adams@gmail.com	
$1,502.00	2/6/2013	9438 White St.	Sidney	RS
9	2/22/2014	John White	John.White@gmail.com	
$3,557.00	2/23/2014	6221 Johnson St.	Hamburg	CS
10	6/9/2011	Tom Bannon	Tom.Bannon@gmail.com	
$507.00	6/10/2011	7728 Brown St.	Cairo	RS
11	10/28/2013	Chris Peters	Chris.Peters@gmail.com	
$521.00	10/29/2013	2645 Richards St.	Rome	RN
12	7/28/2016	Tom Bannon	Tom.Bannon@gmail.com	
$229.00	7/31/2016	7728 Brown St.	Cairo	RS
13	4/5/2015	Sue Brown	Sue.Brown@gmail.com	
$2,491.00	4/8/2015	4916 White St.	Hamburg	CS
14	1/22/2011	Paul Smith	Paul.Smith@gmail.com	
$3,293.00	1/24/2011	9461 Richards St.	Paris	RT

In Code

VirtualScroll.cshtml

To use virtual scrolling mode in MultiRow, set the DisableServerRead property to **false**(default) and set the InitialItemsCount property to a value greater than 0.

HTML Helpers

VirtualScroll.cshtml

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.InitialItemsCount(10).Bind(Model).DisableServerRead(false))
    .IsReadOnly(true)
    .Height(500)
    .Width(780)
    .CssClass("multirow")
    .LayoutDefinition(ld =>
    {
ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
cells.Add(cell => cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
.Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
.Add(cell => cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
.Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
ld.Add().Header("Customer").Colspan(3).Cells(cells =>
```

```
        {
cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
.Add(cell => cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
.Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
.Add(cell => cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
.Add(cell => cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
    }))
```

Tag Helpers

VirtualScroll.cshtml

```
<c1-multi-row id="ovMultiRowCompact" height="500px" width="780px" class="multirow">
<c1-items-source initial-items-count="20" disable-server-read="false" source-
collection="Model" ></c1-items-source>
<c1-multi-row-cell-group header="Order" colspan="2">
<c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
<c1-multi-row-cell binding="Date" header="Ordered" width="150" />
<c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount" />
<c1-multi-row-cell binding="ShippedDate" header="Shipped" />
</c1-multi-row-cell-group>
<c1-multi-row-cell-group header="Customer" colspan="3">
<c1-multi-row-cell binding="Customer.Name" name="CustomerName" header="Cutomer"
width="200" />
<c1-multi-row-cell binding="Customer.Email" name="CustomerEmail" header="Customer
Email" class="email" colspan="2" />
<c1-multi-row-cell binding="Customer.Address" name="CustomerAddress" header="Address"
/>
<c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
</c1-multi-row-cell>
<c1-multi-row-cell binding="Customer.State" name="CustomerState" header="State" />
</c1-multi-row-cell-group>
</c1-multi-row>
```

Selection Modes

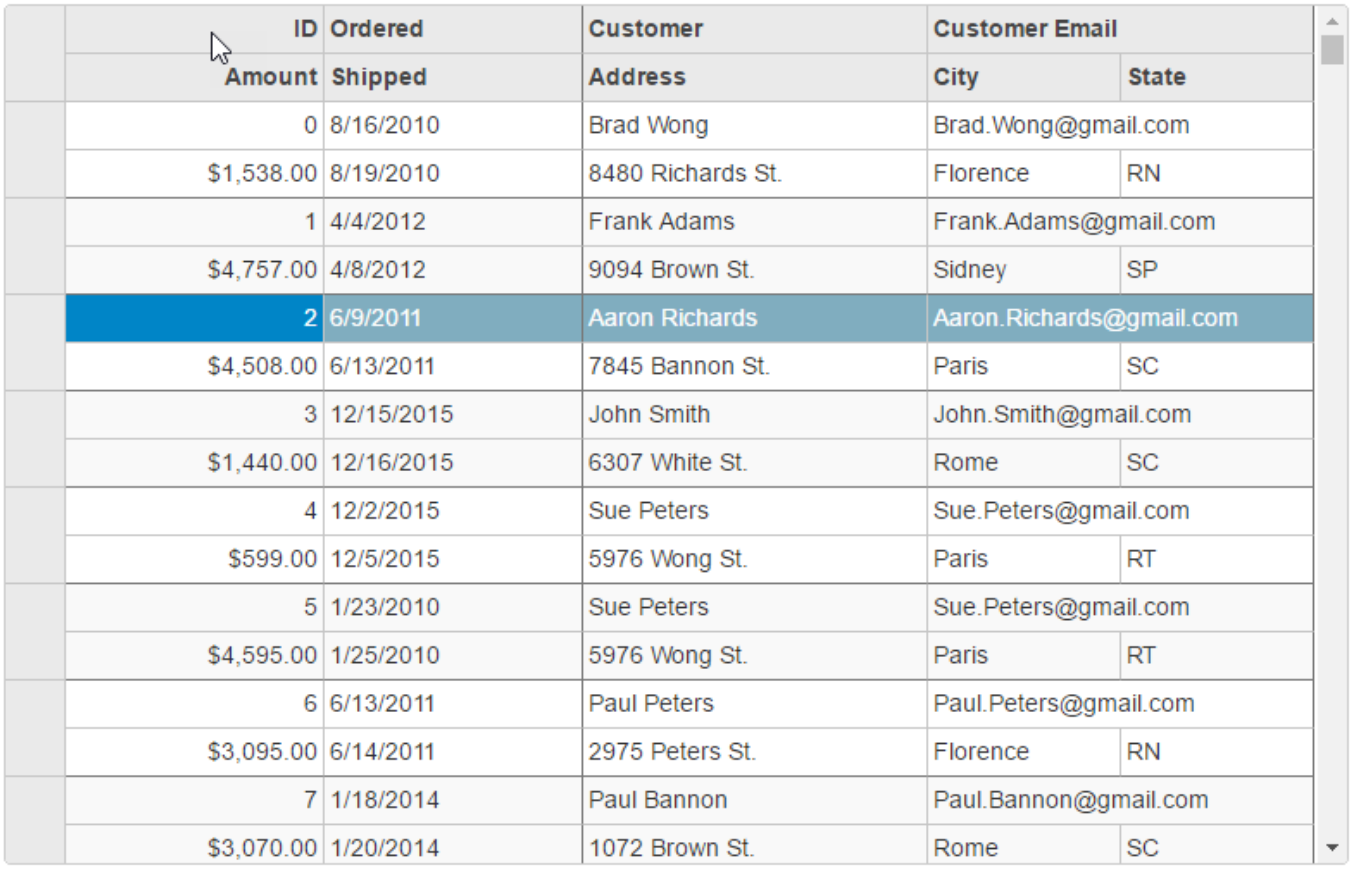
The MultiRow control allows you to select a range of cells with the mouse or keyboard, similar to Excel. The SelectionMode property allows you to select a Row, a Range of Rows, Non-Contiguous Rows (like in a List-Box), a Single Cell, a Range of Cells or disable selection altogether.

To specify the selection behaviour in MultiRow control, you need to set the SelectionMode property to:

- **Cell**: The SelectionMode when set to `Cell`, lets a user select only a single cell at a time.
- **CellRange**: The SelectionMode when set to `CellRange`, lets a user select contiguous blocks of cells.
- **ListBox**: The SelectionMode when set to `ListBox`, lets a user select non-contiguous rows.
- **None**: The SelectionMode when set to `None`, doesn't allow user to select cells with the mouse or keyboard.
- **Row**: The SelectionMode when set to `Row`, lets a user select a single row at a time.

- **RowRange**: The SelectionMode when set to `RowRange`, lets a user select contiguous rows.

The following image shows how the MultiRow control appears on setting SelectionMode to **Row**. This example uses the sample created in the Quick Start topic.



ID	Ordered	Customer	Customer Email	
Amount	Shipped	Address	City	State
0	8/16/2010	Brad Wong	Brad.Wong@gmail.com	
$1,538.00	8/19/2010	8480 Richards St.	Florence	RN
1	4/4/2012	Frank Adams	Frank.Adams@gmail.com	
$4,757.00	4/8/2012	9094 Brown St.	Sidney	SP
2	6/9/2011	Aaron Richards	Aaron.Richards@gmail.com	
$4,508.00	6/13/2011	7845 Bannon St.	Paris	SC
3	12/15/2015	John Smith	John.Smith@gmail.com	
$1,440.00	12/16/2015	6307 White St.	Rome	SC
4	12/2/2015	Sue Peters	Sue.Peters@gmail.com	
$599.00	12/5/2015	5976 Wong St.	Paris	RT
5	1/23/2010	Sue Peters	Sue.Peters@gmail.com	
$4,595.00	1/25/2010	5976 Wong St.	Paris	RT
6	6/13/2011	Paul Peters	Paul.Peters@gmail.com	
$3,095.00	6/14/2011	2975 Peters St.	Florence	RN
7	1/18/2014	Paul Bannon	Paul.Bannon@gmail.com	
$3,070.00	1/20/2014	1072 Brown St.	Rome	SC

In Code

SelectionMode.cshtml

HTML Helpers

Razor

```
.SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
```

Tag Helpers

HTML

```
selection-mode="Row"
```

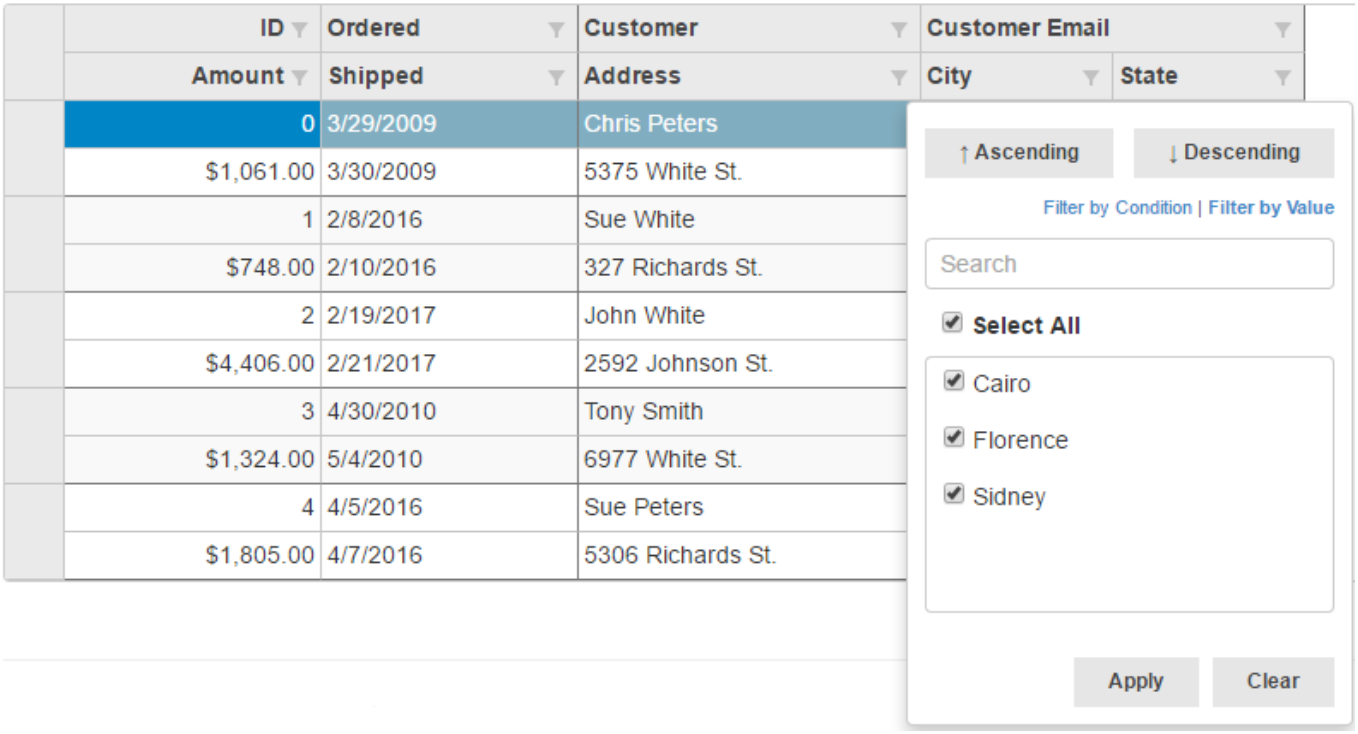
Filtering

MultiRow supports filtering for column entries, you can enable it using ICollectionView interface. It allows you to apply Condition filters and Value filters to the control. MultiRow supports filtering to fetch desired entries from the data.

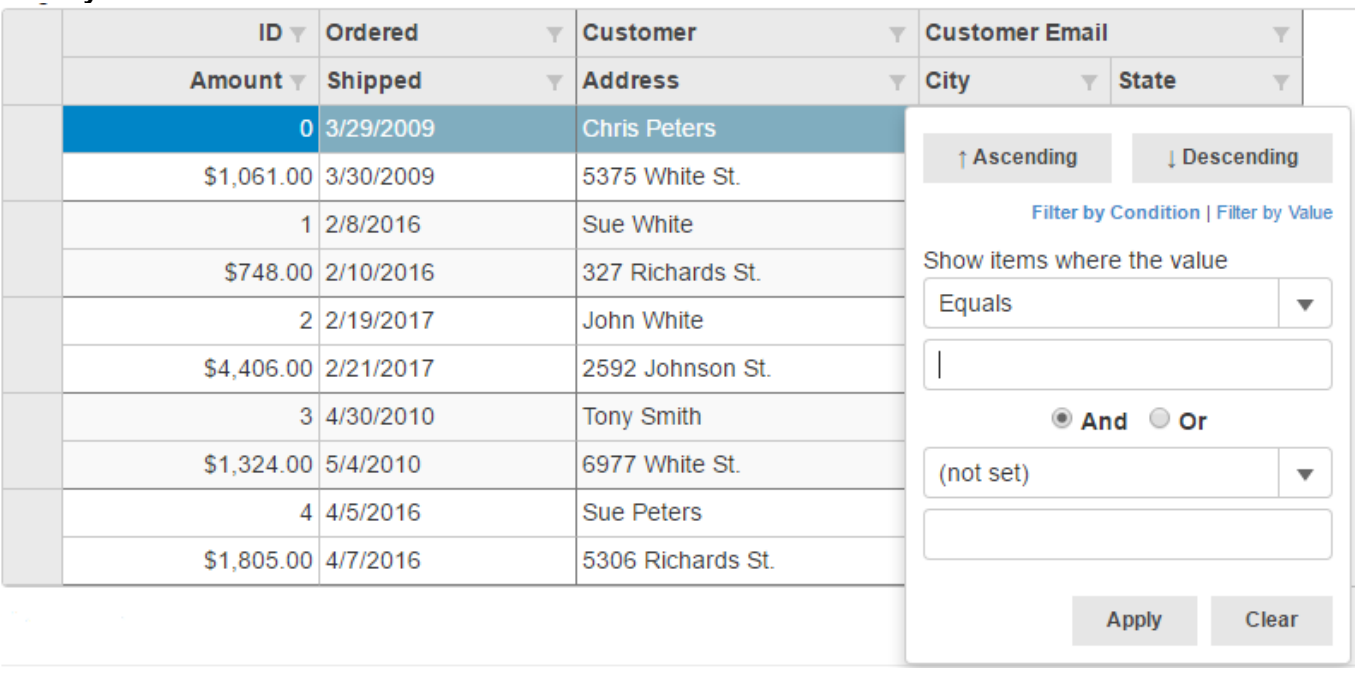
Use **Filter by Condition** to apply conditions to narrow down your search, and **Filter by Value** to precisely locate data corresponding to the desired column value. Ascending and Descending buttons enable you to sort the entries in a particular column in ascending and descending order respectively. Once you enable filtering in MultiRow control, you will notice the filter icon beside each column which helps you to filter the data according your requirements.

The following images show how the MultiRow control appears after applying Filter by Value and Filter by Condition to different columns. This example uses the sample created in the Quick Start topic.

Filter by Value



Filter by Condition



The following code examples demonstrate how to use Filtering in MultiRow control.

In Code

HTML Helpers

Razor

```
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .AllowSorting(true)
    .IsReadOnly(true)
    .SelectionMode(C1.Web.Mvc.Grid.SelectionMode.Row)
    .Filterable(f => f.DefaultFilterType(FilterType.Both))
    .CssClass("multirow")
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
        cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
        .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
        .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
        .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
        cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
        .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
        .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
        .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
        .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
    }))
```

Tag Helpers

HTML

```
<c1-multi-row id="filteringMultiRow" is-read-only="true" selection-mode="Row"
              allow-sorting="true" class="multirow">
    <c1-items-source read-action-url="@Url.Action("RemoteBind_Read")"></c1-items-
source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
```

```
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
    <c1-flex-grid-filter default-filter-type="Both">
        </c1-flex-grid-filter>
</c1-multi-row>
```

Paging

The MultiRow control supports Pager control through which it allows the user to implement paging. Through paging, you can customize the number of items that should be displayed per page and provide a UI for navigating the pages in the grid.

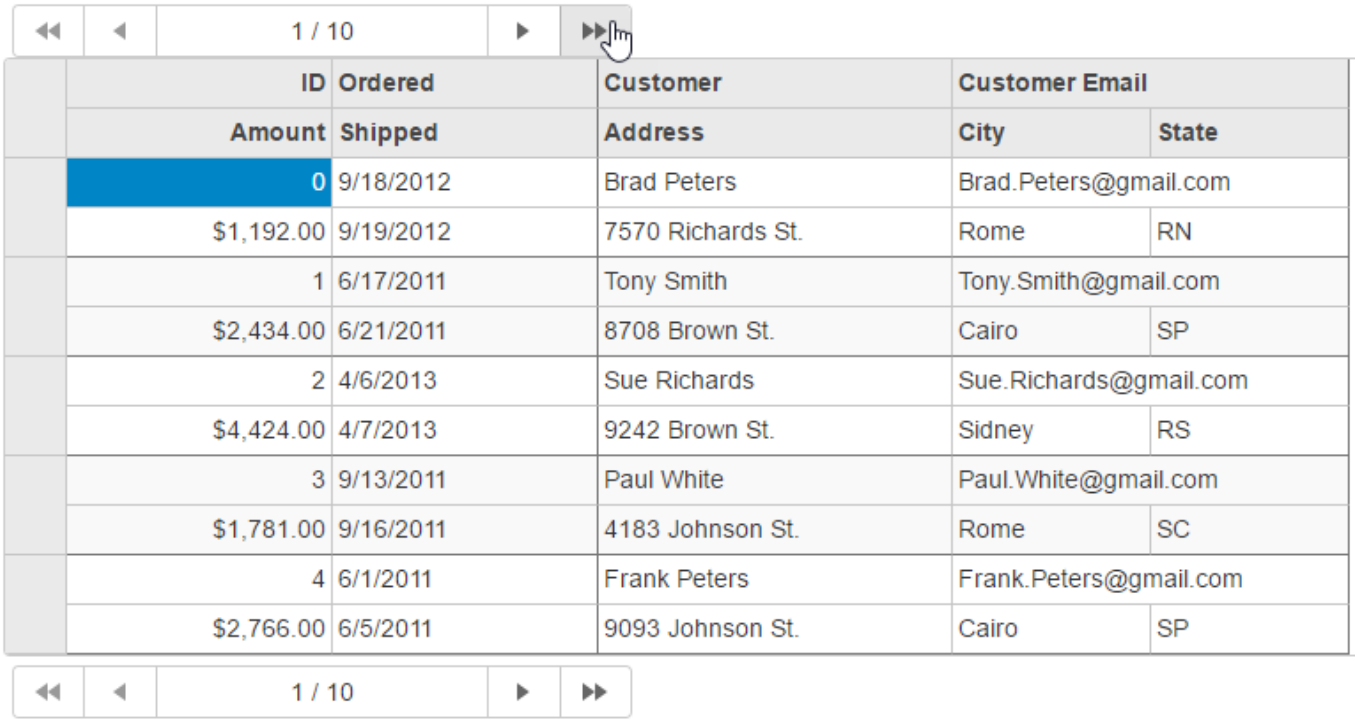
To implement paging in MultiRow using Pager control, set the following properties:

- Set an **ID** for MultiRow control
- Set the **Pager.Owner** property to the id of MultiRow or CollectionViewService
- Set the **PageSize** property of MultiRow or CollectionViewService

> **Note:** That the paging UI is implemented outside of the grid. This gives you complete control over the appearance and functionality of the paging mechanism. To customize the Pager using JavaScript, refer the CollectionView class.

By setting the Owner property of Pager control, the control binds to the MultiRow control and provides function to change the current page of the MultiRow by clicking the following navigation buttons - '<<', '<', '>', '>>'. PageSize property helps the user to specify the number of items to be displayed on each page. In this example, the paging happens on server-side. This is because CollectionView here acts like a service and synchronizes with server data. The CollectionView internally does an Ajax call to fetch next set of data.

The following image shows how the MultiRow control appears after setting the **PageSize** property. This example uses the sample created in the Quick Start topic.



The following code examples demonstrate how to enable Paging using Pager control in MultiRow control.

HTML Helpers

Razor

```
@(Html.C1().CollectionViewService<Orders.Order>
().Bind(Model).Id("collectionViewService")
.PageSize(5))
<br />
@(Html.C1().Pager().Owner("collectionViewService"))
<br />
@(Html.C1().MultiRow<Orders.Order>()
    .Id("pagingMultiRow")
    .ItemsSourceId("collectionViewService").IsReadOnly(true)
    .CssClass("multirow")
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
        cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
        .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
        .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
        .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
        cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
```

```
        .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
        .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
        .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
        .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
    }))
@(Html.C1().Pager().Owner("pagingMultiRow"))
```

Tag Helpers

HTML

```
<c1-items-source id="collectionViewService" read-action-
url="@Url.Action("Paging_Bind")" page-size="5"></c1-items-source>
<br />
<c1-pager owner="collectionViewService"></c1-pager>

<c1-multi-row id="pagingMultiRow" class="multirow customMultiRow" is-read-only="true"
items-source-id="collectionViewService">
    <c1-items-source read-action-url="@Url.Action("RemoteBind_Read")"></c1-items-
source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
header="State" />
    </c1-multi-row-cell-group>
    <c1-flex-grid-filter default-filter-type="Both">
    </c1-flex-grid-filter>
</c1-multi-row>
<c1-pager owner="collectionViewService"></c1-pager>
```

Styling Records, Groups, Cells

The MultiRow control allows you to create your own custom CSS files which are then added to your application. In most of the applications, you would want to show where each record and group starts or ends. The MultiRow control enables this by adding CSS class names to cell elements in the first and last row/column of each group. The class names are **wj-record-start, wj-record-end, wj-group-start, and wj-group-end**. You can use these built-in class names in CSS rules to customize the appearance of the record and group delimiters.

The example below shows how you can use these class names in CSS rules to customize the appearance of the record and group delimiters. It also shows how you can use the standard **CssClass** property to customize the appearance of specific cells within groups.

The following image shows how the MultiRow appears after using the CSS class names and CssClass property. This example uses the sample created in the Quick Start topic.

	ID	Ordered	Customer	Customer Email	
	Amount	Shipped	Address	City	State
	0	8/7/2012	Frank Wong	Frank.Wong@gmail.com	
	$2,994.00	8/10/2012	5723 Smith St.	Sidney	CS
	1	2/23/2014	Joe Wong	Joe.Wong@gmail.com	
	$2,763.00	2/27/2014	8550 Bannon St.	Florence	SC
	2	4/7/2016	Joe Wong	Joe.Wong@gmail.com	
	$3,877.00	4/8/2016	8550 Bannon St.	Florence	SC
	3	5/8/2015	Paul Peters	Paul.Peters@gmail.com	
	$1,553.00	5/10/2015	8417 White St.	Paris	CS
	4	7/13/2012	Frank Wong	Frank.Wong@gmail.com	
	$3,158.00	7/15/2012	2518 Richards St.	Paris	RN

Add CustomMultiRow.css

Create a new ASP.NET MVC application. Once you have created the application, a **Content** folder is created in the Solution Explorer after adding the view to the application. To add a custom style sheet in your application, follow these steps:

1. In the Solution Explorer, right-click the **Content** folder.
2. From the context menu, select **Add | Style Sheet**. The **Specify Name for Item** dialog appears.
3. Set name of the style sheet (for example: `CustomMultiRow.css`)
4. Click **OK** to add CustomMultiRow.css style sheet in your application.
5. Replace the default code of **CustomMultiRow.css** file with the code given below.

CustomMultiRow.css

```
/* custom styling for a MultiRow */
.multirow-css .wj-cell.wj-record-end:not(.wj-header) {
    border-bottom-color: #8fabff; /* blue lines between records */
}
.multirow-css .wj-cell.wj-group-end {
    border-right-color: #bc5505; /* brown lines between groups */
}
.multirow-css .wj-cell.id {
```

```
    color: #c0c0c0;
}
.multirow-css .wj-cell.amount {
    color: #014701;
    font-weight: bold;
}
.multirow-css .wj-cell.email {
    color: #0010c0;
    text-decoration: underline;
}
@font-face {
    font-family: 'Fira';
    src: url("../fonts/fira/FiraSans-Regular.ttf");
    font-weight: normal;
    font-style: normal;
}
@font-face {
    font-family: 'Fira';
    src: url("../fonts/fira/FiraSans-Bold.ttf");
    font-weight: bold;
    font-style: normal;
}
.custom-multi-row .wj-cell {
    font-family: Fira;
}
```

In Code

Styling.cshtml

HTML Helpers

Razor

```
@model IEnumerable<Orders.Order>
@using MultiRow.Models
@section Styles{
    <link rel="stylesheet" href="~/Content/CustomMultiRow.css" />
}
<br />
@(Html.C1().MultiRow<Orders.Order>()
    .Bind(bl => bl.Bind(Model))
    .Id("stylingMultiRow")
    .CssClass("multirow multirow-css")
    .LayoutDefinition(ld =>
    {
        ld.Add().Header("Order").Colspan(2).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Id").Header("ID").CssClass("id").Width("150"))
            .Add(cell => cell.Binding("Date").Header("Ordered").Width("150"))
```

```
            .Add(cell =>
cell.Binding("Amount").Header("Amount").Format("c").CssClass("amount"))
            .Add(cell => cell.Binding("ShippedDate").Header("Shipped"));
        });
        ld.Add().Header("Customer").Colspan(3).Cells(cells =>
        {
            cells.Add(cell =>
cell.Binding("Customer.Name").Name("CustomerName").Header("Customer").Width("200"))
                .Add(cell =>
cell.Binding("Customer.Email").Name("CustomerEmail").Header("Customer
Email").Colspan(2))
                .Add(cell =>
cell.Binding("Customer.Address").Name("CustomerAddress").Header("Address"))
                .Add(cell =>
cell.Binding("Customer.City").Name("CustomerCity").Header("City"))
                .Add(cell =>
cell.Binding("Customer.State").Name("CustomerState").Header("State"));
        });
    }))
```

Tag Helpers

HTML

```
@using MultiRowNetCore.Models
@model IEnumerable<Orders.Order>

@section Styles{
    <link rel="stylesheet" href="~/Content/CustomMultiRow.css" />
}

<c1-multi-row id="stylingMultiRow" class="multirow multirow-css">
    <c1-items-source source-collection="Model" disable-server-read="true"></c1-items-
source>
    <c1-multi-row-cell-group header="Order" colspan="2">
        <c1-multi-row-cell binding="Id" header="ID" width="150" class="id" />
        <c1-multi-row-cell binding="Date" header="Ordered" width="150" />
        <c1-multi-row-cell binding="Amount" header="Amount" format="c" class="amount"
/>
        <c1-multi-row-cell binding="ShippedDate" header="Shipped" />
    </c1-multi-row-cell-group>
    <c1-multi-row-cell-group header="Customer" colspan="3">
        <c1-multi-row-cell binding="Customer.Name" name="CustomerName"
header="Cutomer" width="200" />
        <c1-multi-row-cell binding="Customer.Email" name="CustomerEmail"
header="Customer Email" class="email" colspan="2" />
        <c1-multi-row-cell binding="Customer.Address" name="CustomerAddress"
header="Address" />
        <c1-multi-row-cell binding="Customer.City" name="CustomerCity" header="City">
        </c1-multi-row-cell>
        <c1-multi-row-cell binding="Customer.State" name="CustomerState"
```

```
header="State" />
    </c1-multi-row-cell-group>
</c1-multi-row>
```

MultiRow ASP.NET Core Tags

MultiRow control supports the following ASP.NET Core Tags:

<c1-multi-row> extends <c1-flex-grid>

- center-headers-vertically
- collapsed-headers
- show-header-collapse-button
- LayoutDefinition

<c1-multi-row-cell-group>

- colspan
- header
- cells

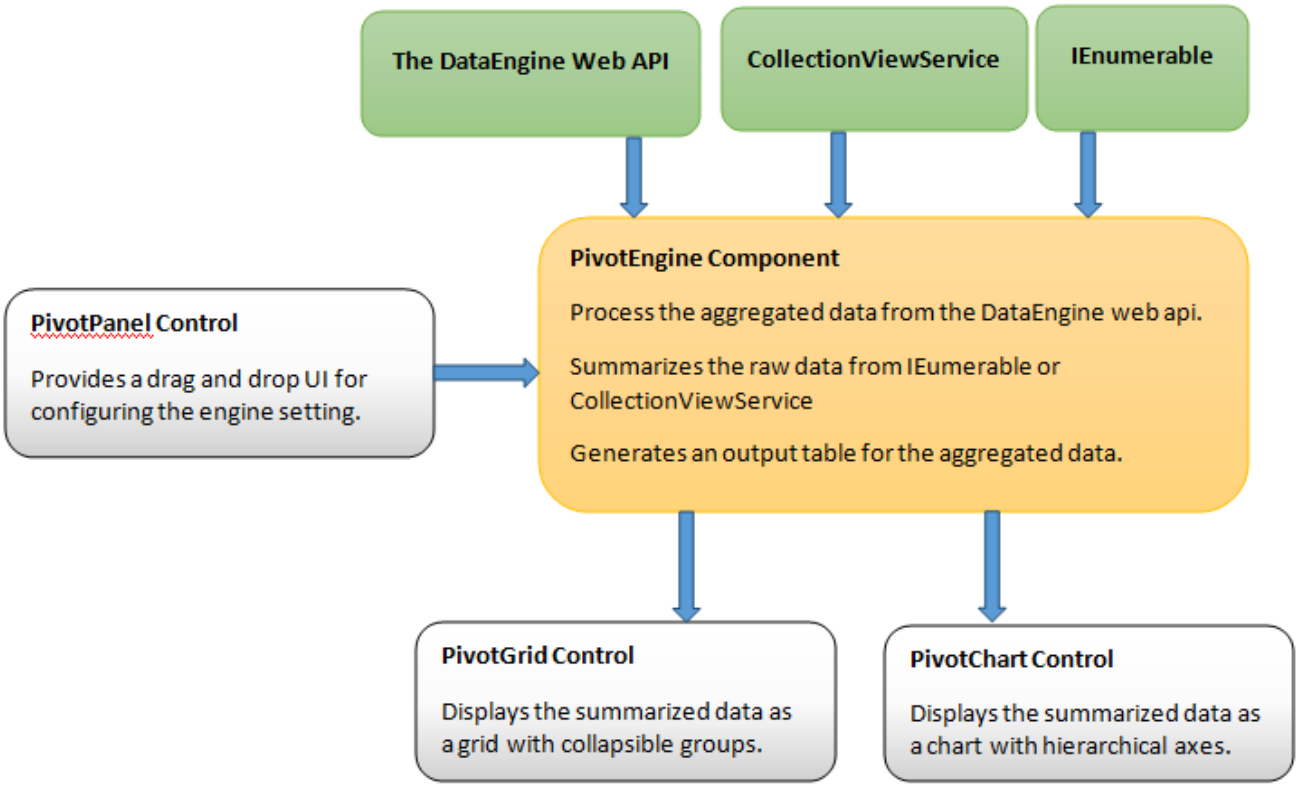
<c1-multi-row-cell> extends <c1-flex-grid-column>

- colspan

OLAP

ComponentOne Studio ASP.NET MVC controls introduces OLAP, a powerful data analysis control designed and developed to cater high-end business intelligence needs. It provides analytical processing features to those found in Microsoft Excel Pivot Table and Pivot Charts. OLAP provides asynchronous processing that improves the performance of the controls as multiple processes can occur simultaneously on separate threads.

OLAP can take up data in any format and provides an easy-to-use interface so users can quickly and intuitively create summaries that display the data in different ways, uncovering trends and providing valuable insights interactively. OLAP uses Pivot Engine which is designed to speed up the retrieval of data from the databases, that helps the user to process huge amount of data with ease.



OLAP uses the following components to do online analytical processing.

1. **Pivot Panel** - The PivotPanel represents as the core component of OLAP, as it provides a user interface for interactively representing the regular data tables into PivotTable, PivotGrid and PivotChart. You can simply provide a database to the PivotPanel, and represent the data in a PivotTable that provides custom views summarizing the data according to your requirements.

2. **Pivot Grid** - The PivotGrid is an extension to an already existing **FlexGrid** class that helps users to create PivotTables and use the FlexGrid features in your OLAP control. There are many different features of FlexGrid that can be utilized in MVC OLAP such as, automatic data binding, grouped row and columns, resizing columns, filtering data and showing details for each cell in the grid.

3. **Pivot Chart** - The PivotChart component is an extension to **FlexChart** class that helps the user to automatically bind data to PivotPanel object, automatic tooltips, chart type selection, and palette selection. You can export these charts to different file formats such as PNG and JPEG, and you can customize the chart styles and interactivity according to your requirements.

4. **Pivot Engine** - The data engine is a a low-footprint C# component that can be easily integrated in your OLAP applications. The engine stores data in memory-mapped files, which are retrieved instantly without any delay in importing these files. The Pivot Engine does not put any restriction on the size of the dataset that you wish to analyze or display, and that too without compromising the performance.

Key Features

The main features of **OLAP for MVC** that you may find useful are as follows:

- **Multiple Value Fields**
  OLAP supports multiple fields in the Values list. Users can drag multiple fields into the Values list and see the results in grid and chart.

- **Enhanced and Powerful Data Engine**
  OLAP is built with a new, powerful data engine that stores data in memory-mapped files using column-oriented technology. The C1DataEngine offers high-speed processing of very large datasets that makes it possible to reach high performance of up to hundreds of millions records in a fraction of a second. The engine also supports query operations such as Aggregation, Joins, Grouping, Filter, Unary/Binary operations, etc. For more information, see Data Binding topic.

- **Enhanced User Interface**
  OLAP comes with an enhanced, Excel-like user interface (UI) offering modern color schemes in toolbar, drop-down menus, tables and charts.

- **Multiple Theme Support**
  OLAP allows developers in choosing a theme that suits their application requirements. To support multiple themes, ASP.NET MVC controls include build in theme support.

- **Plot data on PivotGrid**
  OLAP provides you PivotGrid control support to display data in a grid-like view. The FlexGrid class library enables you to customize your OLAP application. For information on FlexGrid control, see FlexGrid topic.

- **Display data at runtime**
  Use the PivotPanel to determine which fields of your data source should be used to display data and how. Drag fields between the lower areas of the PivotPanel to create a filter, column headers, row headers, or get the sum of values from a column or a row. For more information, see PivotPanel topic.

Introduction to OLAP Technology

Conventional analytical processing tools include **OLAP cubes** and pivot tables such as the ones provided by Microsoft Excel. These tools take up huge chunk of data and summarize it by grouping records based on a set of criteria. For example, an OLAP cube might summarize sales data by grouping it on the basis of product, region and period. In this case, each grid cell would display the total sales for a particular product, in a particular region, and for a specific period. This cell would normally represent data from several records in the original data source

Data analysis and processing tools allow users to redefine grouping criteria dynamically (online). This makes it easy to perform ad-hoc data analysis and discover hidden patterns.

For example, consider the following table:

Date	Product	Region	Sales
Oct 2015	Product A	North	12
Oct 2015	Product B	North	15
Oct 2015	Product C	South	4
Oct 2015	Product A	South	3
Nov 2015	Product A	South	6
Nov 2015	Product C	North	8
Nov 2015	Product A	North	10
Nov 2015	Product B	North	3

Now suppose you were asked to analyze this data and answer questions such as:

- Are sales going up or down?
- Which products are most important to the company?
- Which products are most popular in each region?

In order to answer these simple questions, you would have to summarize the data to obtain tables such as these:

Sales by Date and by Product

Date	Product A	Product B	Product C	Total
Oct 2007	15	15	4	34
Nov 2007	16	3	8	27
Total	**31**	**18**	**12**	**61**

Sales by Product and by Region

Product	North	South	Total
Product A	22	9	31
Product B	18		18
Product C	8	4	12
Total	**48**	**13**	**61**

Each cell in the summary tables represents several records in the original data source, where one or more values fields are summarized (sum of sales in this case) and categorized on the basis of other fields (date, product, or region in this case).

This can be done easily in a spreadsheet, but the work is tedious, repetitive, and error-prone. Even if you write a custom application to summarize the data, probably you spend a lot of time maintaining it to add new views, and users might get constrained in their analyses to the views implemented by you.

OLAP allows users to define the views they want in an interactive, and ad-hoc fashion. They can use pre-defined views or create and save new ones. Any changes to the underlying data are reflected automatically in the views, and users can create and share reports showing these views. In short, **OLAP** control provides flexible and efficient data analysis process.

OLAP Architecture

This section comprises of topics that highlights the architectural features of OLAP control. OLAP is a powerful data analysis control and it mainly includes three major components, which are as follows:

PivotPanel
: Learn about different elements of Pivot Panel in OLAP control.

PivotChart
: Learn about the working of PivotChart in OLAP control.

PivotGrid
: Learn about the working of PivotGrid in OLAP control.

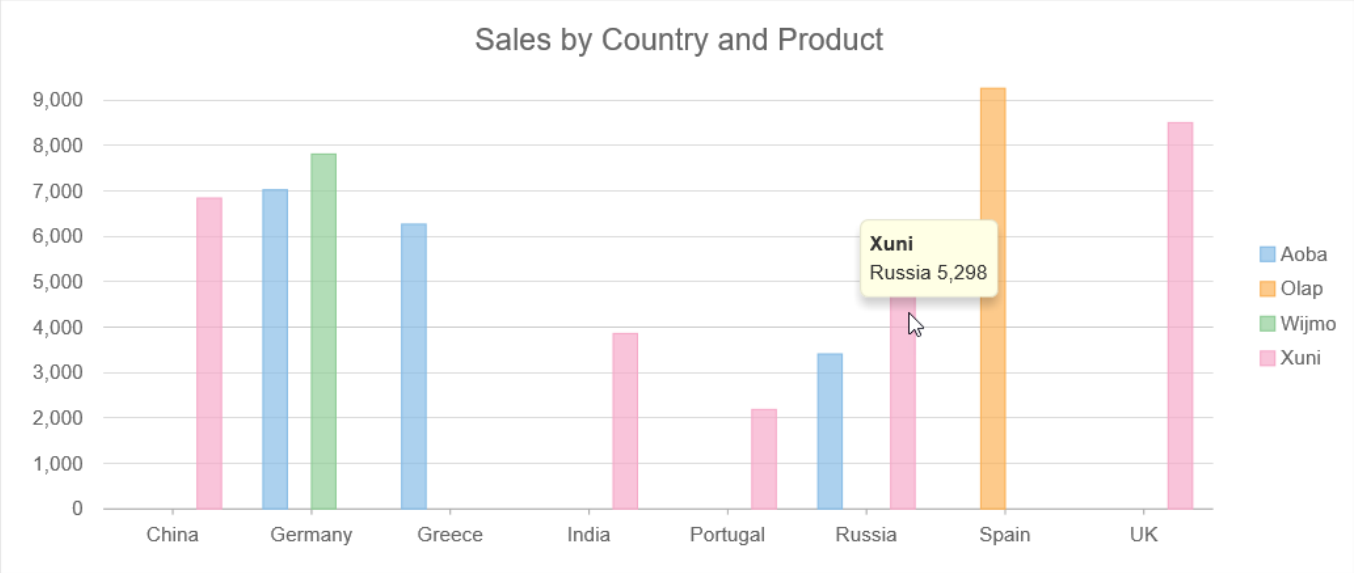
Pivot Panel

PivotPanel component is designed to provide an Excel-like drag-and-drop user interface that enables you to define custom views of the data. The control displays a list containing all the fields that exist in the data. Users can drag-and-drop these fields to lists that represent the row and column dimensions of the output table, the values summarized in the output data cells, and the fields used for filtering the data.

When you are working with PivotPanel, the core of this component is a powerful data engine that is responsible for fetching raw data from the data source as per the criteria selected by the user. The criteria for data selection is determined by the fields enlisted in the various lists appearing on the panel. OLAP features an open architecture. It can accept any regular collection as data, including tables and generic lists; and add LINQ enumerations to summarize the data and produce a regular data table or chart as output. For more information about PivotPanel, see QuickStart:Add Data to OLAP topic.

Pivot Chart

The PivotChart displays data fetched from the data source in the form of Pivot charts. The PivotChart control can be bound to a data source to populate it with data. The control provides automatic data binding with PivotPanel objects, automatic tooltips, chart types and integrated palettes.



You can also export these charts to PNG and JPEG file formats, and customize the chart styles and interactivity according to your requirements. For more information about PivotChart, see QuickStart:Add Data to OLAP topic.

Pivot Grid

The PivotGrid displays data fetched from the data source in the form of Pivot tables. PivotGrid can be bound to a data source for populating it with an ordered dataset. The control provides automatic data binding with PivotPanel objects, grouped row and column headers, and custom behaviors for resizing columns, copying data to the clipboard, and showing details for a given cell.

Country	Aoba	Olap	Wijmo	Xuni	Grand Total
China				6,842	**6,842**
Germany	7,026		7,810		**14,836**
Greece	6,269				**6,269**
India				3,860	**3,860**
Portugal				2,189	**2,189**
Russia	3,410			5,298	**8,708**
Spain		9,260			**9,260**
UK				8,502	**8,502**
Grand Total	**16,705**	**9,260**	**7,810**	**26,691**	**60,466**

Extending the features of PivotGrid control, the PivotGrid control allows users to export the grid content to Excel sheet or use styles and owner-draw cells to customize the grid's overall appearance. For more information about PivotChart, see QuickStart:Add Data to OLAP topic.

Quick Start: Add Data to OLAP

Complete the following steps to add data to OLAP control:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for OLAP**

- **Step 3: Add an OLAP control**
- **Step 4: Build and Run the Project**

The following image shows how OLAP control appears in the browser after completing the above steps:

Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for OLAP

Create a new class inside the Models folder to create a 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 OlapSample.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 3: Add an OLAP control

Create a Controller and View for OLAP control and follow the below steps to initialize an OLAP control.

Add a new Controller

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: `OlapController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
public class OlapController : Controller
{
    private static System.Collections.IEnumerable Data = ProductData.GetData(10).ToList();
    // GET: PivotGrid
    public ActionResult Index()
    {
        return View(Data);
    }
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `OlapController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

```
@using OlapSample.Models;
@model IEnumerable<ProductData>
<br />
@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("indexEngine")
@Html.C1().PivotChart().ItemsSourceId("indexEngine")
@Html.C1().PivotGrid().ItemsSourceId("indexEngine")
```

Tag Helpers

Razor

```
@model IEnumerable<ProductData>
<c1-pivot-engine id="indexEngine">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-collection>
</c1-pivot-engine>
<c1-pivot-panel items-source-id="indexEngine"></c1-pivot-panel>
<c1-pivot-chart items-source-id="indexEngine"></c1-pivot-chart>
```

```
<c1-pivot-grid items-source-id="indexEngine"></c1-pivot-grid>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Olap/Index**) in the address bar of the browser to see the view.

Back to Top

Features

Data Binding

OLAP is designed as powerful data analysis tools to deliver high-end business intelligence capabilities, the data can be added to the OLAP control through various data binding options. Once you have bind the data to your OLAP control, you can use many different interactive features such as filtering, grouping, sorting, drill-down, applying conditional formatting, creating interactive reports.

This section contains information about

Model Binding
: Learn how to add data to your OLAP control using basic Model binding.

Remote Data Binding
: Learn how to remotely bind data to your OLAP control using C1JSONRequest.

Data Engine Service
: Learn how to use Data Engine service.

Cube Data Binding
: Learn how to use cube data in OLAP.

Model Binding

This topic describes the steps required to add data in the OLAP control using model binding. You can also perform remote data binding in OLAP. For more information about remote data binding, see Remote Data Binding topic.

Complete the following steps to implement model data binding in Olap control.

- **Step 1: Create a Datasource for OLAP**
- **Step 2: Add an OLAP control**
- **Step 3: Build and Run the Project**

The following image shows how OLAP control appears in the browser after completing the above steps:

Step 1: Create a Datasource for OLAP

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 OlapSample.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 2: Add an OLAP control

Create a controller and view for OLAP control and follow the below steps to initialize an OLAP control.

Complete the following steps to initialize an OLAP control.

Add a new Controller

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: `OlapController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
public class OlapController : Controller
{
    private static System.Collections.IEnumerable Data = ProductData.GetData(10).ToList();
    // GET: PivotGrid
    public ActionResult Index()
    {
        return View(Data);
    }
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `OlapController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

```
@using OlapSample.Models;
@model IEnumerable<ProductData>
<br />
@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("indexEngine")
@Html.C1().PivotChart().ItemsSourceId("indexEngine")
@Html.C1().PivotGrid().ItemsSourceId("indexEngine")
```

Tag Helpers

Razor

```
@model IEnumerable<ProductData>
<c1-pivot-engine id="indexEngine">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-collection>
</c1-pivot-engine>
<c1-pivot-panel items-source-id="indexEngine"></c1-pivot-panel>
<c1-pivot-chart items-source-id="indexEngine"></c1-pivot-chart>
<c1-pivot-grid items-source-id="indexEngine"></c1-pivot-grid>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Olap/Index**) in the address bar of the browser to see the view.

Back to Top

Remote Binding

OLAP allows you to retrieve data directly using **C1JSONRequest**. This specifies remote data URLs that include the server, table and columns. The arrays returned are used as data sources for **CollectionView** objects.

`CollectionViewHelper` is a static class that enables collections to have editing, filtering, grouping, and sorting services. The `Bind` property of **PivotEngine** is used to bind it to a collection by passing an action URL method to carry out a specific operation. This class also includes the following methods:

- `Read()`: Retrieves data from the collection.
- `Edit()`: Enables excel-style editing in OLAP.
- `BatchEdit()`: Allows editing multiple items at a time.

This topic comprises of three steps:

- **Step 1: Create a Datasource for OLAP**
- **Step 2: Add an OLAP control**
- **Step 3: Build and Run the Project**

Step 1: Create a Datasource for OLAP

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`). See Adding controls to know how to add a new model.
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 OlapSample.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 2: Add an OLAP control

Create a Controller and View for OLAP control and follow the below steps to initialize an OLAP control.

Complete the following steps to initialize an OLAP control.

Add a new Controller

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: `OlapController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
partial class RemoteBindController : Controller
{
    private static IEnumerable<ProductData> RemoteData = ProductData.GetData(100000).ToList();
    // GET: PivotGrid
    public ActionResult RemoteBind()
    {
        return View();
    }

    public ActionResult RemoteBind_Read([C1JsonRequest] CollectionViewRequest<ProductData> requestData)
    {
        return this.C1Json(CollectionViewHelper.Read(requestData, RemoteData));
    }
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `OlapController.`
2. Place the cursor inside the method `Index()`.
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

```
@using RemoteDataOlap.Models;
@model IEnumerable<ProductData>

@(Html.C1().PivotEngine().Id("remoteEngine").Bind(Url.Action("RemoteBind_Read"))
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("remoteEngine")
@Html.C1().PivotGrid().ItemsSourceId("remoteEngine")
```

Tag Helpers

Razor

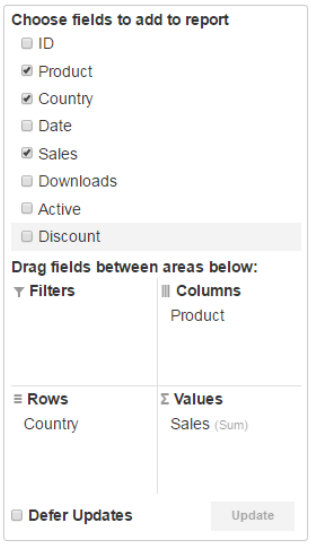
```
@using RemoteDataOlap.Models;
@model IEnumerable<ProductData>
<c1-pivot-engine id="remoteEngine">
    <c1-items-source read-action-url="@Url.Action("RemoteBind_Read")"></c1-items-source>
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-collection>
</c1-pivot-engine>
<c1-pivot-panel items-source-id="remoteEngine"></c1-pivot-panel>
<c1-pivot-grid items-source-id="remoteEngine"></c1-pivot-grid>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Olap/Index**) in the address bar of the browser to see the view.

The following image shows how OLAP control appears in the browser after completing the above steps:



Country	oba	Olap	Wijmo	Xuni	Grand Total
China	,539,962	10,900,745	11,532,409	11,863,010	**45,836,126**
Germany	,438,646	11,247,991	11,825,597	11,337,380	**45,849,614**
Greece	,364,532	11,506,770	11,420,060	11,218,474	**45,509,836**
India	,553,404	11,412,533	11,393,715	11,425,251	**45,784,903**
Italy	,557,860	11,247,687	11,532,478	11,618,197	**45,956,222**
Japan	,912,164	11,375,710	11,136,572	11,577,188	**45,001,634**
Portugal	,796,337	10,663,634	11,380,756	11,835,447	**45,676,174**
Russia	,334,546	10,641,443	11,198,018	11,179,263	**44,353,270**
Spain	,621,533	11,493,202	11,200,970	11,098,308	**45,414,013**
UK	,123,948	10,981,620	11,451,796	11,460,067	**45,017,431**
US	,494,967	11,394,754	10,709,622	11,747,479	**45,346,822**
Grand Total	,737,899	122,866,089	124,781,993	126,360,064	**499,746,045**

Back to Top

Data Engine Service

This topic describes the steps required to use the Data Engine service for data aggregation in OLAP. In the example below, the **PivotEngine** component connects the DataEngine data using the Data Engine Service. The **PivotPanel** control and **PivotGrid** control binds to the **PivotEngine**. You can change the view definition in the **PivotPanel** control. The aggregated data will be obtained from the service. In the example below, the **PivotGrid** control shows the aggregated data. You can find the detail raw data shown in a grid by double-clicking some cell in the PivotGrid control.

This topic comprises of the following steps:

- **Step 1: Create an MVC Application using Visual Studio template**
- **Step 2: Install the DataEngine Web API**
- **Step 3: Create a Datasource for OLAP**
- **Step 4: Configure Startup.cs**
- **Step 5: Add an OLAP control**
- **Step 6: Build and Run the Project**

Step 1: Create an MVC Application using Visual Studio template

Create an ASP.NET MVC Application using Visual Studio template to enable WebAPI configuration.

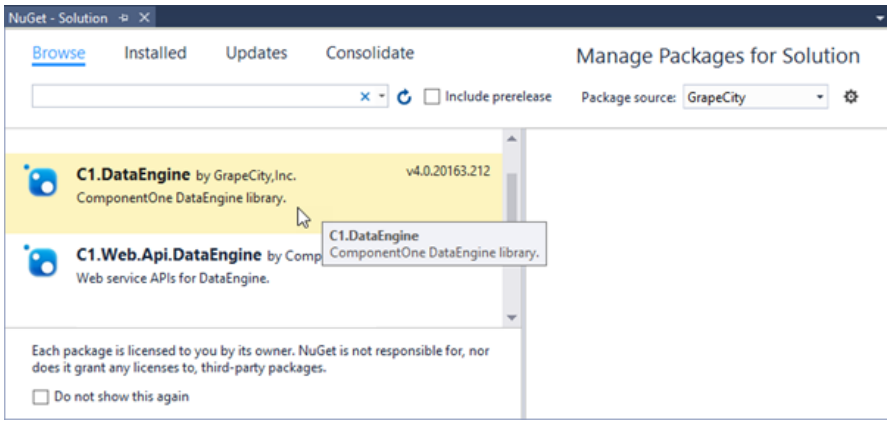
1. Select **File | New | Project**.
2. Under installed templates, select **Visual C# | Web | ASP.NET Web Application (.NET Framework)**.
3. In the **New ASP.NET Web Application** dialog, select the **MVC** template.
4. Under "Add folders and core references for", check **Web API**. Click **OK**.
   For more information about licensing, resource registration, and assembly references, see Using Visual Studio Template.

Back to Top

Step 2: Install the DataEngine Web API

Install the DataEngine Web API and C1.WebApi packages from the NuGet server.

1. Select **GrapeCity** from the **Package source** drop-down, in NuGet Package manager of your application.
2. Browse for the DataEngine Web API packages, and install them.



You will see that the following references are added to your Visual Studio project.

- C1.WebApi.dll
- C1.WebApi.DataEngine.dll
- C1.DataEngine.4.dll
- System.Net.Http.Formatting.dll
- System.Web.Http.dll
- System.Web.Http.Owin.dll
- System.Web.Http.WebHost.dll

Step 3: Create a Datasource for OLAP

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). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes.

C#

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

namespace OlapSample.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: Configure Startup.cs

After adding the required references, you need to configure the `Startup.cs` to fetch the aggregated data from the data engine service.

1. In **Solution Explorer**, select your 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. In the Startup.cs file, add the following code inside the **Startup** class.

```
using C1.DataEngine;
using Microsoft.Owin;
using Owin;
using System.IO;
using System.Linq;
using System.Web.Http;
using OlapSSAS.Models;

[assembly: OwinStartupAttribute(typeof(OlapSSAS.Startup))]
namespace OlapSSAS
{
public partial class Startup
   {
 private readonly HttpConfiguration config = GlobalConfiguration.Configuration;
 private static string DATAPATH = Path.Combine(System.Web.HttpRuntime.AppDomainAppPath, "Data");
 public void Configuration(IAppBuilder app)
     {
     app.UseDataEngineProviders()
     .AddDataEngine("complex10", () =>
     {
         return ProductData.GetData(100000);
     })
        }
    }
}
```

Back to Top

Step 5: Add an OLAP control

Create a controller and view for OLAP control and follow the below steps to initialize an OLAP control.

Add a new Controller

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: `OlapController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
// GET: SSAS
public ActionResult Index()
{
    return View();
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `OlapController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

```
@using C1.Web.Mvc
@Html.C1().Scripts().Basic().Olap()
```

```
@(Html.C1().PivotEngine().Id("dataEngine")
        .BindService("~/api/dataengine/complex10")
        .RowFields(pfcb => pfcb.Items("Country"))
        .ColumnFields(cfcb => cfcb.Items("Product"))
        .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("dataSourceEngine")
@Html.C1().PivotChart().ItemsSourceId("dataSourceEngine")
@Html.C1().PivotGrid().ItemsSourceId("dataSourceEngine")
```

Tag Helpers

Razor

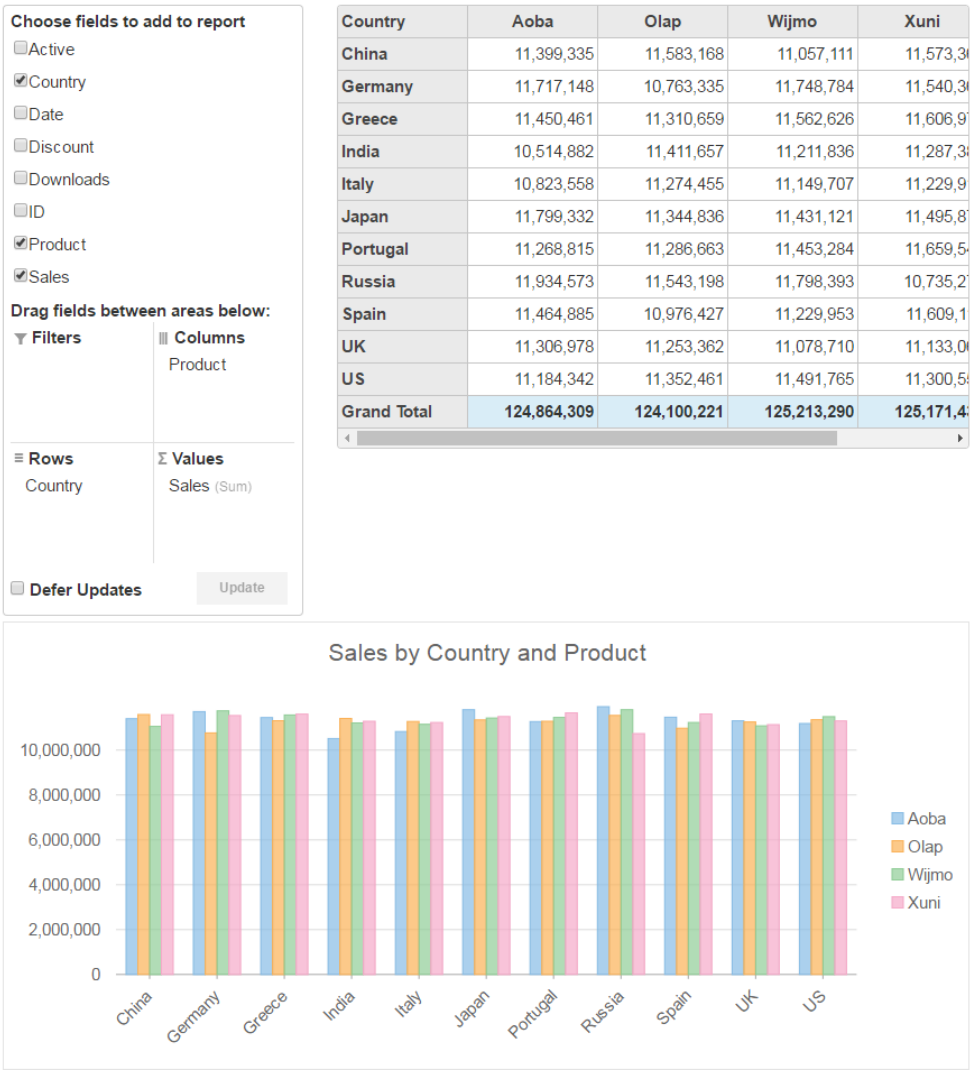
```
@using C1.Web.Mvc.Grid
<c1-pivot-engine id="dataSourceEngine" service-url="~/api/dataengine/complex10">
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-collection>
</c1-pivot-engine>
<c1-pivot-panel items-source-id="dataSourceEngine"></c1-pivot-panel>
<c1-pivot-chart items-source-id="dataSourceEngine"></c1-pivot-chart>
<c1-pivot-grid items-source-id="dataSourceEngine"></c1-pivot-grid>
```

Step 6: Build and Run the Project

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

> Append the folder name and view name to the generated URL (for example: http://localhost:1234/**Olap/Index**) in the address bar of the browser to see the view.

The following image shows how OLAP control appears in the browser after completing the above steps:



Back to Top

Cube Data using SSAS

This section describes the steps required to add cube data in the OLAP control using **SSAS** (SQL Server Analysis Services). In the example below, the PivotEngine component binds to a service. In this the **DataEngine** Web Api is supported, the data engine is responsible for data aggregation in this example. The **PivotPanel** control and **PivotGrid** control binds the **PivotEngine**. You can change the view definition in the **PivotPanel** control. The aggregated data will be obtained from the service. Then the **PivotGrid** control displays the aggregated data.

This topic comprises of the following steps:

- **Step 1: Create an MVC Application using Visual Studio template**
- **Step 2: Install the DataEngine Web API**
- **Step 3: Configure Startup.cs**
- **Step 4: Add an OLAP control**
- **Step 5: Build and Run the Project**

Step 1: Create an MVC Application using Visual Studio template

Create an ASP.NET MVC Application using Visual Studio template to enable WebAPI configuration.

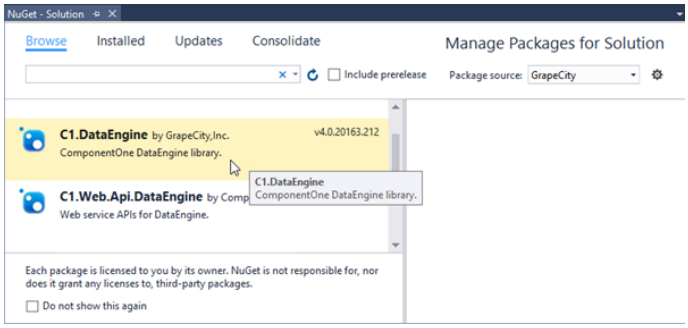
1. Select **File | New | Project**.
2. Under installed templates, select **Visual C# | Web | ASP.NET Web Application (.NET Framework)**.
3. In the **New ASP.NET Web Application** dialog, select the **MVC** template.
4. Under "Add folders and core references for", check **Web API**. Click **OK**.
   For more information about licensing, resource registration, and assembly references, see Using Visual Studio Template.

Back to Top

Step 2: Install the DataEngine Web API

Install the DataEngine Web API and C1.WebApi packages from the NuGet server.

1. Select **GrapeCity** from the **Package source** drop-down, in NuGet Package manager of your application.
2. Browse for the DataEngine Web API packages, and install them.



You will see that the following references are added to your Visual Studio project.

- C1.WebApi.dll
- C1.WebApi.DataEngine.dll
- C1.DataEngine.4.dll
- System.Net.Http.Formatting.dll
- System.Web.Http.dll
- System.Web.Http.Owin.dll
- System.Web.Http.WebHost.dll

Step 3: Configure Startup.cs

After adding the required references, you need to configure the `Startup.cs` to fetch the data that is stored on SSAS server.

1. In **Solution Explorer**, select your 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. In the Startup.cs file, add the following code inside the **Startup** class.

```
using C1.DataEngine;
using Microsoft.Owin;
using Owin;
using System.IO;
using System.Linq;
using System.Web.Http;
using OlapSSAS.Models;

[assembly: OwinStartupAttribute(typeof(OlapSSAS.Startup))]
namespace OlapSSAS
{
 public partial 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");
        }
    }
}
```

Back to Top

Step 4: Add an OLAP control

Create a Controller and View for OLAP control and follow the below steps to initialize an OLAP control.

Add a new Controller

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: `OlapController`).
    3. Click **Add**.
4. Replace the method Index() with the following method.

C#

```
// GET: SSAS
public ActionResult Index()
{
    return View();
}
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `OlapController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

```
@using C1.Web.Mvc.Olap;
@using C1.Web.Mvc.Grid
@Html.C1().Styles()
@Html.C1().Scripts().Basic().Olap()

@(Html.C1().PivotEngine().Id("ssasEngine")
```

```
        .BindService("~/api/dataengine/cube")
        .Fields(pfcb =>
            pfcb.Items(c =>
                c.AddCubeField(fb => fb.Header("Internet Orders").DimensionType(DimensionType.Folder).SubFields(sfsb =>
                        sfsb.Add(sfb => sfb.Header("Internet Order Count").Binding("[Measures].[Internet Order Count]").DataType(DataType.Number))))
                .AddCubeField(fb => fb.Header("Product").SubFields(sfsb =>
                        sfsb.Add(sfb => sfb.Header("Category").Binding("[Product].[Category]").DataType(DataType.String))
                        .Add(sfb => sfb.Header("Stocking").SubFields(sfs =>
                                sfs.Add(f => f.Binding("[Product].[Color]").Header("Color").DataType(DataType.String))
                                .Add(f => f.Binding("[Product].[Class]").Header("Class").DataType(DataType.String))))
                        .Add(sfb => sfb.Header("Product").Binding("[Product].[Product]").DataType(DataType.String))
                ))))
        .RowFields(rfb => rfb.Items("[Product].[Product]"))
        .ValueFields(vfb => vfb.Items("[Measures].[Internet Order Count]"))
)
@Html.C1().PivotPanel().ItemsSourceId("ssasEngine")
@Html.C1().PivotGrid().ItemsSourceId("ssasEngine")
```

Tag Helpers

Razor

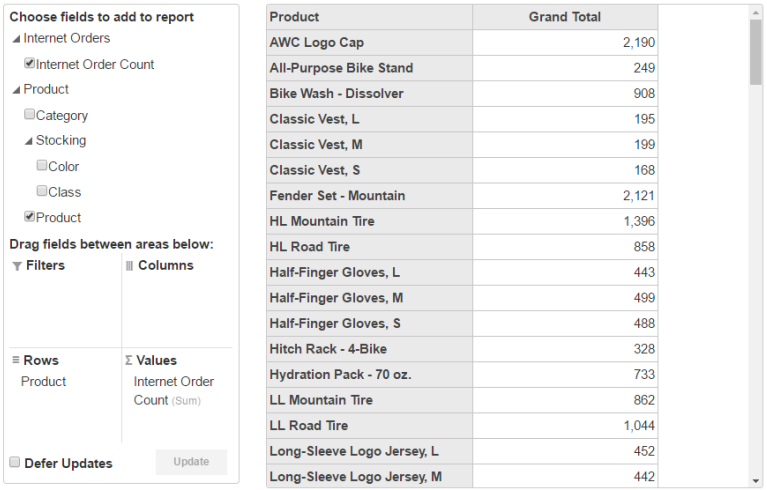
```
@using C1.Web.Mvc.Grid

<c1-pivot-engine id="ssasEngine" service-url="~/api/dataengine/cube">
<c1-pivot-field-collection>
<c1-pivot-field header="Internet Orders">
<c1-pivot-field binding="[Measures].[Internet Order Count]" header="Internet Order Count" type="DataType.Number"></c1-pivot-field>
</c1-pivot-field>
<c1-pivot-field header="Product">
<c1-pivot-field binding="[Product].[Category]" header="Category" type="DataType.String"></c1-pivot-field>
<c1-pivot-field header="Stocking">
<c1-pivot-field binding="[Product].[Color]" header="Color" type="DataType.String"></c1-pivot-field>
<c1-pivot-field binding="[Product].[Class]" header="Class" type="DataType.String"></c1-pivot-field>
</c1-pivot-field>
<c1-pivot-field binding="[Product].[Product]" header="Product" type="DataType.String"></c1-pivot-field>
</c1-pivot-field>
</c1-pivot-field-collection>
<c1-view-field-collection c1-property="RowFields" items="[Product].[Product]"></c1-view-field-collection>
<c1-view-field-collection c1-property="ValueFields" items="[Measures].[Internet Order Count]"></c1-view-field-collection>
</c1-pivot-engine>
```

Step 5: Build and Run the Project

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

> Append the folder name and view name to the generated URL (for example: http://localhost:1234/**Olap/Index**) in the address bar of the browser to see the view.

The following image shows how OLAP control appears in the browser after completing the above steps:



Back to Top

Defining Pivot Fields

The main advantage of an OLAP application is interactivity. User must be able to create and modify views, easily and quickly see the output on the web browser. This is possible in OLAP with the help of Excel-like user interface and user friendly dialogs. However, in case you want to configure the view and add pivot fields through code, you can achieve that using the PivotField and PivotEngine classes.

> **Note**: In case your field name has multiple upper case letters (For example - ProductName), then view field adds a space between the field names on every instance of upper case letter and throws an exception. You can

> resolve this by setting the header property of pivot fields to a value same as the field name.

This topic helps you to understand, how to define pivot fields in an OLAP control through code. The following image shows how OLAP control appears after defining pivot fields.

Country	Aoba	Olap	Wijmo	Grand Total
China		8,814.00		8,814.00
Germany			6,802.00	6,802.00
India			9,631.00	9,631.00
Japan		9,823.00	3,614.00	6,718.50
Portugal	4,067.00		8,046.00	6,056.50
Spain	6,503.00			6,503.00
UK			1,861.00	1,861.00
US	4,911.00			4,911.00
Grand Total	5,160.33	9,318.50	5,990.80	6,407.20

If you do not want to auto-generate the fields, use the following example code to define pivot fields in OLAP control. The example uses **ProductData.cs** model added in the Quick Start: Add Data to OLAP topic.

HTML Helpers

Razor

```
@using <ApplicationName>.Models;
@model IEnumerable<ProductData>

@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .Fields(fld =>
    {
        fld.Items(it => it.AddPivotField(pf => pf.Binding("Country"))
        .AddPivotField(pf => pf.Binding("Product"))
        .AddPivotField(pf =>
pf.Binding("Sales").Aggregate(C1.Web.Mvc.Grid.Aggregate.Avg)));
    })
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))

@(Html.C1().PivotGrid().ItemsSourceId("indexEngine"))
```

Tag Helpers

HTML

```
@using <ApplicationName>.Models;
@model IEnumerable<ProductData>
<br />
```

```
<c1-pivot-engine id="indexEngine">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-pivot-field-collection>
        <c1-pivot-field binding="Country"></c1-pivot-field>
        <c1-pivot-field binding="Product"></c1-pivot-field>
        <c1-pivot-field binding="Sales" aggregate="Avg"></c1-pivot-field>
    </c1-pivot-field-collection>
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-
field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-
field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-
field-collection>
</c1-pivot-engine>
<c1-pivot-grid items-source-id="indexEngine"></c1-pivot-grid>
```

Update Field Properties at Runtime

OLAP control allows you to access and update field properties at runtime using the object model. You can auto generate the fields and set the properties of value fields at the client side.

> **Note**: In case your field name has multiple upper case letters (For example - ProductName), then view field adds a space between the field names on every instance of upper case letter and throws an exception. You can resolve this by setting the header property of pivot fields to a value same as the field name.

This topic helps you to understand, how to access and update field properties in an OLAP control at client side. The following image shows how OLAP control appears after updating field properties at runtime using the JavaScript code.

Country	Aoba	Olap	Wijmo	Grand Total
China		8,814.00		8,814.00
Germany			6,802.00	6,802.00
India			9,631.00	9,631.00
Japan		9,823.00	3,614.00	6,718.50
Portugal	4,067.00		8,046.00	6,056.50
Spain	6,503.00			6,503.00
UK			1,861.00	1,861.00
US	4,911.00			4,911.00
Grand Total	5,160.33	9,318.50	5,990.80	6,407.20

If you want to auto-generate the fields and set/update the properties of value fields at runtime, use the following example code to define pivot fields in OLAP control. The example uses **ProductData.cs** model added in the Quick Start: Add Data to OLAP topic.

HTML Helpers

Razor

```
@using <ApplicationName>.Models;
@model IEnumerable<ProductData>

<script type="text/javascript">
        c1.documentReady(function () {
            var engine = c1.getService('indexEngine');
            var field = engine.fields.getField("Sales");
            field.aggregate = wijmo.Aggregate.Avg;
            field.format = "n2";
        })
</script>
@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@(Html.C1().PivotGrid().ItemsSourceId("indexEngine"))s
```

Tag Helpers

HTML

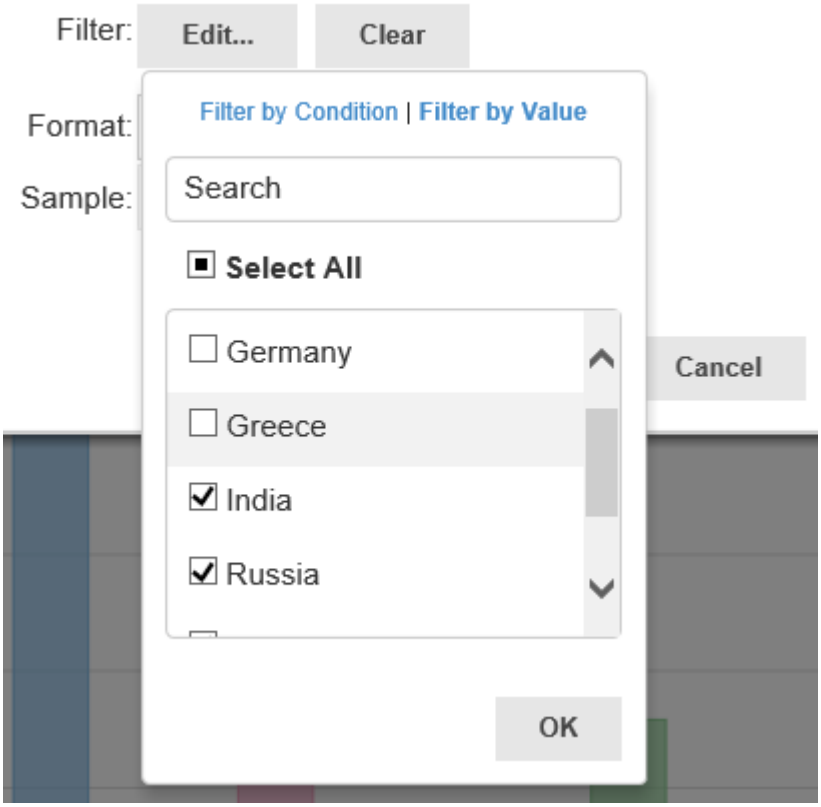
```
@using <ApplicationName>.Models;
@model IEnumerable<ProductData>

<script type="text/javascript">
        c1.documentReady(function () {
            var engine = c1.getService('indexEngine');
            var field = engine.fields.getField("Sales");
            field.aggregate = wijmo.Aggregate.Avg;
            field.format = "n2";
        })
</script>
<c1-pivot-engine id="indexEngine">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-
field-collection>
    <c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-
field-collection>
    <c1-view-field-collection c1-property="ValueFields" items="Sales">
    </c1-view-field-collection>
</c1-pivot-engine>
<c1-pivot-grid items-source-id="indexEngine"></c1-pivot-grid>
```

Filtering Data in Field

In the PivotPanel control, you can filter the data in a field from the **Drag fields between areas below** section of the panel at run time. Each field has two filters: the **value filter**, which allows you to check specific values in a list, and the **range filter**, which allows you to specify one or two criteria. The two filters are independent, and values must pass both the filters in order to be included in the OLAP PivotGrid.

Using the Value Filter

1. Right-click a field in the **Filter**, **Column Fields**, **Row Fields**, or **Values** area.
2. Click **Field Settings** in the context menu. The **Field Settings** dialog box opens.
3. Click the **Edit** button beside the **Filter** option and, then select **Filter by Value** tab. You can clear the selection for any of the fields that you do not want to display in the OLAP table.



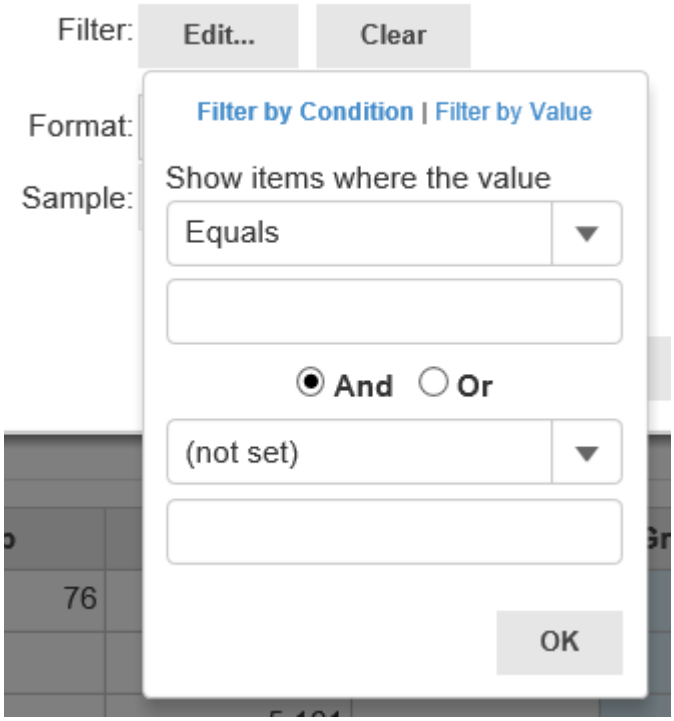
4. Once you have selected the fields to appear in the table, click **OK** to filter the data in the PivotGrid and PivotChart.

Using the Range Filter

1. Right-click a field in the **Filter**, **Column Fields**, **Row Fields**, or **Values** area.
2. Click **Field Settings** in the context menu. The **Field Settings** dialog box opens.
3. Click the **Edit** button beside the **Filter** option and, then select **Filter by Condition** tab. You can clear the selection for any of the fields that you do not want to display in the OLAP table.
4. Select one of the following items from the drop-down list box.

Equals	Create a filter to show items that are equal to the value specified in the field.
Does Not Equal	Create a filter where items that are not the same as the specified value are shown.
Begins With	Create a filter where items that begin with the specified values are shown.
Ends With	Create a filter where items that end with the specified values are shown.
Contains	Create a filter where items that contain the specified values are shown.
Does Not Contain	Create a filter where items that do not contain the specified values are shown.

5. Add an item to filter on, in the first blank text box.



6. Select **And** or **Or**.
7. Add a second filter condition, if necessary. If you select an option other than **None**, the second text box becomes active and you can enter an item.
8. Click **OK** to close the **Custom Filter** dialog box and click **OK** again to close the **Field Settings** dialog box.

Sorting OLAP Data

By default, results in the OLAP output table are sorted by key, for example, "Argentina", "Brazil", and so on. This is not always the most useful way to show the data. To allow this, set the AllowSorting property on the PivotGrid to **True** (default). It allow the users to sort the data by clicking on the column headers, just like a regular grid. Clicking the header repeatedly changes the sort orders from ascending to descending to unsorted.

Country	Aoba	Olap	Wijmo	Xuni	Grand Total
Italy			1,751		1,751
US				1,753	1,753
Germany				4,127	4,127
India				5,664	5,664
Portugal		5,824		839	6,663
China	8,978				8,978
Greece				18,494	18,494
Grand Total	8,978	5,824	1,751	30,877	47,430

Grouping Data

You can use field formatting to group data. Suppose you have a bound list of products and you want to group all of the items ordered together within a particular quarter. You can use the **Field Settings** dialog box at run time or code. This example uses the sample created in the Quick Start topic.

To group data by year (quarter) at run time:

1. Add the following fields to the grid view by selecting them in the PivotPanel: *Date, Product*, and *Downloads*.
2. Right-click the **Date** field, under **Row Fields** and select **Field Settings**. The **Field Settings** dialog box appears.
3. In the **Filter** option, make sure **Select All** option is selected.
4. From the **Format** drop-down list, select **Year Quarter (yyyy "Q"q)** option to group the data in the **PivotGrid** according to quarter.
5. Click **OK** again to close the **Field Settings** dialog box.

The following image displays data where products are grouped by the year (qaurter).

Date	Product	Grand Total
— 2015 Q1	Aoba	2,425
	Olap	4,592
	Wijmo	29
	Subtotal	7,046
— 2015 Q2	Aoba	6,308
	Olap	13,480
	Wijmo	3,443
	Xuni	4,405
	Subtotal	27,636
Grand Total		34,682

Excel Export

The PivotGrid control extends the FlexGrid control, so you can export it to any of the formats supported by the extension modules provided with the FlexGrid. The list of the supported formats includes XLSLX, CSV, and PDF. In the example below, it exports the excel file with two sheets: the current view and a transposed version of the current view.

The image below shows how the OLAP control appears after you implement the code below.

The following code examples demonstrate how export OLAP data to excel file. This example uses the sample created in the Quick Start topic.

In Code (Index.cshtml)

HTML Helpers

Razor

```
@using OlapSample.Models;
@model IEnumerable<ProductData>
<br />
@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("indexEngine")
@Html.C1().PivotChart().ItemsSourceId("indexEngine")
@Html.C1().PivotGrid().Id("indexGrid").ItemsSourceId("indexEngine")

<button type="button" class="btn btn-default" onclick="excelExport()">Export to
XLSX</button>

@section Scripts{
    <script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script type="text/javascript">
         function excelExport() {
            var pivotGrid = wijmo.Control.getControl('#indexGrid');
            // create book with current view
```

```
        var book = wijmo.grid.xlsx.FlexGridXlsxConverter.save(pivotGrid, {
            includeColumnHeaders: true,
            includeRowHeaders: true
        });
        book.sheets[0].name = 'Main View';
        addTitleCell(book.sheets[0], getViewTitle(pivotGrid.engine));

        // add sheet with transposed view
        transposeView(pivotGrid.engine);
        var transposed = wijmo.grid.xlsx.FlexGridXlsxConverter.save(pivotGrid, {
            includeColumnHeaders: true,
            includeRowHeaders: true
        });
        transposed.sheets[0].name = 'Transposed View';
        addTitleCell(transposed.sheets[0], getViewTitle(pivotGrid.engine));
        book.sheets.push(transposed.sheets[0]);
        transposeView(pivotGrid.engine);

        // save the book
        book.save('wijmo.olap.xlsx');
    }
    // build a title for the current view
    function getViewTitle(ng) {
        var title = '';
        for (var i = 0; i < ng.valueFields.length; i++) {
            if (i > 0) title += ', ';
            title += ng.valueFields[i].header;
        }
        title += ' by ';
        if (ng.rowFields.length) {
            for (var i = 0; i < ng.rowFields.length; i++) {
                if (i > 0) title += ', ';
                title += ng.rowFields[i].header;
            }
        }
        if (ng.rowFields.length && ng.columnFields.length) {
            title += ' and by ';
        }
        if (ng.columnFields.length) {
            for (var i = 0; i < ng.columnFields.length; i++) {
                if (i > 0) title += ', ';
                title += ng.columnFields[i].header;
            }
        }
        return title;
    }
    function transposeView(ng) {
        ng.deferUpdate(function () {

            // save row/col fields
            var rows = [],
```

```
                cols = [];
            for (var r = 0; r < ng.rowFields.length; r++) {
                rows.push(ng.rowFields[r].header);
            }
            for (var c = 0; c < ng.columnFields.length; c++) {
                cols.push(ng.columnFields[c].header);
            }
            // clear row/col fields
            ng.rowFields.clear();
            ng.columnFields.clear();

            // restore row/col fields in transposed order
            for (var r = 0; r < rows.length; r++) {
                ng.columnFields.push(rows[r]);
            }
            for (var c = 0; c < cols.length; c++) {
                ng.rowFields.push(cols[c]);
            }
        });
    }
    //adds a title cell into an xlxs sheet
    function addTitleCell(sheet, title) {

        // create cell
        var cell = new wijmo.xlsx.WorkbookCell();
        cell.value = title;
        cell.style = new wijmo.xlsx.WorkbookStyle();
        cell.style.font = new wijmo.xlsx.WorkbookFont();
        cell.style.font.bold = true;

        // create row to hold the cell
        var row = new wijmo.xlsx.WorkbookRow();
        row.cells[0] = cell;

        // and add the new row to the sheet
        sheet.rows.splice(0, 0, row);
    }
</script>
}
```

Tag Helpers

Razor

```
<c1-pivot-engine id="dataSourceEngine" source-key="dataset10">
<c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-collection>
<c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-collection>
<c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-collection>
```

```
</c1-pivot-engine>
<c1-pivot-panel items-source-id="dataSourceEngine"></c1-pivot-panel>
<c1-pivot-chart items-source-id="dataSourceEngine"></c1-pivot-chart>
<c1-pivot-grid items-source-id="dataSourceEngine"></c1-pivot-grid>

<button type="button" class="btn btn-default" onclick="excelExport()">Export to
XLSX</button>
@section Scripts{
    <script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script type="text/javascript">
        function excelExport() {
            var pivotGrid = wijmo.Control.getControl('#indexGrid');
            // create book with current view
            var book = wijmo.grid.xlsx.FlexGridXlsxConverter.save(pivotGrid, {
                includeColumnHeaders: true,
                includeRowHeaders: true
            });
            book.sheets[0].name = 'Main View';
            addTitleCell(book.sheets[0], getViewTitle(pivotGrid.engine));
            // add sheet with transposed view
            transposeView(pivotGrid.engine);
            var transposed = wijmo.grid.xlsx.FlexGridXlsxConverter.save(pivotGrid, {
                includeColumnHeaders: true,
                includeRowHeaders: true
            });
            transposed.sheets[0].name = 'Transposed View';
            addTitleCell(transposed.sheets[0], getViewTitle(pivotGrid.engine));
            book.sheets.push(transposed.sheets[0]);
            transposeView(pivotGrid.engine);
            // save the book
            book.save('wijmo.olap.xlsx');
        }
        // build a title for the current view
        function getViewTitle(ng) {
            var title = '';
            for (var i = 0; i < ng.valueFields.length; i++) {
                if (i > 0) title += ', ';
                title += ng.valueFields[i].header;
            }
            title += ' by ';
            if (ng.rowFields.length) {
                for (var i = 0; i < ng.rowFields.length; i++) {
                    if (i > 0) title += ', ';
                    title += ng.rowFields[i].header;
                }
            }
            if (ng.rowFields.length && ng.columnFields.length) {
                title += ' and by ';
            }
            if (ng.columnFields.length) {
```

```
                for (var i = 0; i < ng.columnFields.length; i++) {
                    if (i > 0) title += ', ';
                    title += ng.columnFields[i].header;
                }
            }
            return title;
        }
        function transposeView(ng) {
            ng.deferUpdate(function () {
                // save row/col fields
                var rows = [],
                    cols = [];
                for (var r = 0; r < ng.rowFields.length; r++) {
                    rows.push(ng.rowFields[r].header);
                }
                for (var c = 0; c < ng.columnFields.length; c++) {
                    cols.push(ng.columnFields[c].header);
                }
                // clear row/col fields
                ng.rowFields.clear();
                ng.columnFields.clear();
                // restore row/col fields in transposed order
                for (var r = 0; r < rows.length; r++) {
                    ng.columnFields.push(rows[r]);
                }
                for (var c = 0; c < cols.length; c++) {
                    ng.rowFields.push(cols[c]);
                }
            });
        }
        //adds a title cell into an xlxs sheet
        function addTitleCell(sheet, title) {
            // create cell
            var cell = new wijmo.xlsx.WorkbookCell();
            cell.value = title;
            cell.style = new wijmo.xlsx.WorkbookStyle();
            cell.style.font = new wijmo.xlsx.WorkbookFont();
            cell.style.font.bold = true;
            // create row to hold the cell
            var row = new wijmo.xlsx.WorkbookRow();
            row.cells[0] = cell;
            // and add the new row to the sheet
            sheet.rows.splice(0, 0, row);
        }
    </script>
}
```

Save and Load View

The following code examples demonstrate how export save and load the view of an OLAP control. This example uses

the sample created in the Quick Start topic.

In Code (Index.cshtml)

HTML Helpers

Razor

```
@using OlapSample.Models;
@model IEnumerable<ProductData>
<br />
@(Html.C1().PivotEngine().Id("indexEngine").Bind(Model)
    .RowFields(pfcb => pfcb.Items("Country"))
    .ColumnFields(cfcb => cfcb.Items("Product"))
    .ValueFields(vfcb => vfcb.Items("Sales")))
@Html.C1().PivotPanel().ItemsSourceId("indexEngine")
@Html.C1().PivotChart().ItemsSourceId("indexEngine")
@Html.C1().PivotGrid().Id("indexGrid").ItemsSourceId("indexEngine")

<button type="button" class="btn btn-default" onclick="saveView()">Save View</button>
<br />
<button type="button" class="btn btn-default" onclick="loadView()">Load View</button>
<br />@section Scripts{
    <script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script type="text/javascript">
        function saveView() {
            var ng = c1.getService('indexEngine');
            if (ng && ng.isViewDefined) {
                localStorage.viewDefinition = ng.viewDefinition;
            }
        }
        function loadView() {
            var ng = c1.getService('indexEngine');
            if (ng && localStorage.viewDefinition) {
                ng.viewDefinition = localStorage.viewDefinition;
                var cmbRowTotals = wijmo.Control.getControl('#RowTotals');
                if (cmbRowTotals) {
                    cmbRowTotals.selectedValue = ng.showRowTotals;
                }

                var cmbColTotals = wijmo.Control.getControl('#ColTotals');
                if (cmbColTotals) {
                    cmbColTotals.selectedValue = ng.showColumnTotals;
                }

                var chkShowZeros = document.getElementById('ColTotals');
                if (chkShowZeros) {
                    chkShowZeros.checked = ng.showZeros;
                }
            }
        }
```

```
    </script>
}
```

Tag Helpers

Razor

```
<c1-pivot-engine id="dataSourceEngine" source-key="dataset10">
<c1-view-field-collection c1-property="RowFields" items="Country"></c1-view-field-
collection>
<c1-view-field-collection c1-property="ColumnFields" items="Product"></c1-view-field-
collection>
<c1-view-field-collection c1-property="ValueFields" items="Sales"></c1-view-field-
collection>
</c1-pivot-engine>
<c1-pivot-panel items-source-id="dataSourceEngine"></c1-pivot-panel>
<c1-pivot-chart items-source-id="dataSourceEngine"></c1-pivot-chart>
<c1-pivot-grid items-source-id="dataSourceEngine"></c1-pivot-grid>

<button type="button" class="btn btn-default" onclick="saveView()">Save View</button>
<br />
<button type="button" class="btn btn-default" onclick="loadView()">Load View</button>
<br />
@section Scripts{
    <script src="http://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js">
</script>
    <script type="text/javascript">
        function saveView() {
            var ng = c1.getService('indexEngine');
            if (ng && ng.isViewDefined) {
                localStorage.viewDefinition = ng.viewDefinition;
            }
        }
        function loadView() {
            var ng = c1.getService('indexEngine');
            if (ng && localStorage.viewDefinition) {
                ng.viewDefinition = localStorage.viewDefinition;
                var cmbRowTotals = wijmo.Control.getControl('#RowTotals');
                if (cmbRowTotals) {
                    cmbRowTotals.selectedValue = ng.showRowTotals;
                }
                var cmbColTotals = wijmo.Control.getControl('#ColTotals');
                if (cmbColTotals) {
                    cmbColTotals.selectedValue = ng.showColumnTotals;
                }
                var chkShowZeros = document.getElementById('ColTotals');
                if (chkShowZeros) {
                    chkShowZeros.checked = ng.showZeros;
                }
            }
        }
```

```
</script>
}
```

OLAP ASP.NET Core Tags

OLAP control supports the following ASP.NET Core Tags:

Pivot Chart

- c1-pivot-grid
- chart-type
- items-source-id
- max-points
- max-series
- show-hierarchical-axes
- show-totals
- stacking

Pivot Engine

- items-source-id
- ItemsSource
- service-url
- source-key
- show-row-totals
- show-column-totals
- totals-before-data
- show-zeros
- default-filter-type
- allow-field-editing
- c1-pivot-field-collection
- c1-view-field-collection
- c1-view-field-collection
- c1-view-field-collection
- c1-view-field-collection
- async
- items-source-changed
- view-definition-changed
- updating-view
- updated-view
- c1-pivot-field-collection
- max-items
- Items (*1)
- Items
- aggregate
- binding
- type
- descending
- dimension-type
- format
- header
- is-content-html
- property-changed

- show-as
- weight-field
- width
- word-wrap

Pivot Grid

- c1-pivot-grid
- allow-resizing
- allow-sorting
- auto-clipboard
- auto-sized-column
- auto-sized-row
- auto-size-mode
- auto-sizing-column
- auto-sizing-row
- center-headers-vertically
- collapsible-subtotals
- custom-context-menu
- format-item
- frozen-columns
- frozen-rows
- headers-visibility
- c1-pivot-grid
- allow-resizing
- allow-sorting
- auto-clipboard
- auto-sized-column
- auto-sized-row
- auto-size-mode
- auto-sizing-column
- auto-sizing-row
- center-headers-vertically
- collapsible-subtotals
- custom-context-menu
- format-item
- frozen-columns
- frozen-rows
- headers-visibility
- show-sort
- sorted-column
- sorting-column
- sort-row-index
- sticky-headers
- updated-layout
- updating-layout
- updated-view
- updating-view

Pivot Panel

- items-source-id
- ItemsSource
- Engine
- Fields

- RowFields
- ColumnFields
- FilterFields
- ValueFields
- items-source-changed
- view-definition-changed
- updating-view
- updated-view

PDF

The **PDF** control allows you to create Adobe PDF documents from your web applications. The commands provided for adding content to documents are similar to the ones available in the .NET Graphics class and you get security, compression, outlining, hyper-linking, and attachments.

PDF for .NET enables you to create Adobe PDF documents from your apps. The benefit of creating PDFs from your .NET apps is that you can essentially create data bound PDF documents.

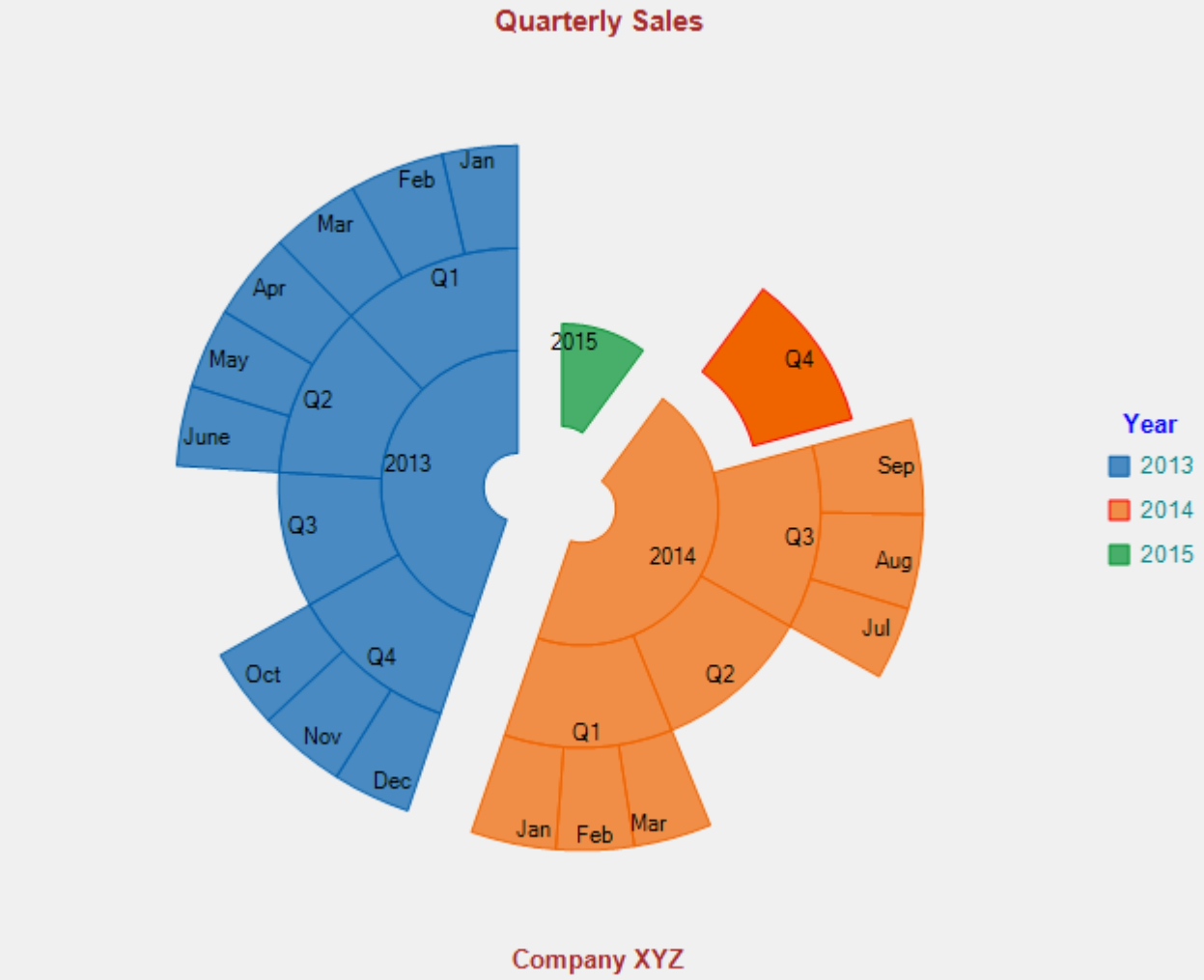
Key Features

- **Fast Rendering and Compression of Images in Metafiles:** Add charts or technical drawings to your PDF document using metafiles as they are better than bitmap images. Metafiles are not converted into bitmaps; they are parsed and converted into vector graphics commands and thus retain the best possible resolution.
- **AcroForms Support:** Use the AddField method to add Acrobat form fields to your PDF documents. This PDF control supports the following field types: textbox, check box, radio button, push button, combo box, and list box.
- **HTML Support:** Use the DrawStringHtml method to render HTML into your PDF documents. You can flow HTML content into multiple pages or columns, use existing style sheets, and mix HTML with other types of content (images, RTF, plain text, form fields, and so on).
- **DrawImage Method:** Use DrawImage method to add an image at a specified location in PDF. DrawImage has parameters that provide control over the image alignment and scaling. You can render any regular .NET Image object in PDF, including metafiles.
- **Manage Document Permissions:** Allow users to copy and edit content, restrict users from printing the document, set annotation edit permission to the user, and more.
- **Attachments:** Add an attachment to your PDF by simply specifying which file you want to attach, what area of the page should contain the attachment, and optionally, the appearance of the attachment.
- **Password Protection:** Encrypt PDF documents containing sensitive information so that only authorized users can access it. There is a separate password for the owner of the document and for all other users. The user's access can be selectively restricted to allow only certain operations, such as viewing, printing, or editing the document.
- **Outlines (bookmarks):** Build outline structure for long PDF documents by adding outline entries (bookmarks) that is displayed on a pane on the left of the reader. The outline makes it easy to browse through a document's structure and find specific topics.
- **Hyperlink Support:** Add hyperlinks and hyperlink targets to your PDF documents. You can also add local links, that when clicked take the user to another location within the same PDF document.
- **Document Metadata and Preferences:** Add meta data to the PDF documents you create. Specify author, creation date, keywords, and so on. You can also provide default viewer preferences to be applied when the document is opened in the Adobe Reader. Specify the initial page layout, window position, as well as reader toolbar and menu visibility.

For more information, refer to PDF for .NET.

Sunburst Chart

Sunburst chart is used to display hierarchical data, represented using concentric circles. The circle in the center represents the root node, with the data moving outside from the center. A section of the inner circle supports a hierarchical relationship to those sections of the outer circle which lie within the angular area of the parent section. Sunburst chart can be effectively used in scenarios where you want to display hierarchical data, represented using relationship between outer rings and inner rings.



Key Features and Properties

- **Donut Chart Support** - Specify the control's inner radius property to support donut charts.
- **Legend**: Provides a short description of data being rendered on chart and you can also change position of the legend as needed.
- **Selection**: Change the selection mode and customize the selected pie slice appearance.
- **Color Themes** - Select different colors from an array of default colors to be used when rendering sunburst slices.
- **Header and Footer**: Specifes the title header and footer text.

For more information on Sunburst chart, see QuickStart: Add data to Sunburst chart.

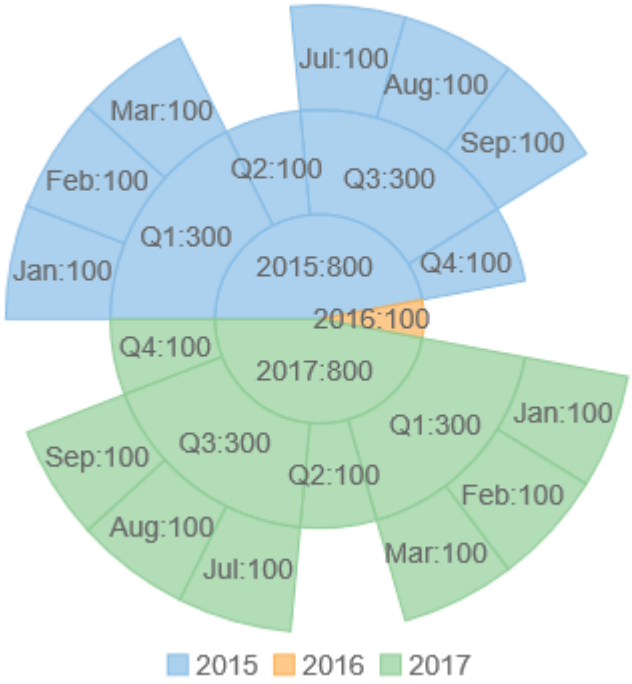
Quick Start: Add data to Sunburst

This section describes how to add a Sunburst chart to your MVC web application and add data to it.

This topic comprises of four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for Sunburst**
- **Step 3: Add a Sunburst chart**
- **Step 4: Build and Run the Project**

The following image shows how Sunburst chart appears after completing the steps:



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for Sunburst

1. Add a new class to the folder **Models** (for example: `HierarchicalData.cs`). See Adding controls to know how to add a new model.
2. Add the following code to the new model to define the classes.

C#

SunburstController.cs

```
public class HierarchicalData
{
 public int ID {get; set;}
 public int Year {get; set;}
 public string Quarter {get; set;}
 public string Month {get; set;}
 public int Value {get; set;}
public HierarchicalData(int year, string quarter, string month, int value)
 {
```

```
  Year = year;
  Quarter = quarter;
  Month = month;
  Value = value;
 }
 public HierarchicalData() {}
 public static List <HierarchicalData> GetData()
 {
  var data = new List <HierarchicalData> ();

  var times = new string[4, 3] {
   {"Jan","Feb","Mar"},
   {"Apr","May","June"},
   {"Jul","Aug","Sep"},
   {"Oct","Nov","Dec"}
  };
  var years = new int[] {2015,2016,2017};

  for (int i = 0; i < years.Length; i++)
  {
   if (i % 2 == 0)
   {
    for (int j = 0; j < 4; j++)
    {
     string quar = "Q" + (j + 1);
     if (j % 2 == 0)
     {
      for (int k = 0; k < 3; k++)
      {
       data.Add(new HierarchicalData(years[i], quar, times[j, k], 100));
      }
     } else

     {
      data.Add(new HierarchicalData(years[i], quar, null, 100));
     }
    }
   } else
   {
    data.Add(new HierarchicalData(years[i], null, null, 100));
   }
  }
  return data;
 }
}
```

Back to Top

Step 3: Add a Sunburst chart

Complete the following steps to initialize a Sunburst chart.

Add a new Controller

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: `SunburstController`).
    3. Click **Add**.
4. Include the MVC references as shown below.

C#

```
using C1.Web.Mvc;
using C1.Web.Mvc.Serializition;
using C1.Web.Mvc.Chart;
```

5. Replace the method Index() with the following method.

C#

SunburstController.cs

```
public ActionResult Index()
        {
            return View(Models.HierarchicalData.GetData());
        }
```

Add a View for the Controller

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `SunburstController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the view name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view is for the controller. Copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Razor

```
@(Html.C1().Sunburst<HierarchicalData>()
.Bind("Year", "Value", Model)
.DataLabel(dl => dl.Content("{name}:{value}").Position(PieLabelPosition.Center))
.Header("2015, 2016, 2017")
.BindingName("Year", "Quarter", "Month")
.Width(500).Height(500))
```

Tag Helpers

Razor

```
<c1-sunburst header="2015, 2016, 2017"
binding-name="Year, Quarter, Month" binding="Value">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-flex-pie-datalabel content="{name}:{value}" position="Center">
</c1-flex-pie-datalabel>
```

```
</c1-sunburst>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**Sunburst/Index**) in the address bar of the browser to see the view.

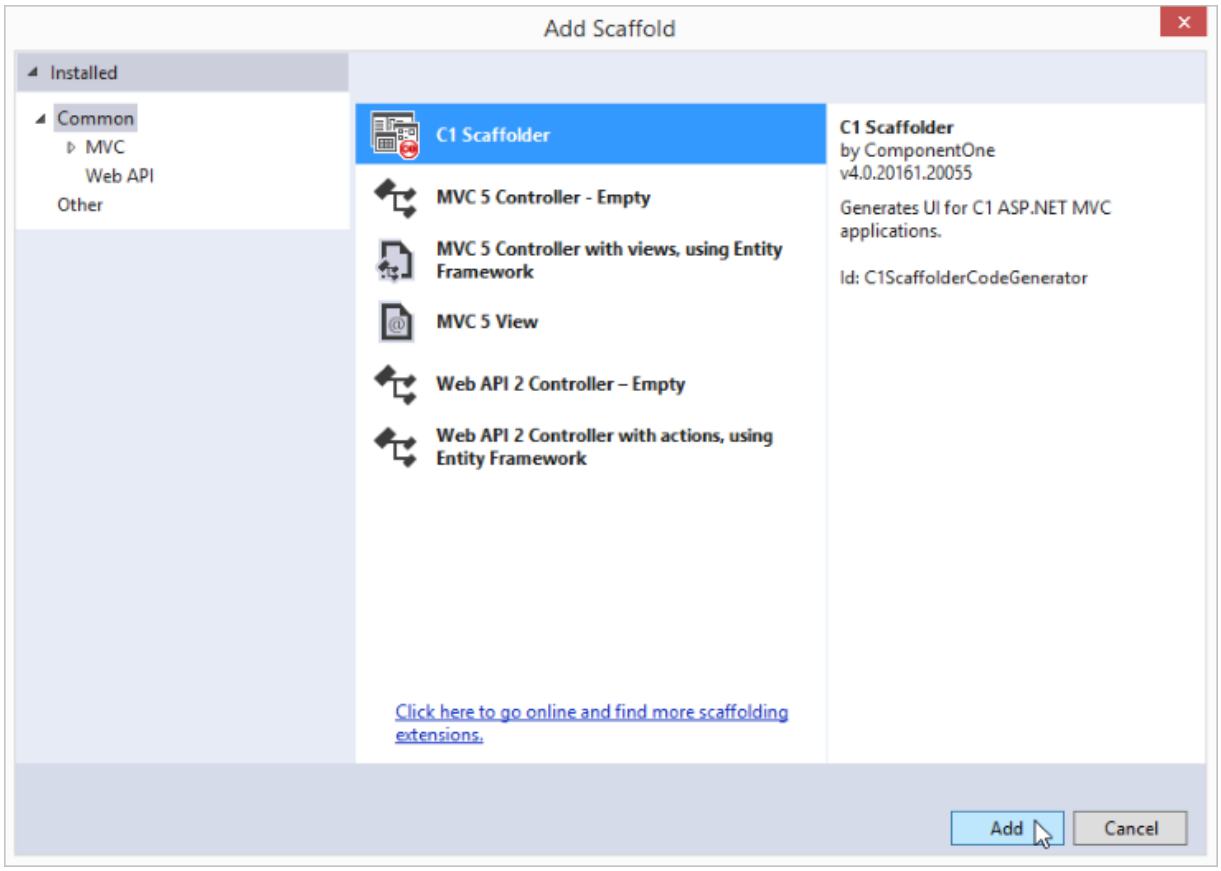
Back to Top

Features

Scaffolding

The steps to scaffold **ComponentOne Sunburst** chart for ASP.NET MVC are as follows:

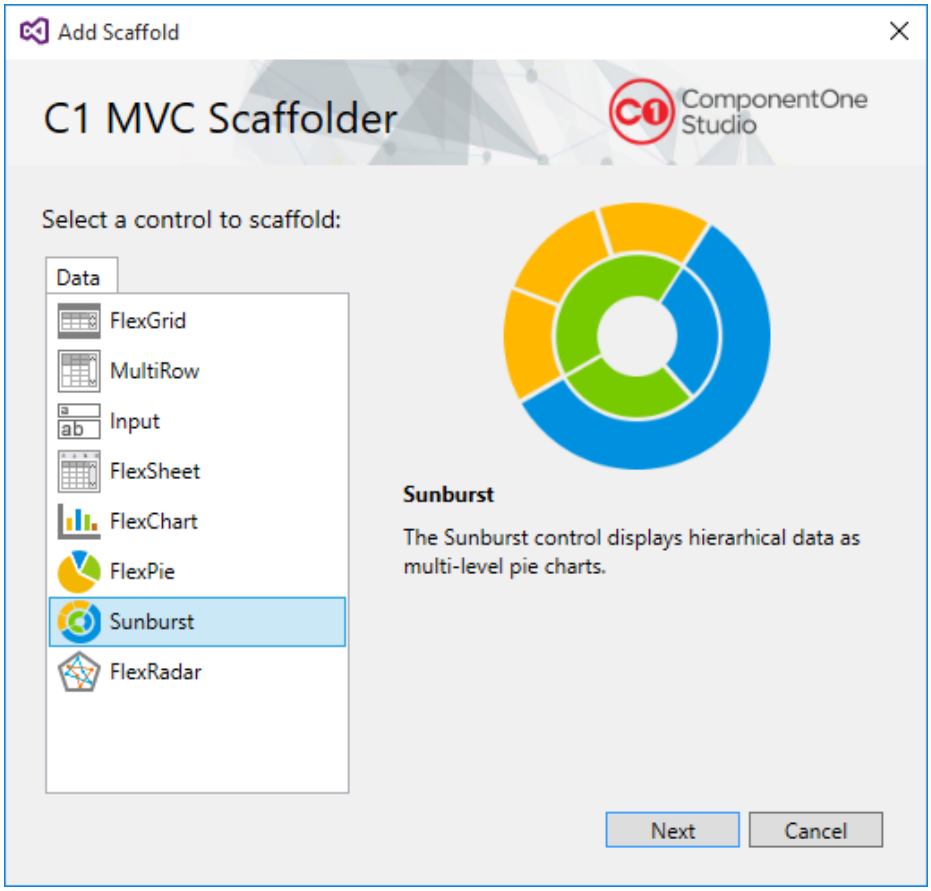
1. Add a new class to the **Models** folder (for example: `HierarchicalData.cs`). For more information on adding a new model, see Quick Start: Add data to Sunburst topic.
2. In the **Solution Explorer**, right-click the project name and select **Add|New Scaffolded Item**. The Add Scaffold wizard appears.
3. In the **Add Scaffold** wizard, select **Common** and then select **C1 Scaffolder** from the right pane.



You can also select **Common|MVC|Controller** or **Common|MVC|View** and then **C1 Scaffolder** to add only a controller or a view.

4. Click **Add**.

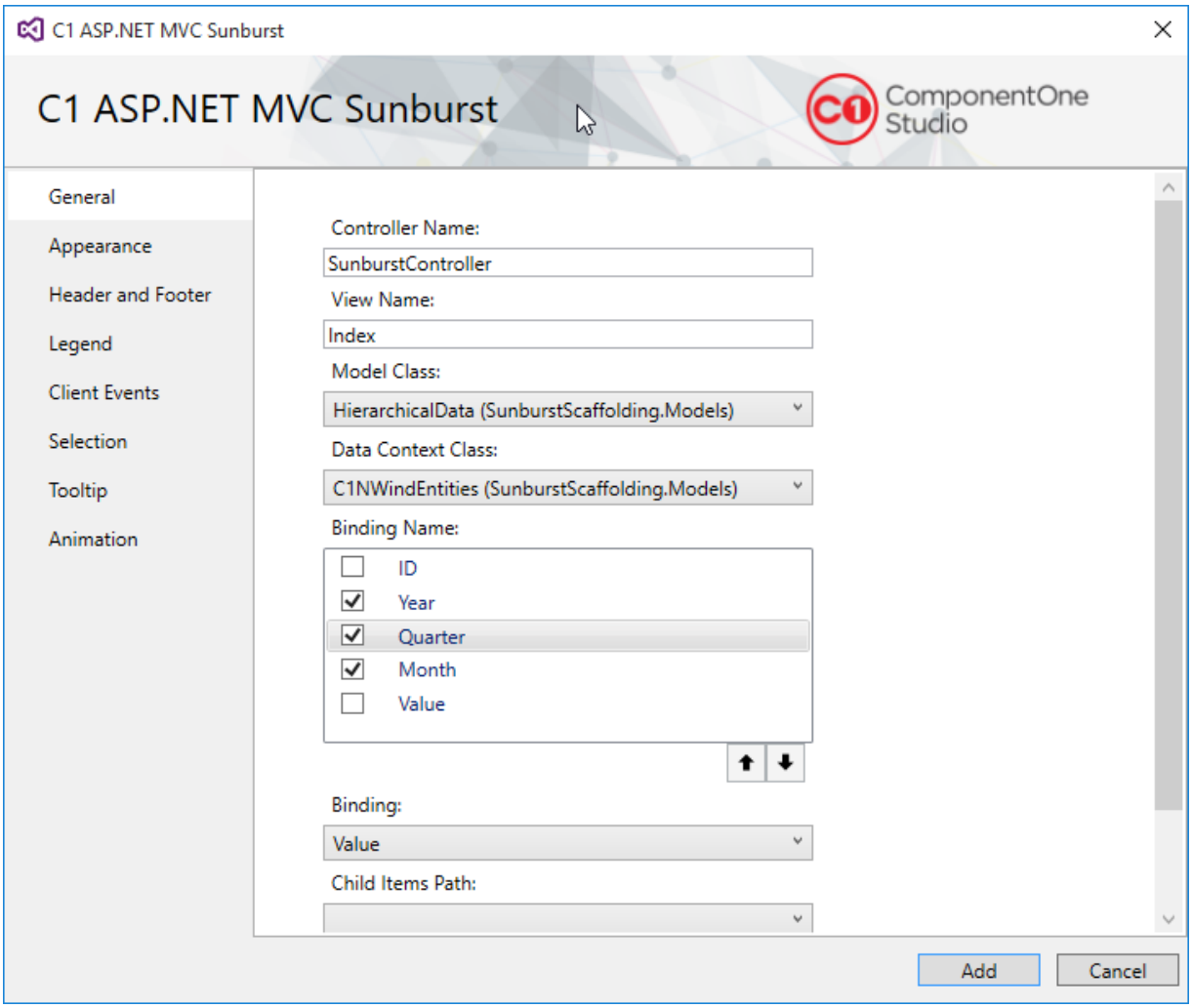
5. In the **Add Scaffold** dialog, select Sunburst control, and then click **Next**.



The **C1 ASP.NET MVC Sunburst** wizard appears with the General tab selected by default.

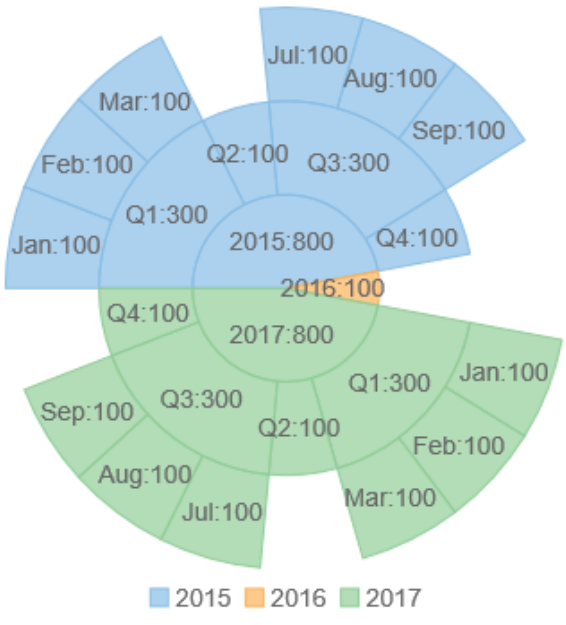
6. In the General tab, specify the model details as follows:

    1. Enter the **Controller Name** and **View Name**.
    2. Select **Model Class** from the drop-down list. In our case, we select **HierarchicalData** to populate data in Sunburst chart.
    3. Select **Data Context Class** from the drop-down list. In our case, we select **C1NWindEntities**.
    4. In the **Binding Name** section, select **Year**, **Quarter**, and **Month** check boxes.
    5. In the **Binding** drop-down, select a column to bind it with your Sunburst chart. In our case, we have selected **Value**.



7. Click **Add**. You will see that the Controller and View for the selected model are added to your project. Once the code is generated, you can run the project using the **F5** button.

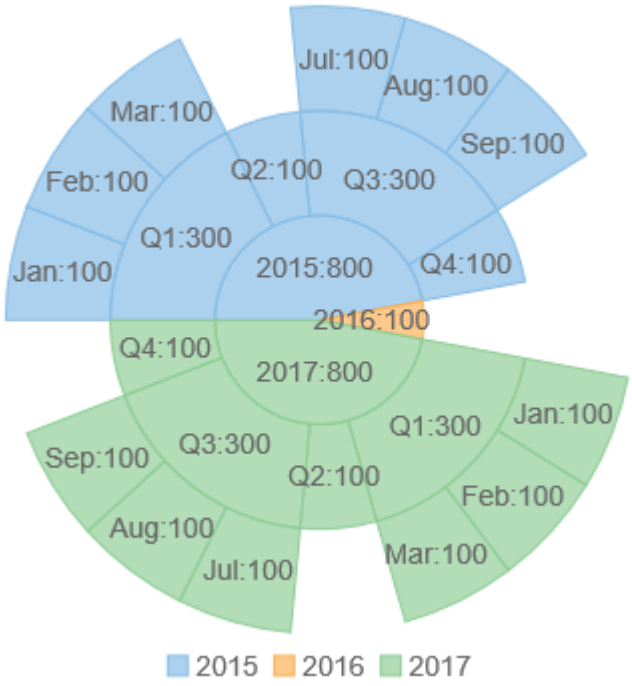
Output



Legend

You can specify the position where you want to display the legend using the Legend property of the Sunburst chart. Legend helps in displaying the series of a chart with a predefined symbol and name of the series.

The image below shows how the Sunburst chart appears after you set the Chart.Position property to **Bottom**.



The following code example demonstrates how to set the Position property. This example uses the sample created in the Quick Start topic.

In Code

HTML Helpers

Razor

```
.Legend(C1.Web.Mvc.Chart.Position.Bottom)
```

Tag Helpers

Razor

```
<c1-sunburst binding-name="Year, Quarter, Month" binding="Value" legend-
position="Bottom">
<c1-items-source source-collection="Model"></c1-items-source>
</c1-sunburst>
```

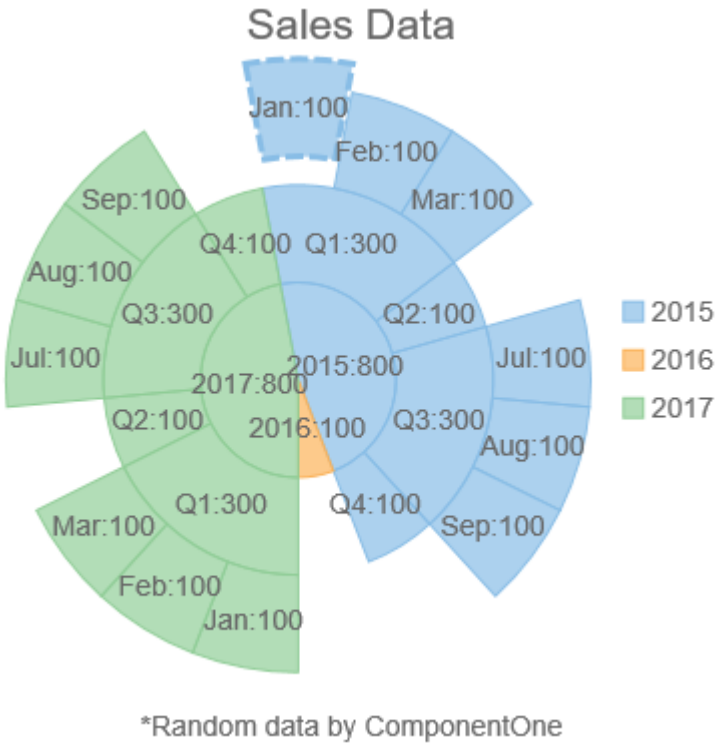
Selection

You can choose which element of the Sunburst chart should be selected when the user clicks on any region in Sunburst chart by setting the `SelectionMode` property. The three different options provided are as follows:

- **None**: Does not select any element.
- **Point**: Highlights the pie slice that the user clicks.
- **Series**: Highlights the entire pie.

When the `SelectionMode` is set to **Point**, you can change the position of the selected sunburst slice by setting the `SelectedItemPosition` property. Also, you can set the `SelectedItemOffset` property to move the selected sunburst slice away from the center.

The image below show how the Sunburst chart appears after you set the **SelectionMode** property.



The following code example demonstrates how to set the SelectionMode property. This example uses the sample created in the Quick Start topic.

In Code

HTML Helpers

Razor

```
.SelectionMode(C1.Web.Mvc.Chart.SelectionMode.Point)
.SelectedItemPosition(C1.Web.Mvc.Chart.Position.Top)
.SelectedItemOffset(0.1f)
```

Tag Helpers

Razor

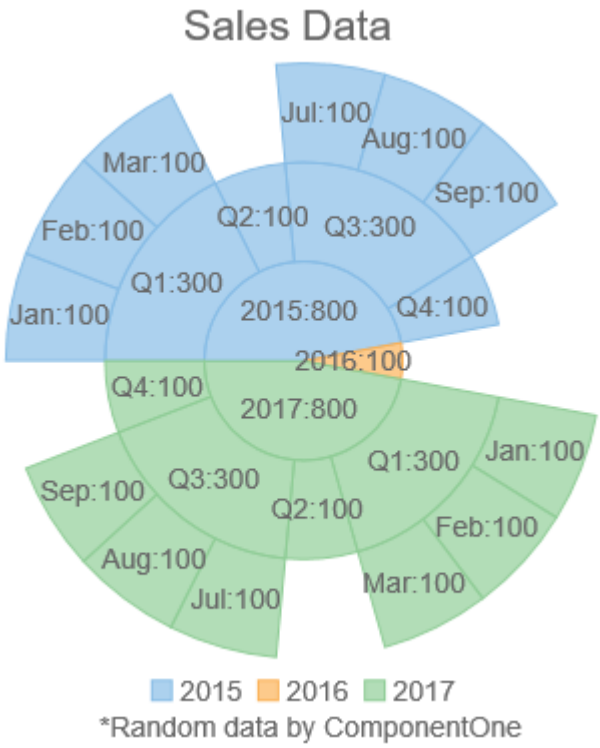
```
<c1-sunburst binding-name="Year, Quarter, Month" binding="Value"
```

```
            selection-mode="Point"
            selected-item-position="Top"
            selected-item-offset="0.1f">
<c1-items-source source-collection="Model"></c1-items-source>
<c1-flex-pie-datalabel content="{name}:{value}" position="Center">
</c1-flex-pie-datalabel>
</c1-sunburst>
```

Header and Footer

You can provide a heading or title to the Sunburst chart with the help of Header property. Moreover, you may also set a footer for your Sunburst chart with the help of Footer property. These properties also allows you to style your header and footer text with the help of HeaderStyle and FooterStyle properties.

The image below shows how the Sunburst chart appears, after you set the Header and Footer properties.



The following code example demonstrates how to set Header and Footer properties. This example uses the sample created in the Quick Start topic.

HTML Helpers

Razor

```
.Header("Sales Data")
.HeaderStyle(style => style.FontSize("30"))
.Footer("*Random data by ComponentOne")
.FooterStyle(style => style.FontSize("15"))
```

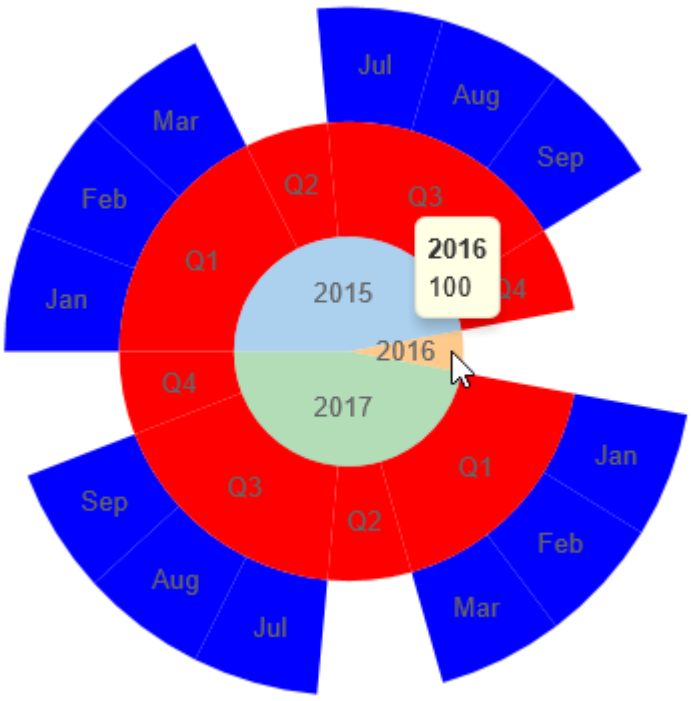
Tag Helpers

```
Razor
<c1-sunburst header="Sales Data" footer="*Random data by ComponentOne"
binding-name="Year, Quarter, Month" binding="Value">
<c1-flex-chart-title-style c1-property="FooterStyle"></c1-flex-chart-title-style>
<c1-flex-chart-title-style c1-property="HeaderStyle"></c1-flex-chart-title-style>
<c1-items-source source-collection="Model"></c1-items-source>
</c1-sunburst>
```

Theming

The appearance of the Sunburst chart control is largely defined in CSS. In addition to the default theme, we include several professionally designed themes to enable customizing the appearance of all MVC controls to achieve a consistent and attractive look. You can customize the appearance of the Sunburst chart control using CSS. To do this, copy the CSS rules from the default theme to a new CSS file and modify the properties as needed.

In this example, we have added the "**custom-sunburst-chart**" CSS class to the control and defined CSS rules to change the fill, font family, and font weight of the header and fill color of the slices. The below example code uses **HierarchicalData.cs** model added in the QuickStart section.



In Code

HTML Helpers

```
Razor
@using C1.Web.Mvc.Chart
<style type="text/css">
        .custom-sunburst-chart.wj-sunburst .wj-header .wj-title {
```

```
            fill: #666;
            font-family: 'Courier New', Courier, monospace;
            font-weight: bold;
        }

        .custom-sunburst-chart.wj-sunburst ellipse,
        .custom-sunburst-chart.wj-sunburst path {
            stroke-width: 0;
        }

        .custom-sunburst-chart.wj-sunburst ellipse {
            fill: yellow;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level2 > path {
            fill: red;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level3 > path {
            fill: blue;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level4 > path {
            fill: black;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level5 > path {
            fill: white;
            stroke-width: 2;
            stroke: black;
        }
</style>
@using SunburstSample.Models;
@model IEnumerable<HierarchicalData>

@(Html.C1().Sunburst<HierarchicalData>()
    .Bind("Year", "Value", Model)
    .DataLabel(dl => dl.Content("{name}").Position(PieLabelPosition.Center))
    .BindingName("Year, Quarter, Month")
    .CssClass("custom-sunburst-chart"))
```

Tag Helpers

Razor

```
@using C1.Web.Mvc.Chart
<style type="text/css">
        .custom-sunburst-chart.wj-sunburst .wj-header .wj-title {
            fill: #666;
            font-family: 'Courier New', Courier, monospace;
            font-weight: bold;
```

```
        }

        .custom-sunburst-chart.wj-sunburst ellipse,
        .custom-sunburst-chart.wj-sunburst path {
            stroke-width: 0;
        }

        .custom-sunburst-chart.wj-sunburst ellipse {
            fill: yellow;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level2 > path {
            fill: red;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level3 > path {
            fill: blue;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level4 > path {
            fill: black;
        }

        .custom-sunburst-chart.wj-sunburst .slice-level5 > path {
            fill: white;
            stroke-width: 2;
            stroke: black;
        }
</style>
@using <ApplicationName.Models>
@model IEnumerable<HierarchicalData>

<c1-sunburst header="Sales" class="custom-sunburst-chart"
             binding-name="Year, Quarter, Month" binding="Value">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-pie-datalabel content="{name}" position="Center"></c1-flex-pie-
datalabel>
</c1-sunburst>
```

Sunburst ASP.NET Core Tags

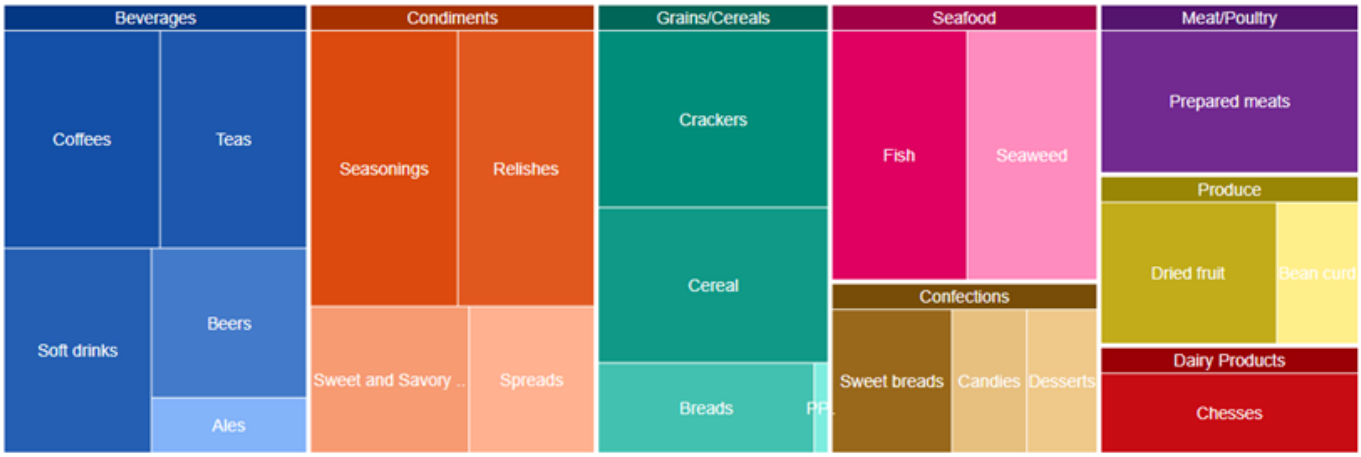
Sunburst Chart control supports the following ASP.NET Core Tags:

- c1-sunburst
- binding-name
- child-items-path

TreeMap

The TreeMap control display hierarchical data as a set of nested rectangles. Hierarchical data are useful in varied walks

of life and setups, be it family tree, programming, organization structure, or directories. Visualizing such a data and spotting information in them is a difficult task, especially if the data is huge. TreeMap control enables visualization of hierarchical data as nested rectangles on a limited space. It is useful in having a quick glimpse of patterns in large data and comparing proportions.



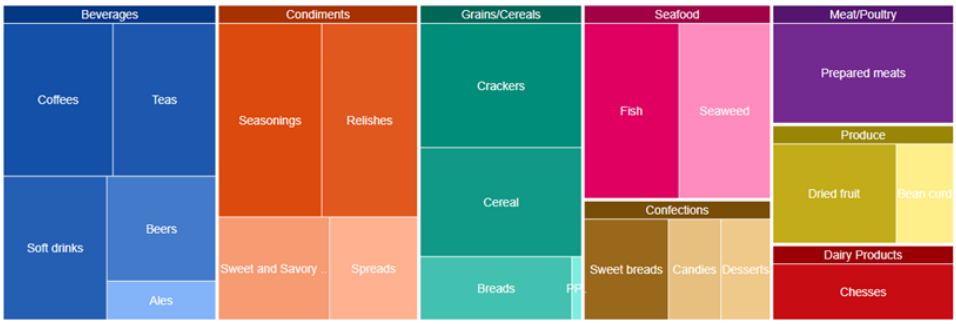
TreeMap control supports data binding to show hierarchy, and allows user to drill down the data further to numerous levels for detailed analysis. The control can be customized to display data in horizontal, vertical, and squarified layouts of constituting rectangles.

Quick Start: Add Data to TreeMap

The topic describes how to add TreeMap control to your MVC web application and add data to it using model binding. This topic comprises following four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for TreeMap**
- **Step 3: Add a TreeMap Control**
- **Step 4: Build and Run the Project**

The following image shows TreeMap control after completing above the steps.



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for TreeMap

1. Add a new class to the **Models** folder (Name: `FoodSale.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `FoodSale.cs` model. We are using **`FoodSale`** class to represent a list of hierarchical data.

C#

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

namespace TreeMapQuickStart.Models
{
    public class FoodSale
    {
        private static List<string> Categories = new List<string> { "Beverages", "Condiments", "Confections", "Dairy
Products", "Grains/Cereals", "Meat/Poultry", "Produce", "Seafood" };
```

```
        private static List<string[]> SubCategories = new List<string[]>
        {
            new string[] { "Soft drinks", "Coffees", "Teas", "Beers", "Ales" },
            new string[] { "Sweet and Savory sauces", "Relishes", "Spreads", "Seasonings" },
            new string[] { "Desserts", "Candies", "Sweet breads" },
            new string[] { "Chesses" },
            new string[] { "Breads", "Crackers", "Pasta", "Cereal" },
            new string[] { "Prepared meats" },
            new string[] { "Dried fruit", "Bean curd" },
            new string[] { "Seaweed", "Fish" }
        };
        public string Category { get; set; }
        public string SubCategory { get; set; }
        public double Sales { get; set; }
        public static IEnumerable<FoodSale> GetData()
        {
            var result = new List<FoodSale>();
            var rand = new Random(0);
            var index = 0;
            foreach (var cat in Categories)
            {
                var subCategories = SubCategories[index++];
                foreach (var subCat in subCategories)
                {
                    result.Add(new FoodSale
                    {
                        Category = cat,
                        SubCategory = subCat,
                        Sales = rand.NextDouble() * 100
                    });
                }
            }
            return result;
        }
        public static IEnumerable<FoodSale> GetGroupData()
        {
            var result = new List<FoodSale>();
            var rand = new Random(0);
            var catLen = Categories.Count;
            for (var i = 0; i < 1000; i++)
            {
                var randomC = rand.Next(0, catLen - 1);
                var subCat = SubCategories[randomC];
                var randomSC = rand.Next(0, subCat.Length - 1);
                result.Add(new FoodSale
                {
                    Category = Categories[randomC],
                    SubCategory = subCat[randomSC],
                    Sales = rand.NextDouble() * 100
                });
            }
            return result;
        }
    }
}
```

Back to Top

Step 3: Add a TreeMap Control

Steps to add a TreeMap control to the application, are as follows:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (For example: `TreeMapController`).
    3. Click **Add**.
4. Include the following references as shown below.

C#

```
using <ApplicationName>.Models;
```

5. Replace the **Index()** method with the following method.

TreeMapController.cs

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

Add a View for the Controller

In the view, we create an instance of TreeMap and bind it to a data source using **Bind** method. The **Type** property allows you to select the type of TreeMap you want to use in the application.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `TreeMapController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller, and then copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml

```
@using <ApplicationName>.Models;
@using System.Drawing;
@model IEnumerable<FoodSale>

@(Html.C1().TreeMap().Id("TreeMap")
    .Type(TreeMapType.Squarified)
    .Binding("Sales")
    .BindingName("Category", "SubCategory")
    .Bind(Model)
    .DataLabel(dlb => dlb.Position(C1.Web.Mvc.Chart.LabelPosition.Center).Content("{name}")))
```

Tag Helpers

Index.cshtml

```
@model IEnumerable<FoodSale><c1-tree-map id="TreeMap" binding="Sales" binding-name="Category,SubCategory">
<c1-items-source source-collection="Model"></c1-items-source> \
<c1-flex-chart-datalabel position="Center" content="{name}">
</c1-flex-chart-datalabel>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**TreeMap/Index**) in the address bar of the browser to see the view.

Back to Top

Key Features

TreeMap provides many different features that enable the developers to build intuitive and professional-looking applications. The main features for TreeView are as follows:

- **Multiple Layouts**
  Multiple Layouts TreeMap supports multiple display arrangements, where the tree branches can be shown as squares, horizontal rectangles or vertical rectangles.

- **Customize Appearance**
  TreeMap enables users to stylize the control and modify its appearance as per their preference. A set of varied color palettes are available to clearly display categories in a TreeMap.

- **Custom Hierarchical Levels**
  TreeMap enables users to vary the depth of data to be visualized and further drill down (or reverse drill down)

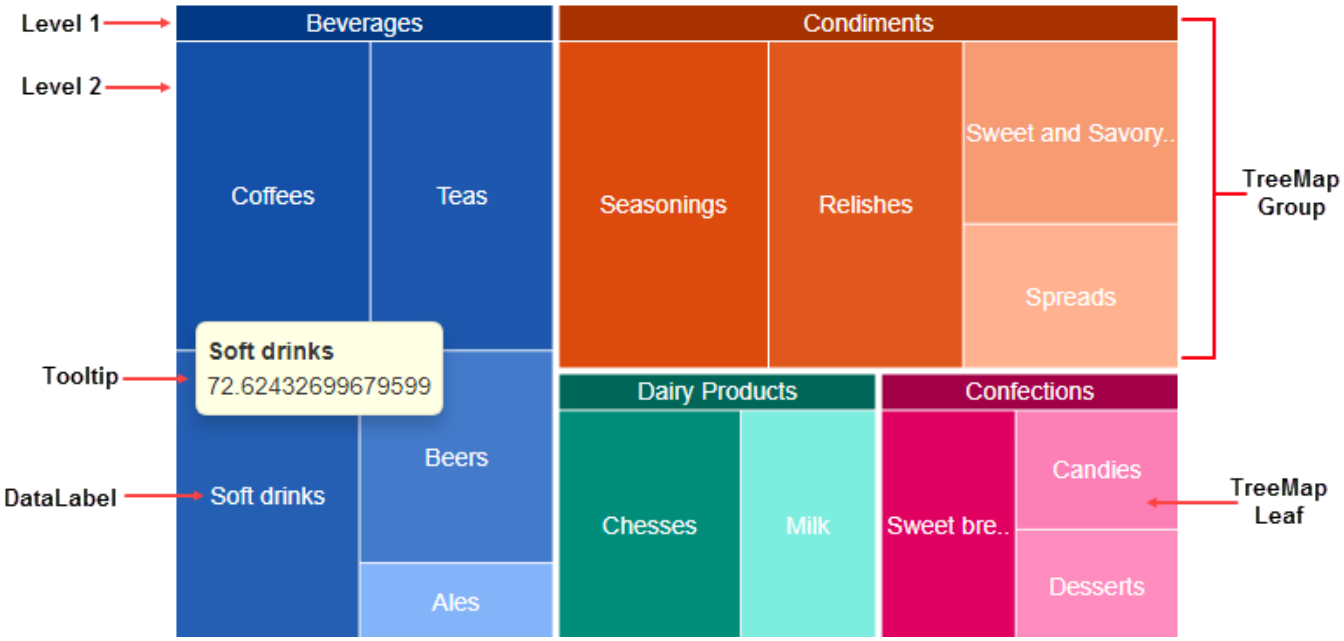
the data for analysis and comparison.

- **Space Utilization**
  TreeMap is ideal for compact display and visualization of huge data. The nested rectangles and groups constituting the TreeMap adjust their size to fit the display area.

Elements

The TreeMap control is composed of rectangles, representing individual data items, which are grouped into categories, to represent the hierarchical nature of data. The individual data items which make group are known as leaf nodes. The sizes of these nodes are proportional to the data they represent.

The following image exhibits main elements of TreeMap control.

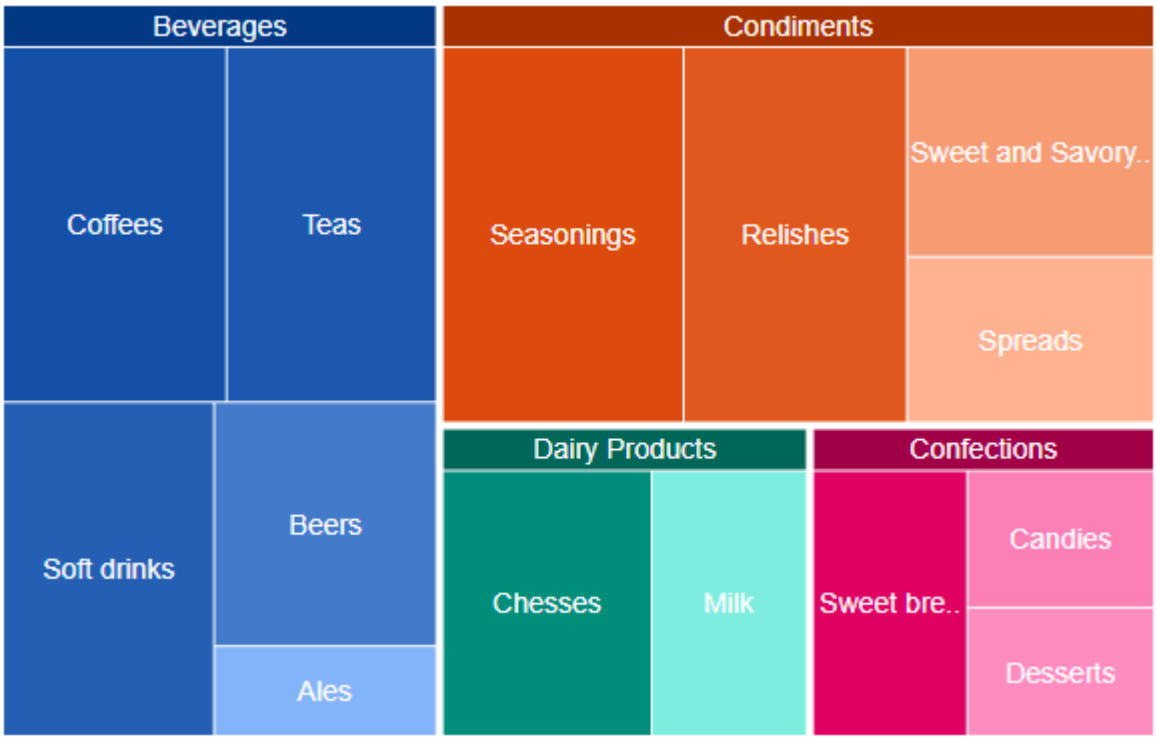


Layouts

TreeMap enables its data items and groups, represented as rectangles, to be displayed in a variety of arrangements. The tree map rectangles can be arranged into squarified, horizontal, and vertical layouts. To set the desired tree map layout, you need to use Type property of **TreeMap** class, which takes the value from TreeMapType Enum. The default layout of the TreeMap chart control is squarified.

Squarified

The squarified layout tries to arrange the tree map rectangles (data items and groups) as approximate squares. This layout makes it easier to make comparisons and point patterns, as the accuracy of presentation is enhanced in squarified arrangement. This layout is very useful for large data sets.

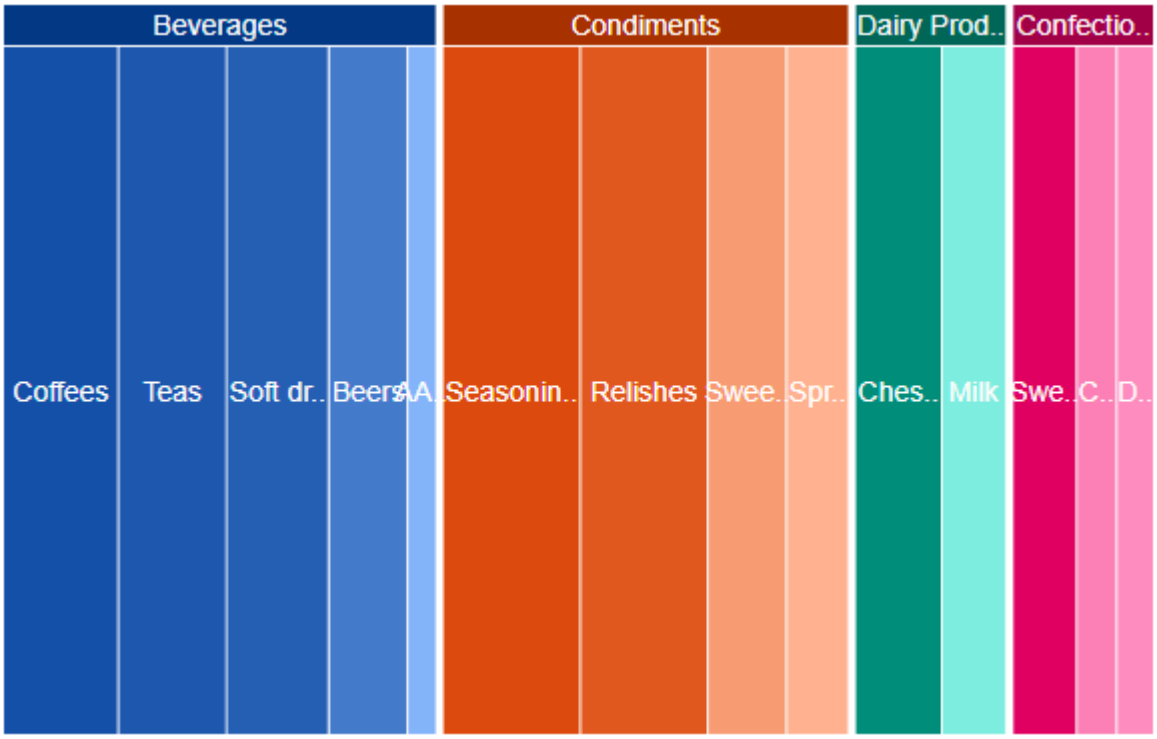


Horizontal

The horizontal layout stacks the tree map rectangles one over the other as rows. Here the width of the rectangles is greater than their height.

Vertical

The vertical layout arranges the tree map rectangles adjacent to each other as columns. Here the height of the rectangles is greater than their width.



Features

Learn about concepts that help you to understand how best to use TreeMap control.

This section contains information about

Max Depth
: Learn how to set the MaxDepth property in the TreeView control.

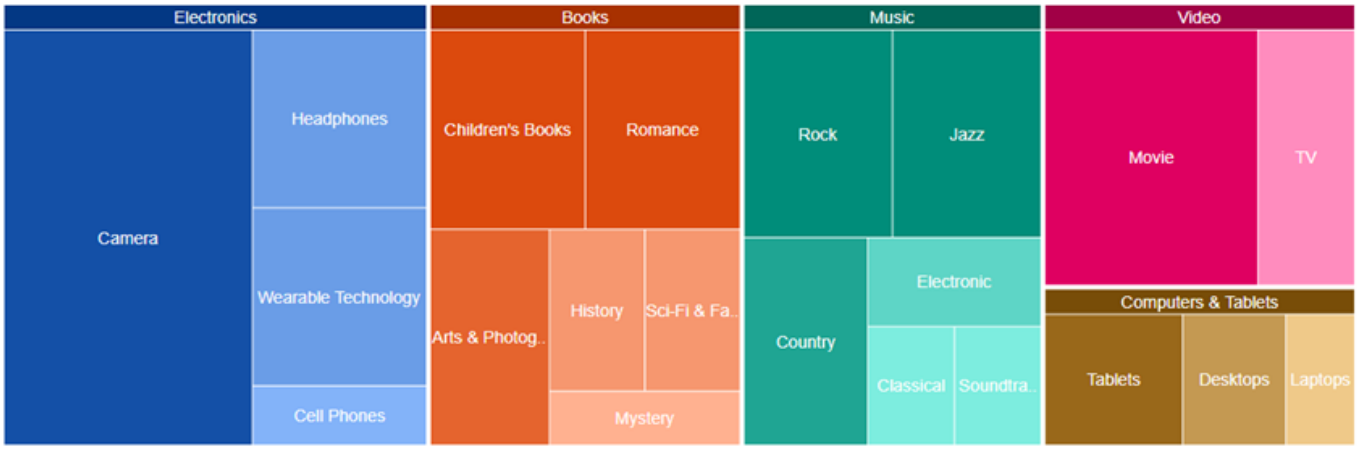
Group Collection
: Learn how to implement grouping in the TreeMap control.

Themeing
: Learn how to apply different themes in the TreeMap control.

Max Depth

In the TreeMap control, you can define how many levels you want to be shown simultaneously on the control. It's possible to show as many levels of hierarchy as per your requirements. Use MaxDepth method with a number of levels you want to see as an argument. Note that the more levels you show the less understandable your TreeMap might become (depends on the levels' number and values they represent). In our example, we will set MaxDepth property to 2.

The following image shows how a TreeMap control appears after setting the MaxDepth property to 2.



The following code examples demonstrate how to set the MaxDepth property in a TreeMap. The example uses **ThingSale.cs** model.

In Code

ThingSale.cs

C#

```
public class ThingSale
    {
        private static List<string> Categories = new List<string> { "Music", "Video",
"Books", "Electronics", "Computers & Tablets" };
        private static Dictionary<string, List<string>> AllCategories = new
Dictionary<string, List<string>>();
        public string Category { get; set; }

        public double? Sales { get; set; }
        public List<ThingSale> Items { get; set; }

        public static void EnsureInitAllCategories()
        {
            if (AllCategories.Count > 0)
            {
                return;
            }

            AllCategories.Add("Music", new List<string> { "Country", "Rock",
"Classical", "Soundtracks", "Jazz", "Electronic" });
            AllCategories.Add("Country", new List<string> { "Classic Country",
"Cowboy Country", "Outlaw Country", "Western Swing", "Roadhouse Country" });
            AllCategories.Add("Rock", new List<string> { "Hard Rock", "Blues Rock",
"Funk Rock", "Rap Rock", "Guitar Rock", "Progressive Rock" });
            AllCategories.Add("Classical", new List<string> { "Symphonies", "Chamber
Music" });
            AllCategories.Add("Soundtracks", new List<string> { "Movie Soundtracks",
"Musical Soundtracks" });
            AllCategories.Add("Jazz", new List<string> { "Smooth Jazz", "Vocal Jazz",
"Jazz Fusion", "Swing Jazz", "Cool Jazz", "Traditional Jazz" });
            AllCategories.Add("Electronic", new List<string> { "Electronica",
```

```
"Disco", "House" });

            AllCategories.Add("Video", new List<string> { "Movie", "TV" });
            AllCategories.Add("Movie", new List<string> { "Kid & Family", "Action &
Adventure", "Animation", "Comedy", "Drama", "Romance" });
            AllCategories.Add("TV", new List<string> { "Science Fiction",
"Documentary", "Fantasy", "Military & War", "Horror" });

            AllCategories.Add("Books", new List<string> { "Arts & Photography",
"Children's Books", "History", "Mystery", "Romance", "Sci-Fi & Fantasy" });
            AllCategories.Add("Arts & Photography", new List<string> {
"Architecture", "Graphic Design", "Drawing", "Photography", "Performing Arts" });
            AllCategories.Add("Children's Books", new List<string> { "Beginning
Readers", "Board Books", "Chapter Books", "Coloring Books", "Picture Books", "Sound
Books" });
            AllCategories.Add("History", new List<string> { "Ancient", "Medieval",
"Renaissance" });
            AllCategories.Add("Mystery", new List<string> { "Thriller & Suspense",
"Mysteries" });
            AllCategories.Add("Romance", new List<string> { "Action & Adventure",
"Holidays", "Romantic Comedy", "Romantic Suspense", "Western", "Historical" });
            AllCategories.Add("Sci-Fi & Fantasy", new List<string> { "Fantasy",
"Gaming", "Science Fiction" });

            AllCategories.Add("Electronics", new List<string> { "Camera",
"Headphones", "Cell Phones", "Wearable Technology" });
            AllCategories.Add("Camera", new List<string> { "Digital Cameras", "Film
Photography", "Lenses", "Video", "Accessories" });
            AllCategories.Add("Headphones", new List<string> { "Earbud headphones",
"Over-ear headphones", "On-ear headphones", "Bluetooth headphones", "Noise-cancelling
headphones", "Audiophile headphones" });
            AllCategories.Add("Cell Phones", new List<string> { "Cell Phone",
"Accessories" });
            AllCategories.Add("Accessoriess", new List<string> { "Batteries",
"Bluetooth Headsets", "Bluetooth Speakers", "Chargers", "Screen Protectors" });
            AllCategories.Add("Wearable Technology", new List<string> { "Activity
Trackers", "Smart Watches", "Sports & GPS Watches", "Virtual Reality Headsets",
"Wearable Cameras", "Smart Glasses" });

            AllCategories.Add("Computers & Tablets", new List<string> { "Desktops",
"Laptops", "Tablets" });
            AllCategories.Add("Desktops", new List<string> { "All-in-ones", "Minis",
"Towers" });
            AllCategories.Add("Laptops", new List<string> { "2 in 1 laptops",
"Traditional laptops" });
            AllCategories.Add("Tablets", new List<string> { "IOS", "Andriod", "Fire
OS", "Windows" });
        }
        public static IEnumerable<ThingSale> GetData()
        {
            EnsureInitAllCategories();
```

```
            var result = new List<ThingSale>();
            Categories.ForEach(cat =>
            {
                result.Add(Create(cat));
            });

            return result;
        }

        private static ThingSale Create(string category)
        {
            var rand = new Random(0);
            var item = new ThingSale { Category = category };
            if (!AllCategories.ContainsKey(category))
            {
                item.Sales = rand.NextDouble() * 100;
            }
            else
            {
                item.Items = new List<ThingSale>();
                AllCategories[category].ForEach(subCat =>
                {
                    item.Items.Add(Create(subCat));
                });
            }
            return item;
        }
    }
```

TreeMapController.cs

C#

```
public class TreeMapController : Controller
    {
        // GET: TreeMap
        public ActionResult Index()
        {
            return View(ThingSale.GetData());
        }
    }
```

MaxDepth.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@using System.Drawing;
@model IEnumerable<ThingSale>
```

```
@(Html.C1().TreeMap().Id("TreeMap")
    .Binding("Sales")
    .BindingName("Category")
    .ChildItemsPath("Items")
    .Bind(Model)
    .MaxDepth(2)
    .DataLabel(dlb => dlb.Position(C1.Web.Mvc.Chart.LabelPosition.Center).Content("
{name}")))
```

Tag Helpers

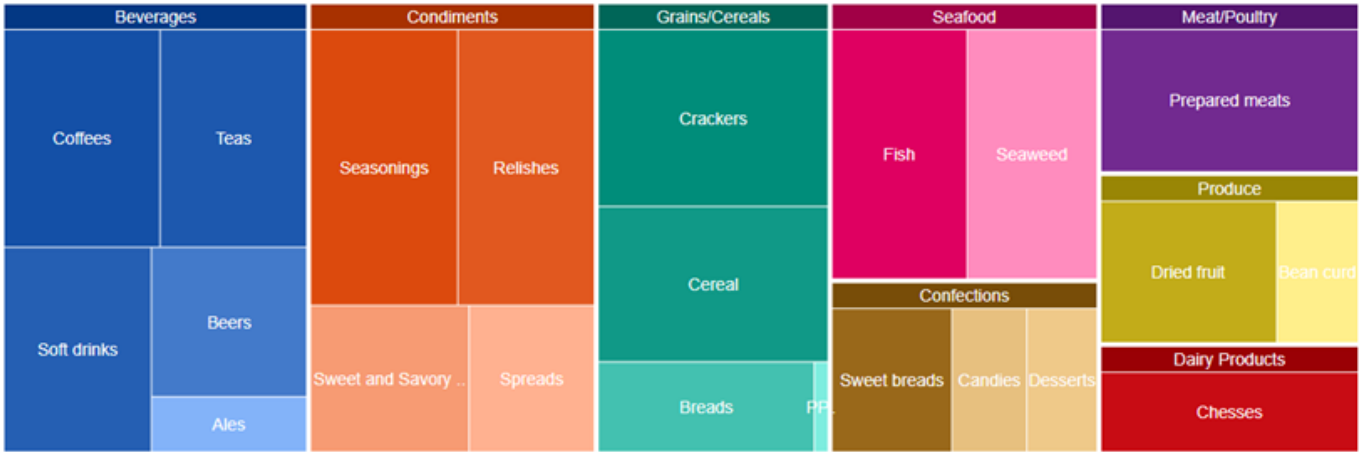
HTML

```
@using <ApplicationName.Models>
@using System.Drawing;
@model IEnumerable<ThingSale>

<c1-tree-map id="TreeMap"
             binding="Sales" binding-name="Category"
             child-items-path="Items"
             max-depth="2">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-datalabel position="Center" content="{name}"></c1-flex-chart-
datalabel>
</c1-tree-map>
```

Group Collection

The TreeMap control allows you to group the data to display the contents with better visualization. You can configure a group in TreeMap using the GroupBy property in view. The grouped TreeMap item represents a collection of leaf tree map items grouped by some value. The GroupHeader allows you to identify by which value leaf items are grouped in the TreeMap control.

The following image shows how a TreeMap control appears after setting the **GroupBy** property.



The below example code uses FoodSale.cs model added in the QuickStart section.

In Code

Grouping.cshtml

HTML Helpers

Index.cshtml

```
@using <ApplicationName>.Models;
@using System.Drawing;
@model IEnumerable<FoodSale>

@(Html.C1().TreeMap()
    .Binding("Sales")
    .BindingName("Category", "SubCategory")
    .Bind(isb => isb.Bind(Model).GroupBy("Category", "SubCategory"))
    .DataLabel(dlb => dlb.Position(C1.Web.Mvc.Chart.LabelPosition.Center).Content("
{name}")))
```

Tag Helpers

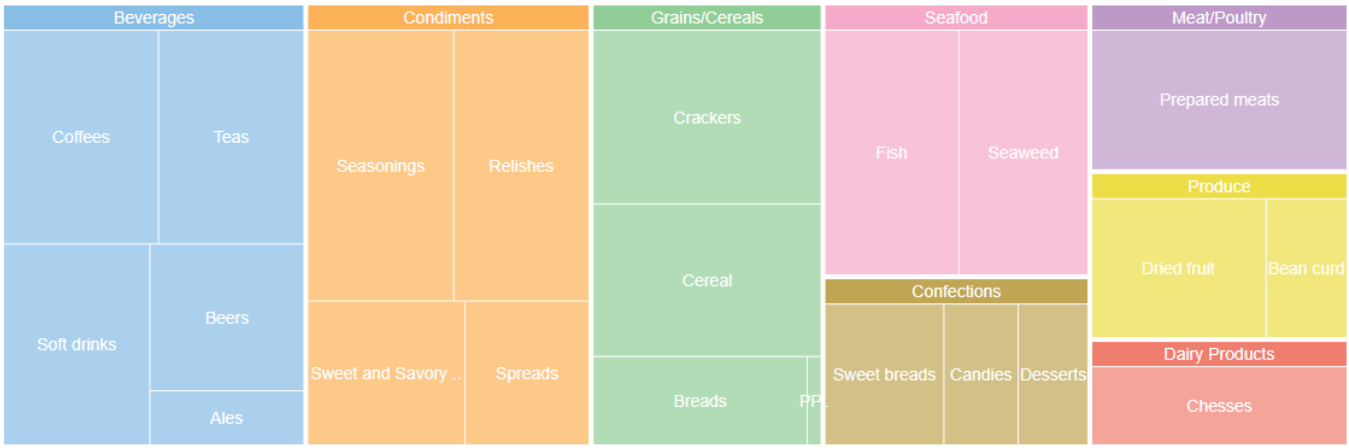
Index.cshtml

```
@model IEnumerable<FoodSale><c1-tree-map binding="Sales" binding-
name="Category,SubCategory">
<c1-items-source source-collection="Model" group-by="Category,SubCategory"></c1-
items-source>
<c1-flex-chart-datalabel position="Center" content="{name}"></c1-flex-chart-
datalabel>
</c1-tree-map>
```

Theming

The TreeMap chart control allows you to customize its appearance by using the Palette property. You can set a collection value to this property. The collection item type could be **System.Drawing.Color** or C1.Web.Mvc.TreeMapItemStyle.

The following image shows how a TreeMap control appears after applying a theme using **Palette** property.



The below example code uses FoodSale.cs model added in the QuickStart section.

In Code

Themeing.cshtml

HTML Helpers

Index.cshtml

```
@using <ApplicationName>.Models;
@using System.Drawing;
@model IEnumerable<FoodSale>

@{
    var colors = new List<string> { "#88bde6", "#fbb258", "#90cd97", "#f6aac9",
"#bfa554", "#bc99c7", "#eddd46", "#f07e6e", "#8c8c8c" };
}

@(Html.C1().TreeMap()
    .Binding("Sales")
    .BindingName("Category", "SubCategory")
    .Bind(Model)
    .Palette(colors.Select(ColorTranslator.FromHtml).ToList())
    .DataLabel(dlb => dlb.Position(C1.Web.Mvc.Chart.LabelPosition.Center).Content("
{name}")))
```

Tag Helpers

Index.cshtml

```
@model IEnumerable<FoodSale>@{
    var colors = new List<string> { "#88bde6", "#fbb258", "#90cd97", "#f6aac9",
"#bfa554", "#bc99c7", "#eddd46", "#f07e6e", "#8c8c8c" };
}

<c1-tree-map binding="Sales" binding-name="Category,SubCategory"
             palette="@colors">
    <c1-items-source source-collection="Model"></c1-items-source>
    <c1-flex-chart-datalabel position="Center" content="{name}"></c1-flex-chart-
datalabel>
</c1-tree-map>
```

TreeMap ASP.NET Core Tags

TreeMap control supports the following ASP.NET Core Tags:

<c1-tree-map>

- binding
- type

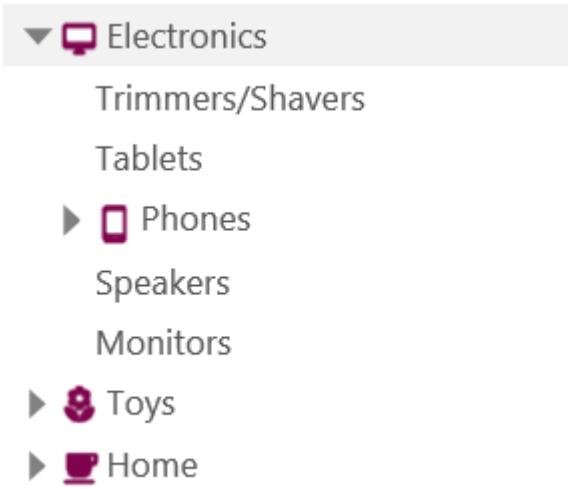
- binding-name
- child-items-path
- max-depth
- palette
- c1-flex-chart-datalabel

c1-tree-map-item-style

- title-color
- max-color
- min-color

TreeView

The TreeView control displays a hierarchical list which may contain text, check boxes, images, or arbitrary HTML content. A treeview is typically used to display the headings in a document, the entries in an index, the files and directories on a disk, or any other kind of information that might usefully be displayed as a hierarchy. The TreeView control manages data through nodes that can be selected, edited, and used to display simple text, images, and other elements such as check boxes.



Following are the properties required to create a tree:

- Bind/Source - An array that contains the hierarchical data. Each item in the array contains information about a node and (optionally) an array of child nodes.
- DisplayMemeberPath - Defines the name of the property in the items that contains the text to be displayed in the tree nodes. By default, this property is set to the string 'header'.
- ChildItemsPath - Defines the name of the property in the items that contains the array of child nodes. By default, this property is set to the string 'items'.

Apart from these properties, TreeView also offers properties for binding node images, check boxes, and collapsed state to the itemSource array. For more information, see Features.

Key Features

TreeView provides many different features that enable the developers to build intuitive and professional-looking applications. The main features for TreeView are as follows:

- **Node Expansion**

TreeView allows you to expand either a single node or all nodes in the tree as per the requirement. The TreeView control class provides the ExpandOnClick() method to expand a single node (parent or child), when you click on any node. And, AutoCollapse() method, which automatically closes the sibling nodes of the current selected parent node. For more information about node expansion, see Expanding and Collapsing Nodes.

- **Node Selection and Navigation**
  TreeView supports both keyboard and mouse navigation of nodes. In addition, the control supports single and multiple node selection. You can select multiple nodes in a contiguous or a non-contiguous manner. For more information about navigation, see Navigation Nodes.

- **Lazy Loading**
  TreeView allows you to delay the loading of a nodes child items until they are actually required. This is especially useful if you have a very deep tree, with lots of levels and child nodes and a simple example of this, is the folder structure of your Windows computer.

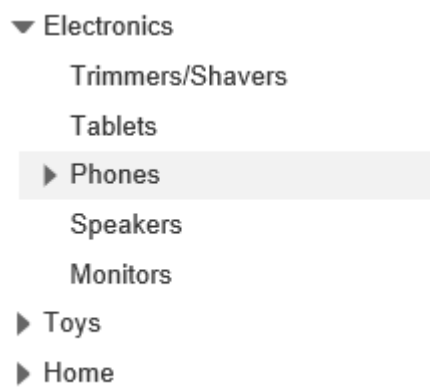
- **Custom Nodes**
  TreeView allows the user to set custom nodes, these nodes can be a simple text, image icons, multiple icons, and check boxes. You can customize the content of the TreeView nodes using the formatItem event. The event handler parameters include the element, which represents the node and the data item being rendered. For more information about custom node content, see Custom Node.

Quick Start: Add Data to TreeView

The topic describes how to add TreeView control to your MVC web application and add data to it using model binding. This topic comprises following four steps:

- **Step 1: Create an MVC Application**
- **Step 2: Create a Datasource for TreeView**
- **Step 3: Add a TreeView Control**
- **Step 4: Build and Run the Project**

The following image shows TreeView control after completing above the steps.



Step 1: Create an MVC Application

Create a new MVC application using the ComponentOne or VisualStudio templates. For more information about creating an MVC application, see Configuring your MVC Application topic.

Step 2: Create a Datasource for TreeView

1. Add a new class to the **Models** folder (Name: `Property.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `Property.cs` model. We are using **`Property`** class to represent a list of hierarchical data.

C#

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

public class Property
    {
        public string Header { get; set; }
        public string Image { get; set; }
        public bool NewItem { get; set; }
        public Property[] Items { get; set; }
        public static Property[] GetData()
        {
            return new Property[]
            {
                new Property
                {
                    Header = "Electronics",
                    Items = new Property[]
                    {
                        new Property { Header="Trimmers/Shavers" },
                        new Property { Header="Tablets" },
                        new Property { Header="Phones",
                            Items = new Property[] {
                                new Property { Header="Apple" },
                                new Property { Header="Motorola", NewItem=true
},
                                new Property { Header="Nokia" },
                                new Property { Header="Samsung" }}
                        },
                        new Property { Header="Speakers", NewItem=true },
                        new Property { Header="Monitors" }
                    }
                },
                new Property{
                    Header = "Toys",
                    Items = new Property[]{
                        new Property{ Header = "Shopkins" },
                        new Property{ Header = "Train Sets" },
                        new Property{ Header = "Science Kit", NewItem = true },
                        new Property{ Header = "Play-Doh" },
                        new Property{ Header = "Crayola" }
                    }
                },
                new Property{
                    Header = "Home",
```

```
                    Items = new Property[] {
                        new Property{ Header = "Coffeee Maker" },
                        new Property{ Header = "Breadmaker", NewItem = true },
                        new Property{ Header = "Solar Panel", NewItem = true },
                        new Property{ Header = "Work Table" },
                        new Property{ Header = "Propane Grill" }
                    }
                }
            };
        }
    }
```

Back to Top

Step 3: Add a TreeView Control

Steps to add a TreeView control to the application, are as follows:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (For example: `TreeViewController`).
    3. Click **Add**.
4. Include the following references as shown below.

C#

```
using <ApplicationName>.Models;
```

5. Replace the **Index()** method with the following method.

TreeViewController.cs

```
public ActionResult Index()
{
   return View(Property.GetData(Url));
}
```

Add a View for the Controller

In the view, we create an instance of TreeView and bind it to a data source using **Bind** method.
The DisplayMemberPath contains the text to be displayed in the tree nodes and ChildItemsPath contains the array of child nodes.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `TreeViewController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller, and then copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml copyCode

```
#region TreeView
@using TreeView.Models
@model Property[]

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
#endregion
```

Tag Helpers

Index.cshtml copyCode

```
@using TreeViewCore.Models
@model Property[]

<c1-tree-view display-member-path="Header" child-items-path="Items"
source="Model"> </c1-tree-view>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**TreeView/Index**) in the address bar of the browser to see the view.

Back to Top

Features

Learn about concepts that help you to understand how best to use TreeView control.

This section contains information about

Data Binding
: Learn how to do data binding in the TreeView control.

Lazy Loading
: Learn how to implement lazy loading in the TreeView control.

Navigating Nodes
: Learn about navigation in the TreeView control.

Expanding and Collapsing Nodes
: Learn about the concept of expanding and collapsing nodes in TreeView.

Using Check boxes
: Learn how to add check box on TreeView nodes.

Custom Node
: Learn how to add custom node content in TreeView nodes.

Images
: Learn how to add Images in TreeView nodes.

Accordion

Learn how to represent TreeView as an Accordion.

Drag and Drop
: Learn about the concept of drag and drop in TreeView

Editing Nodes
: Learn how to use TreeView nodes in edit mode.

Styling and CSS
: Learn how to apply styling for TreeView.

Data Binding

Before you can use TreeView and it's features, you must bind data to your TreeView control. Once you have bind the data to your TreeView, you can use many different features such as lazy loading, custom nodes, editing nodes, styling and much more.

This section contains information about

Model Binding
: Learn how to add data to your TreeView using basic Model binding.

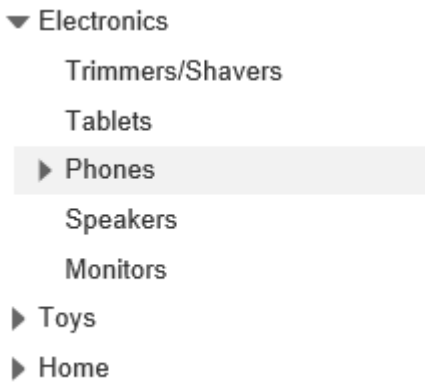
Remote Binding
: Learn how to remotely bind data to your TreeView using C1JSONRequest.

Model Binding

The topic describes how to add TreeView control to your MVC web application and add data to it using model binding. This topic comprises following three steps:

- **Step 1: Create a Datasource for TreeView**
- **Step 2: Add a TreeView Control**
- **Step 3: Build and Run the Project**

The following image shows TreeView control after completing the above steps.



Step 1: Create a Datasource for TreeView

1. Add a new class to the **Models** folder (Name: `Property.cs`). For more information on how to add a new model, see Adding Controls.
2. Add the following code to `Properties.cs` model. We are using **`Property`** class to represent a list of hierarchical data.

```
C#
```

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

public class Property
    {
        public string Header { get; set; }
        public string Image { get; set; }
        public bool NewItem { get; set; }
        public Property[] Items { get; set; }
        public static Property[] GetData()
        {
            return new Property[]
            {
                new Property
                {
                    Header = "Electronics",
                    Items = new Property[]
                    {
                        new Property { Header="Trimmers/Shavers" },
                        new Property { Header="Tablets" },
                        new Property { Header="Phones",
                            Items = new Property[] {
                                new Property { Header="Apple" },
                                new Property { Header="Motorola", NewItem=true
},
                                new Property { Header="Nokia" },
                                new Property { Header="Samsung" }}
                        },
                        new Property { Header="Speakers", NewItem=true },
                        new Property { Header="Monitors" }
                    }
                },
                new Property{
                    Header = "Toys",
                    Items = new Property[]{
                        new Property{ Header = "Shopkins" },
                        new Property{ Header = "Train Sets" },
                        new Property{ Header = "Science Kit", NewItem = true },
                        new Property{ Header = "Play-Doh" },
                        new Property{ Header = "Crayola" }
                    }
                },
                new Property{
                    Header = "Home",
                    Items = new Property[] {
                        new Property{ Header = "Coffeee Maker" },
                        new Property{ Header = "Breadmaker", NewItem = true },
                        new Property{ Header = "Solar Panel", NewItem = true },
                        new Property{ Header = "Work Table" },
```

```
                        new Property{ Header = "Propane Grill" }
                    }
                }
            };
        }
    }
```

Back to Top

Step 2: Add a TreeView Control

Steps to add a TreeView control to the application, are as follows:

Add a new Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | Controller**. The **Add Scaffold** dialog appears.
3. In the **Add Scaffold** dialog, follow these steps:
    1. Select the **Empty MVC Controller** template.
    2. Set name of the controller (For example: `TreeViewController`).
    3. Click **Add**.
4. Include the following references as shown below.

C#

```
using <ApplicationName>.Models;
```

5. Replace the **Index()** method with the following method.

TreeViewController.cs

```
public ActionResult Index()
{
   return View(Property.GetData(Url));
}
```

Add a View for the Controller

In the view, we create an instance of TreeView control and bind it to a data source using **Bind** method. The DisplayMemberPath sets the text to be displayed in the tree nodes and ChildItemsPath sets the array of child nodes.

1. From the **Solution Explorer**, expand the folder **Controllers** and double click the `TreeViewController.`
2. Place the cursor inside the method `Index().`
3. Right click and select **Add View**. The **Add View** dialog appears.
4. In the **Add View** dialog, verify that the View name is **Index** and View engine is **Razor (CSHTML).**
5. Click **Add** to add a view for the controller, and then copy the following code and paste it inside **Index.cshtml**.

HTML Helpers

Index.cshtml copyCode

```
@using TreeView.Models
@model Property[]

@(Html.C1().TreeView()
```

```
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
```

Tag Helpers

Index.cshtml copyCode

```
@using TreeViewCore.Models
@model Property[]

<c1-tree-view display-member-path="Header" child-items-path="Items"
source="Model"> </c1-tree-view>
```

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 folder name and view name to the generated URL (for example: http://localhost:1234/**TreeView/Index**) in the address bar of the browser to see the view.

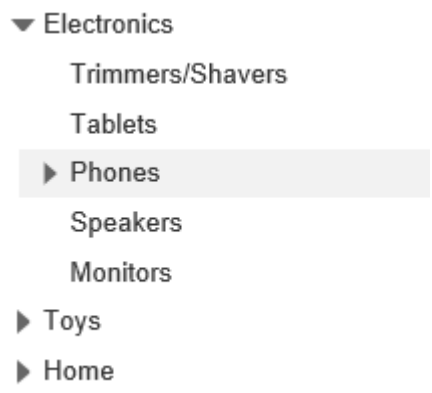
Back to Top

Remote Binding

The TreeView control allows you to retrieve data directly using **JSONResult** object. This specifies remote data URLs, which include the server, table and columns. The arrays returned are used as data sources for CollectionView objects. In tag helpers, LoadActionURL property in TreeView is used to bind it to a collection by passing an action URL method to carry out a specific operation. Similarly, in HTML helpers, **Bind** property is used to bind to a collection.

This topic demonstrates how to retrieve data from an existing data source remotely. This is useful for developing data-intensive applications and scenarios for representing data as dashboards.

The below example code uses **Property** model added in the QuickStart section. The following image shows how the TreeView control is displayed after making the C1JSON Request to fetch data from the model.



In Code

RemoteBindController.cs

C#

```
public class TreeViewController : Controller
    {
        // GET: TreeView
        public ActionResult Index()
        {
            return View();
        }
        public ActionResult RemoteLoading_LoadAction()
        {
            return Json(Property.GetData(url));
        }
    }
```

RemoteBind.cshtml

HTML Helpers

Razor

```
@(Html.C1().TreeView()
    .Bind(Url.Action("RemoteLoading_LoadAction"))
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
```

Tag Helpers

HTML

```
<c1-tree-view display-member-path="Header" child-items-path="Items"
load-action-url="@Url.Action("RemoteLoading_LoadAction")"></c1-tree-view>
```

Lazy Loading

The common process when using the TreeView is to bind to a collection of items. However, in some cases, you would want to delay the loading of a nodes child items until they are actually needed. This method is especially useful if you have a very deep tree, with lots of levels and child nodes and a simple example of this, is the folder structure of your Windows computer.

The TreeView control makes lazy loading easy to implement. To display TreeView in lazy loading mode, define an action to load the items when required and set the action URL to the **lazy-load-action-url tag** in case of tag helpers. In HTML helpers, you need to use the **Bind** property which uses a second parameter for action that returns lazy loading data.

Apart from this, you can also set lazy loading at client-side using JavaScript code. Set the LazyLoadFunction property to a function name, which invokes when a user expands the node. This function takes two parameters; the parent node and a callback to be invoked when the data becomes available.

The example code below helps you to understand the implementation of lazy loading feature in the TreeView control.

Note: In this sample, we are using **C1Nwind.mdf** database file and **EmployeeEx.cs** as a data source for the TreeView control. By default, the **C1Nwind.mdf** file is available on your system in *C:\Users*

> *<username>\Documents\ComponentOne Samples\ASP.NET MVC\MVC\CS\MvcExplorer\App_Data* folder. For more information on C1Nwind database, see Configure Data.

In Code

Model - EmployeeEx.cs

C#

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

namespace LazyLoading.Models
{
    public class EmployeeEx
    {
        private static DbSet employees = new C1NWindEntities().Employees;
        public int EmployeeID { get; set; }
        public string Name { get; set; }
        public EmployeeEx[] SubExployees { get; set; }

        public static IEnumerable GetEmployees(int? leaderID)
        {
            return employees.Where(e => e.ReportsTo == leaderID).ToList().Select(e =>
new EmployeeEx
            {
                EmployeeID = e.EmployeeID,
                Name = e.FirstName + " " + e.LastName,
                SubExployees = employees.Any(ep => ep.ReportsTo == e.EmployeeID) ?
new EmployeeEx[0] : null
            });
        }
    }
}
```

Controller - LazyLoadingController.cs

HTML Helpers

Razor

```
using C1.Web.Mvc;
using System.Web.Mvc;
using <ApplicationName.Models>;

namespace LazyLoading.Controllers
{
    partial class TreeViewController : Controller
    {

        // GET: LazyLoad
        public ActionResult LazyLoading()
        {
            return View();
        }

        public ActionResult Load()
        {
            return Json(EmployeeEx.GetEmployees(null));
        }

        public ActionResult LazyLoading_LoadAction([C1JsonRequest]TreeNode node)
        {
            var leaderID = (int?)node.DataItem["EmployeeID"];
            return Json(EmployeeEx.GetEmployees(leaderID));
        }
    }
}
```

Tag Helpers

HTML

```
using <ApplicationName.Models>;
using Microsoft.AspNetCore.Mvc;

namespace MvcExplorer.Controllers
{
    partial class TreeViewController : Controller
    {
        // GET: RemoteLoading
        public ActionResult RemoteLoading()
        {
            return View();
```

```
        }

        public ActionResult RemoteLoading_LoadAction()
        {
            return Json(Property.GetData(Url), new
Newtonsoft.Json.JsonSerializerSettings
            {
                ContractResolver = new
Newtonsoft.Json.Serialization.DefaultContractResolver()
            });
        }
    }
}
```

View - LazyLoading.cshtml

HTML Helpers

Razor

```
@(Html.C1().TreeView()
    .Bind(Url.Action("Load"), Url.Action("LazyLoading_LoadAction"))
    .DisplayMemberPath("Name")
    .ChildItemsPath("SubExployees"))
```

Tag Helpers

HTML

```
<c1-tree-view display-member-path="Name"
    child-items-path="SubExployees"
    load-action-url="@Url.Action("Load")"
    lazy-load-action-url="@Url.Action("LazyLoading_LoadAction")">
</c1-tree-view>
```

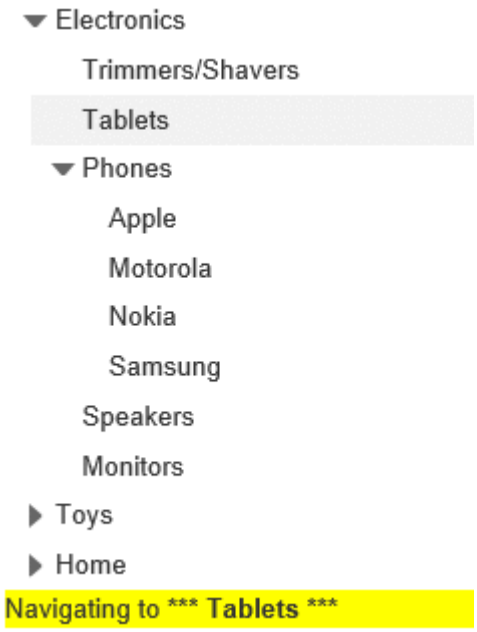
Back to Top

Node Navigation

The simplest and most common use for the TreeView control is navigation. The TreeView's hierarchical structure and auto-search functionality makes it easy for users to drill-down and find the items they are interested in. TreeView supports both mouse and keyboard navigation of nodes.

You can use the OnClientSelectedItemChanged or OnClientItemClicked events for navigation in the TreeView control. The difference is that **OnClientSelectedItemChanged** occurs when user moves the selection with keyboard, and **OnClientItemClicked** occurs when user clicks an item or presses the Enter key.

This example uses the OnClientItemClicked event for navigation. The below example code uses **Property** model added in the QuickStart section.



In Code

NavigationController.cs

C#

```
public class TreeViewController : Controller
    {
        // GET: TreeView
        public ActionResult Index()
        {
            return View(Property.GetData(Url));
        }
    }
```

Navigation.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
function itemClicked(treeView)

{
document.getElementById('tvNavItem').innerHTML = 'Navigating to <b> *** ' +
treeView.selectedItem.Header + ' ***</b>';
}
</script>
```

```
@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .OnClientItemClicked("itemClicked"))

<div id="tvNavItem" style="background-color:yellow"></div>
```

Tag Helpers

HTML

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
function itemClicked(treeView)

{
document.getElementById('tvNavItem').innerHTML = 'Navigating to <b> *** ' +
treeView.selectedItem.Header + ' ***</b>';

}
</script>
<c1-tree-view display-member-path="Header" child-items-path="Items"
source="Model" item-clicked="itemClicked"></c1-tree-view>

<br/>
<div id="tvNavItem" style="background-color:yellow"></div>
```

Mouse Navigation

The following table lists the actions and the corresponding Mouse commands for navigating nodes in TreeView:

Action	Mouse Command
Expand a node	Click on the plus sign on the left side of the node's name.
Collapse a node	Click on the minus sign on the left side of the node's name.
Select a node	Click on the node's name.

Keyboard Navigation

The following table describes the actions and their associated keys to use when navigating nodes in TreeView:

Action	Keyboard Command
Move up a node	Up Arrow Key
Move down a node	Down Arrow Key
Distributed selection	MOUSE + CTRL
Continuous selection	MOUSE + SHIFT

Expanding and Collapsing Nodes

The TreeView control displays a hierarchical list of items. Each tree view node can have a number of child nodes, which are indented below the parent node. Child nodes are displayed when their parent node is expanded and hidden when the node is collapsed.

The TreeView class provides the ExpandOnClick property to expand a single node (parent or child), when you click on any node. The ExpandOnClick method accepts a Boolean value to determine whether the child nodes within a particular node should be expanded or not. Once you set this property to true, it allows you to expand all the child nodes with a mouse click.

The TreeView control provides AutoCollapse property, which automatically closes the sibling nodes of the currently selected parent node. This happens when you click on any other parent node within the same TreeView.

This example uses the **ExpandOnClick** and the **AutoCollapse** method to perform expand and collapse operation on nodes. The below example code uses **Property** model added in the QuickStart section.

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .AutoCollapse(true)
    .ExpandOnClick(true))
```

Tag Helpers

HTML

```
@using <ApplicationName.Models>
@model Property[]

<c1-tree-view display-member-path="Header" child-items-path="Items"
    source="Model"
    auto-collapse="true"
    expand-on-click="true"></c1-tree-view>
```

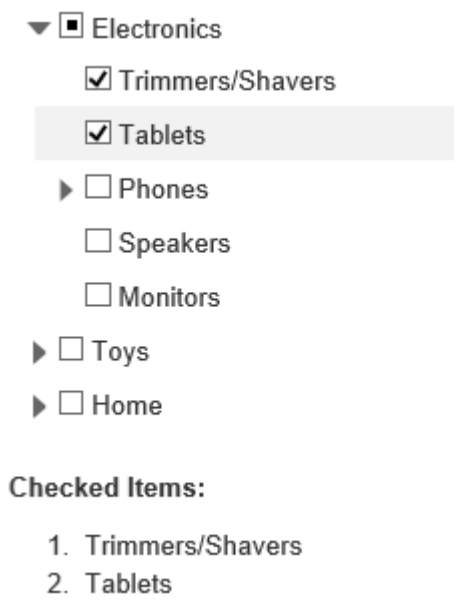
Using Check Box

Treeview provides an option to display the nodes as check boxes. In order to display check boxes in the TreeView control, you need to set the ShowCheckboxes property to true.

When check boxes are displayed, the TreeView control manages their hierarchy so that when a check box is checked or cleared, new value is automatically applied to all child nodes, and reflected on the state of the parent nodes. When items are checked or unchecked, the OnClientCheckedItemsChanged event is raised, and the **checkedItems** property of the client TreeView object is updated with a list of the items that are currently checked.

This example uses the OnClientCheckedItemsChanged event to show the currently selected check box in the TreeView

control. The below example code uses **Property** model added in the QuickStart section.



In Code

Checkboxes.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
        function checkedItemsChanged(treeView) {
            var items = treeView.checkedItems,
                msg = '';
            if (items.length) {
                msg = '<p><b>Checked Items:</b></p><ol>\r\n';
                for (var i = 0; i < items.length; i++) {
                    msg += '<li>' + items[i].Header + '</li>\r\n';
                }
                msg += '</ol>';
            }
            document.getElementById('tvChkStatus').innerHTML = msg;
        }
    </script>

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .ShowCheckboxes(true)
```

```
    .OnClientCheckedItemsChanged("checkedItemsChanged"))
<br />
<div id="tvChkStatus"></div>
```

Tag Helpers

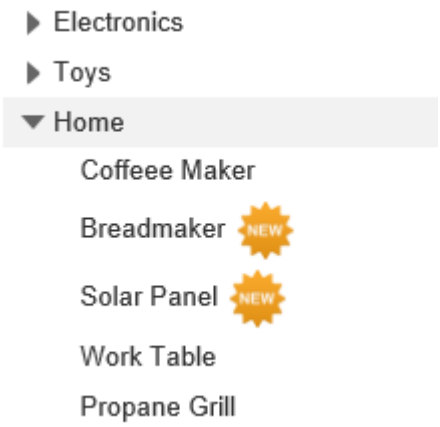
HTML

```
@using <ApplicationName.Models>
@model Property[]
<script type="text/javascript">
        function checkedItemsChanged(treeView) {
            var items = treeView.checkedItems,
                msg = '';
            if (items.length) {
                msg = '<p><b>Checked Items:</b></p><ol>\r\n';
                for (var i = 0; i < items.length; i++) {
                    msg += '<li>' + items[i].Header + '</li>\r\n';
                }
                msg += '</ol>';
            }
            document.getElementById('tvChkStatus').innerHTML = msg;
        }
</script>
<c1-tree-view display-member-path="Header" child-items-path="Items"
              show-checkboxes="true" source="Model"
              checked-items-changed="checkedItemsChanged"></c1-tree-view>
<br />
<div id="tvChkStatus"></div>
```

Custom Node

The TreeView control allows you to display custom content in nodes. You can customize the content of the TreeView nodes using the **formatItem** event. The event handler's parameters include the element that represents the node and the data item being rendered.

The example uses the OnClientFormatItem event to add a "new" badge to the right of new items on the treeview. The below example code uses **Property** model added in the QuickStart section.



In the code example below, we subscribe the **OnClientFormatItem** method by assigning the JavaScript function name in the razor code. In the **<Script>** section, we declare the formatItem function to check, if the dataItem is a new item in the list, then it will add a custom image beside the new nodes. The image is stored inside the **Content** folder of your MVC application.

In the code example below, **OnClientFormatItem()** method raises the formatItem event. In the **formatItem(treeview, args)** function, **treeview** is the sender, and **args** is a type of FormatNodeEventArgs class.

In Code

CustomNode.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
     function formatItem(treeview, args) {
       if (args.dataItem.NewItem) {
        args.element.innerHTML +=
            '<img style="margin-left:6px" src="@Href("~/Content/new.png")"/>';
            }
        }
</script>

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .OnClientFormatItem("formatItem"))
```

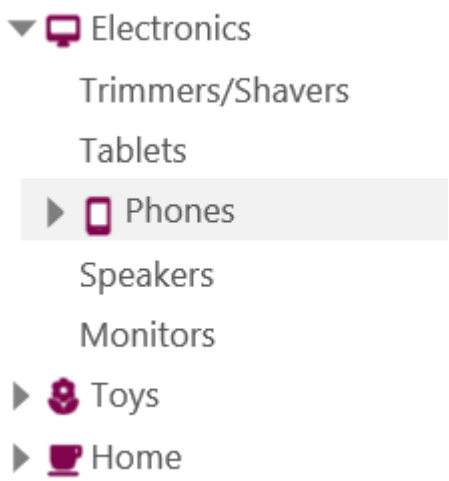
Tag Helpers

HTML

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
    function formatItem(treeview, args) {
        if (args.dataItem.NewItem) {
            args.element.innerHTML +=
                '<img style="margin-left:6px" src="@Href("~/Content/new.png")"/>';
            }
        }
</script>

<c1-tree-view display-member-path="Header" child-items-path="Items"
    source="Model" format-item="formatItem"></c1-tree-view>
```

Images

The TreeView control is capable of showcasing different appearances, providing flexible image customization as well as properties that specify custom user interface (UI) options. TreeView allows you to use images to represent nodes, connecting lines, and the expand and collapse icons. You can use the ImageMemberPath property to add images to nodes by specifying the name of a property on the data items that contains an image URL.



In this sample, we are using electronics.png, phones.png, toys.png, and home.png. These images are stored in **/Content/images/** folder in the application. The data source contains an **Image** field, which contains the image URL as value.

The following example code demonstrate how to add images in the TreeView nodes.

In Code

Model - Property.cs

C#

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

```
public class Property
    {
        public string Header { get; set; }
        public string Image { get; set; }
        public bool NewItem { get; set; }
        public Property[] Items { get; set; }
        public static Property[] GetData(UrlHelper urlHelper)
        {
            return new Property[]
            {
                new Property
                {
                    Header = "Electronics",
                    Image = urlHelper.Content("~/Content/images/electronics.png"),
                    Items = new Property[]
                    {
                        new Property { Header="Trimmers/Shavers" },
                        new Property { Header="Tablets" },
                        new Property { Header="Phones",
                            Image =urlHelper.Content("~/Content/images/phones.png"),
                            Items = new Property[] {
                                new Property { Header="Apple" },
                                new Property { Header="Motorola", NewItem=true },
                                new Property { Header="Nokia" },
                                new Property { Header="Samsung" }}
                        },
                        new Property { Header="Speakers", NewItem=true },
                        new Property { Header="Monitors" }
                    }
                },
                new Property{
                    Header = "Toys",
                    Image = urlHelper.Content("~/Content/images/toys.png"),
                    Items = new Property[]{
                        new Property{ Header = "Shopkins" },
                        new Property{ Header = "Train Sets" },
                        new Property{ Header = "Science Kit", NewItem = true },
                        new Property{ Header = "Play-Doh" },
                        new Property{ Header = "Crayola" }
                    }
                },
                new Property{
                    Header = "Home",
                    Image = urlHelper.Content("~/Content/images/home.png"),
                    Items = new Property[] {
                        new Property{ Header = "Coffeee Maker" },
                        new Property{ Header = "Breadmaker", NewItem = true },
                        new Property{ Header = "Solar Panel", NewItem = true },
                        new Property{ Header = "Work Table" },
                        new Property{ Header = "Propane Grill" }
```

```
                    }
                }
            };
        }
    }
```

TreeViewController

TreeViewController.cs

```
using <ApplicationName>.Models;

public ActionResult Index()
{
   return View(Property.GetData(url));
}
```

View - Index.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .ImageMemberPath("Image"))
```

Tag Helpers

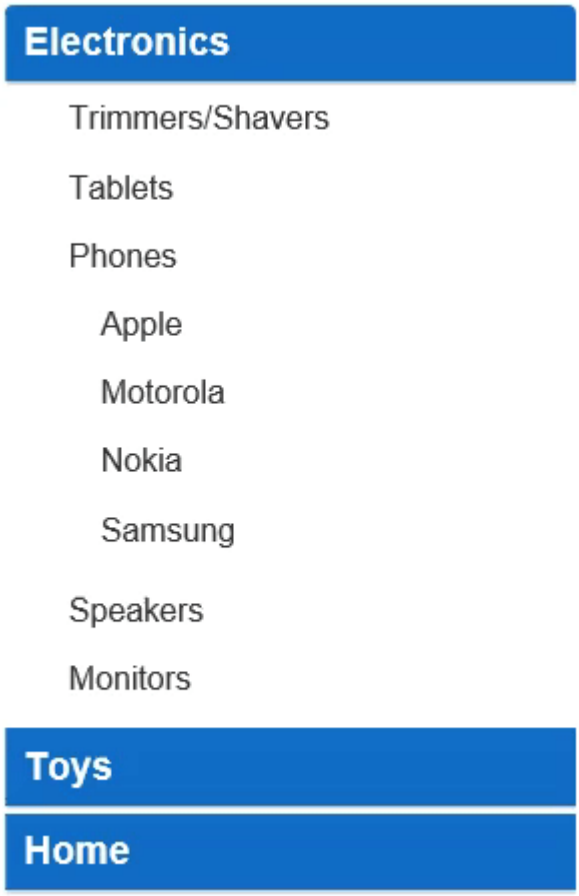
HTML

```
@using <ApplicationName.Models>
@model Property[]

<c1-tree-view display-member-path="Header" child-items-path="Items"
image-member-path="Image" source="Model"></c1-tree-view>
```

Accordion

An accordion menu is a vertically stacked list of headers that can be clicked to reveal or hide content associated with them. They are commonly used for navigation. The main advantage of using accordion is that it reduces page scrolling and allows the user to hide content that makes the web appear less complicated.

In this sample, we will use **CSS** to customize the header display and to hide the collapse/expand glyphs. You need to make sure that AutoCollapse property is set to true (the default), so non-active panels are automatically collapsed.



The example uses CssClass property to display the TreeView control as an accordion. The below example code uses **Property** model added in the QuickStart section.

Add Custom Stylesheet

Create a new ASP.NET MVC application. Once you have created the application, a **Content** folder is created in the **Solution Explorer** after adding the view to the application. To add a custom style sheet in your application, follow these steps:

1. In the Solution Explorer, right-click the **Content** folder.
2. From the context menu, select **Add | Style Sheet**. The **Specify Name for Item** dialog appears.
3. Set name of the style sheet (for example: `app.css`) and click **OK**.
4. Replace the default code of **app.css** file with the code given below.

app.css

```
/* accordion tree styles */

/* hide collapse/expand glyphs */
.accordion-tree.wj-treeview .wj-nodelist .wj-node:before {
    display: none;
}

/* level 0 nodes (headers) */
.accordion-tree.wj-treeview .wj-nodelist > .wj-node {
    font-size: 120%;
    font-weight: bold;
```

```
    padding: 6px 10px;
    color: white;
    background: #106cc8;
    margin-bottom: 4px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

/* level 1 nodes (navigation items) */
.accordion-tree.wj-treeview .wj-nodelist > .wj-nodelist > .wj-node {
    font-size: inherit;
    font-weight: normal;
    padding: 4px 1em;
    color: inherit;
    background: inherit;
    box-shadow: none;
}
    .accordion-tree.wj-treeview .wj-nodelist {
        padding-bottom: 6px;
    }
```

Back to Top

In Code

Accordion.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

@(Html.C1().TreeView().CssClass("accordion-tree")
    .Bind(Model).Id("accordion")
    .IsContentHtml(true)
    .Width(300)
    .AutoCollapse(true)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
```

Tag Helpers

HTML

```
@using <ApplicationName.Models>
@model Property[]

<c1-tree-view id="accordion" class="accordion-tree" is-content-html="true"
    auto-collapse="true" display-member-path="Header"
    child-items-path="Items" source="Model"></c1-tree-view>
```

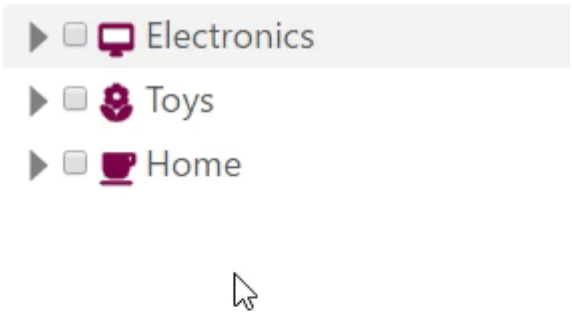
Back to Top

Drag and Drop

The TreeView control supports drag and drop operations within the same treeview and among multiple treeviews. To perform drag and drop operation in TreeView nodes, you need to set the AllowDragging property of TreeView to true.

When drag and drop operations are enabled in TreeView, users may drag any node to any position within the tree. You can customize this behavior by handling the TreeView drag and drop events:

- OnClientDragStart: Occurs when a drag or drop operation is about to start. You may examine the node about to be dragged and cancel the operation by setting the event's cancel parameter to true.
- OnClientDragOver: Occurs while the user drags the node over other nodes on the tree. You may examine the current target node and drop position and prevent the drop or modify its location setting in the event's cancel and position parameters.
- OnClientDrop: Occurs when the user drops the node into its new location. You may examine the current target node and drop position and prevent the drop or modify its location setting the event's cancel and position parameters.
- OnClientDragEnd: Occurs after the drag/drop operation is finished, even if it was cancelled and the source node was not moved.



In this example, we will use OnClientDragStart and OnClientDragOver to provide customized drag and drop operations on a TreeView control. We are using **OnClientDragStart** event to allow users to drag parent nodes and **OnClientDragOver** event to allow users to drop into empty nodes.

In the code example below, we are using OnClientDragStart() method that raises the dragStart event. In the **dragStart(treeview, e)** function, **treeview** is the sender, and **e** is a type of TreeNodeEventArgs class. Similarly, OnClientDragOver() method raises the dragOver event. In the **dragOver(treeview, e)** function, **treeview** is the sender, and **e** is a type of TreeNodeEventArgs class.

The below example code uses **Property** model added in the QuickStart section.

In Code

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

<script type="text/javascript">
        var allowDraggingParentNodes = true;
        var allowDroppingIntoEmpty = true;

        // use OnClientDragStart event to honor the allowDraggingParentNodes setting
        // by setting the 'cancel' event parameter to true
        function dragStart(treeview, e) {
            if (e.node.hasChildren) {
                if (!allowDraggingParentNodes) {
                    e.cancel = true; // prevent dragging parent nodes
                } else {
                    e.node.isCollapsed = true; // collapse parent nodes when dragging
                }
            }
        }

        // use OnClientDragOver event to honor the allowDroppingIntoEmpty setting
        // by changing the 'position' event parameter to 'Before'
        function dragOver(treeview, e) {
            if (!allowDroppingIntoEmpty &&
                !e.dropTarget.hasChildren &&
                 e.position == wijmo.nav.DropPosition.Into) {
                e.position = wijmo.nav.DropPosition.Before;
            }
        }
</script>

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .ImageMemberPath("Image")
    .ShowCheckskboxes(true)
    .AllowDragging(true)
    .OnClientDragStart("dragStart")
    .OnClientDragOver("dragOver"))
```

Tag Helpers

HTML

```
<script type="text/javascript">
        var allowDraggingParentNodes = true;
```

```
        var allowDroppingIntoEmpty = true;

        // use OnClientDragStart event to honor the allowDraggingParentNodes setting
        // by setting the 'cancel' event parameter to true
        function dragStart(treeview, e) {
            if (e.node.hasChildren) {
                if (!allowDraggingParentNodes) {
                    e.cancel = true; // prevent dragging parent nodes
                } else {
                    e.node.isCollapsed = true; // collapse parent nodes when dragging
                }
            }
        }

        // use OnClientDragOver event to honor the allowDroppingIntoEmpty setting
        // by changing the 'position' event parameter to 'Before'
        function dragOver(treeview, e) {
            if (!allowDroppingIntoEmpty &&
                !e.dropTarget.hasChildren &&
                 e.position == wijmo.nav.DropPosition.Into) {
                e.position = wijmo.nav.DropPosition.Before;
            }
        }
</script>

<c1-tree-view display-member-path="Header" child-items-path="Items"
    image-member-path="Image" show-checkboxes="true"
    allow-dragging="true" source="Model"
    drag-start="dragStart" drag-over="dragOver"></c1-tree-view>
```

Editing Nodes

The TreeView control provides an option to enable or disable the editing of a node at run-time through IsReadOnly property. When the IsReadOnly property is set to false, user can edit the content of the nodes by double-clicking a node or by selecting a node, and then pressing the F2 key. Setting the IsReadOnly property to true restricts the users from editing the nodes of tree.

Once user finishes editing the node, node contents are automatically applied to the items in the Source array with the help of **DisplayMemberPath** property. The TreeView control allows you to customize the editing behaviour using the following events:

- OnClientNodeEditStarting - Occurs before a node enters the edit mode.
- OnClientNodeEditStarted - Occurs after a node has entered the edit mode.
- OnClientNodeEditEnding - Occurs after a node has exited the edit mode.
- OnClientNodeEditEnded - Occurs before a node exits the edit mode.

In the example code below, we have enabled editing only for nodes that contain no children using the OnClientNodeEditStarting method. In the code example below, **OnClientNodeEditStarting()** method raises the nodeEditStarting event. In the **nodeEditStarting(treeview, e)** function, **treeview** is the sender, and **e** is a type of TreeNodeEventArgs class.

The below example code uses **Property** model added in the QuickStart section.

In Code

EditingNodes.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]
<script type="text/javascript">
        function nodeEditStarting(treeview, e) {
            if (e.node.hasChildren) {
                e.cancel = true;
            }
        }
</script>

@(Html.C1().TreeView()
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items")
    .ImageMemberPath("Image")
    .ShowCheckboxes(true)
    .IsReadOnly(false)
    .OnClientNodeEditStarting("nodeEditStarting"))
```

Tag Helpers

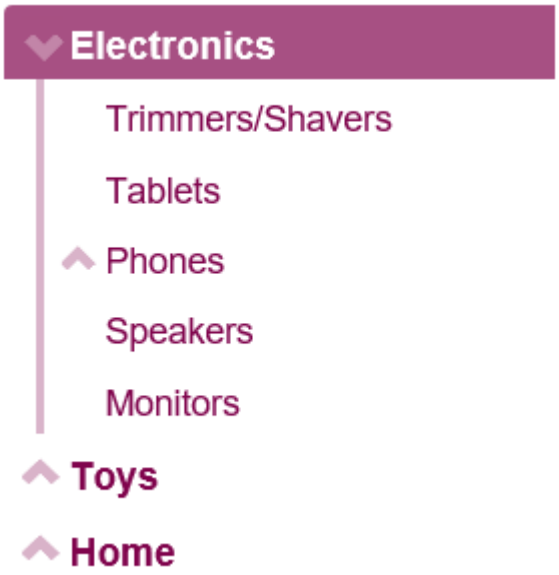
HTML

```
@using <ApplicationName.Models>
@model Property[]
<script type="text/javascript">
        function nodeEditStarting(treeview, e) {
            if (e.node.hasChildren) {
                e.cancel = true;
            }
        }
    </script>

<c1-tree-view display-member-path="Header" child-items-path="Items"
              image-member-path="Image" show-checkboxes="true"
              is-read-only="false" source="Model"
              node-edit-starting="nodeEditStarting"></c1-tree-view>
```

Styling and CSS

The TreeView control allows to customize its appearance using **CSS**. You can create your own custom CSS and then apply it to the TreeView control using CssClass property. The following image shows how the TreeView

control appears after using a custom CSS class.



The example uses **CssClass** property to apply custom cascading styling sheets in the TreeView control. The below example code uses **Property** model added in the QuickStart section.

Add Custom Stylesheet

Create a new ASP.NET MVC application. Once you have created the application, a **Content** folder is created in the **Solution Explorer** after adding the view to the application. To add a custom style sheet in your application, follow these steps:

1. In the Solution Explorer, right-click the **Content** folder.
2. From the context menu, select **Add | Style Sheet**. The **Specify Name for Item** dialog appears.
3. Set name of the style sheet (for example: `app.css`) and click **OK**.
4. Replace the default code of **app.css** file with the code given below.

app.css

```
.wj-treeview {
    font-size: 120%;
    margin-bottom: 8px;
}

/* custom tree styles */
.custom-tree.wj-treeview {
    color: #80044d;
}

    /* default nodes */
    .custom-tree.wj-treeview .wj-node {
    }

    /* level 0 and deeper nodes */
    .custom-tree.wj-treeview .wj-nodelist > .wj-node {
        font-size: 120%;
        font-weight: bold;
```

```
    }

    /* level 1 and deeper nodes (smaller font, vertical line along the left) */
    .custom-tree.wj-treeview .wj-nodelist > .wj-nodelist > .wj-node,
    .custom-tree.wj-treeview .wj-nodelist > .wj-nodelist > .wj-nodelist {
        font-size: 110%;
        font-weight: normal;
        border-left: 4px solid rgba(128, 4, 77, 0.3);
    }

    /* level 2 and deeper nodes (smaller font, thinner border) */
    .custom-tree.wj-treeview .wj-nodelist > .wj-nodelist  > .wj-nodelist > .wj-
node,
    .custom-tree.wj-treeview .wj-nodelist > .wj-nodelist  > .wj-nodelist > .wj-
nodelist {
        font-size: 100%;
        font-style: italic;
        opacity: 0.8;
        border-left: 2px solid rgba(128, 4, 77, 0.3);
    }

    /* expanded node glyph */
    .custom-tree.wj-treeview .wj-nodelist .wj-node:before {
        content: "\e114";
        font-family: 'Glyphicons Halflings';
        top: 4px;
        border: none;
        opacity: .3;
        transition: all .3s cubic-bezier(.4,0,.2,1);
    }

    /* collapsed node glyph */
    .custom-tree.wj-treeview .wj-nodelist .wj-node.wj-state-collapsed:before,
    .custom-tree.wj-treeview .wj-nodelist .wj-node.wj-state-collapsing:before {
        transform: rotate(-180deg);
        transition: all .3s cubic-bezier(.4,0,.2,1);
    }

    /* selected node */
    .custom-tree.wj-treeview .wj-node.wj-state-selected {
        color: white;
        background: rgba(128, 4, 77, 0.70);
    }
```

In Code

The example code changes the collapse/expand icons, uses different font sizes depending on node level, and adds a vertical bar to the left of the level one nodes.

Styling.cshtml

HTML Helpers

Razor

```
@using <ApplicationName.Models>
@model Property[]

@(Html.C1().TreeView().CssClass("custom-tree")
    .Bind(Model)
    .DisplayMemberPath("Header")
    .ChildItemsPath("Items"))
```

Tag Helpers

HTML

```
@using <ApplicationName.Models>
@model Property[]

<c1-tree-view class="custom-tree"
        display-member-path="Header" child-items-path="Items"
        source="Model"></c1-tree-view>
```

TreeView ASP.NET Core Tags

TreeView control supports the following ASP.NET Core Tags:

<c1-tree-view>

- source
- is-contentHtml
- show-checkboxes
- auto-collapse
- is-animated
- is-readOnly
- allow-dragging
- expand-on-click
- lazy-load-function
- items-source-changed
- loading-items
- loaded-items
- item-clicked
- selected-item-changed
- checked-items-changed
- is-collapsed-changing
- is-collapsed-changed
- is-checked-changing
- is-checked-changed
- format-item
- drag-start
- drag-over
- drop
- drag-end

- node-edit-starting
- node-edit-started
- node-edit-ending
- node-edit-ended
- lazy-load-action-url
- load-action-url
- child-items-path
- display-member-path
- image-member-path

ASP.NET MVC Samples

With the **C1Studio** installer, you get our MVC control samples that help you understand the implementation of the product. The C# and VB samples are available at the default installation folder - *Documents\ComponentOne Samples\ASP.NET MVC\MVC*.

The list of available C# samples is as follows:

Sample	Description
FinancialChartExplorer	This sample demonstrates working of financial charts and their features.
FlexSheetExplorer	This sample demonstrates working of FlexSheet control and its features. The FlexSheet control extends the FlexGrid control and provides an Excel-like functionality.
FlexViewerExplorer	This sample demonstrates various features of ReportViewer and PdfViewer.
HowTo	The samples included in HowTo folder are basic getting started guide for our popular controls. It includes the sample for the following applications. • **BookMyFlight** - A flight booking application that lets you Search for and book flights, filter results by price and duration, or view historical bookings, manage your profile, and print invoices. • **C1Finance** - The application retrieves financial data from Quandl and calculates the current value and change for each stock. It also creates charts comparing the evolution in stock value for each company. • **CollectionView** - CollectionView 101 sample shows how to get started with ASP.NET MVC's CollectionView controls. • **Financial Chart** - This sample shows how to get started with ASP.NET MVC's FinancialChart control. • **FlexChart** - This sample demonstrates how to get started with ASP.NET MVC's FlexChart controls. • **FlexGrid** - This sample demonstrates how to get started with ASP.NET MVC's FlexGrid controls. • **FlexSheet** - The sample shows one page with

	how-to's for the most important excel-like tasks. • **Gauge** - This sample demonstrates how to get started with ASP.NET MVC's Gauge controls. • **ImageCombo** - This sample demonstrates how to create ImageCombo class which inherits C1 MVC ComboBox. • **Input -** This sample demonstrates how to get started with MVC's Input controls. • **MultiRow LOB -** This sample demonstrates working of MultiRow control and its features. • **Olap -**The sample shows how to use the PivotEngine, PivotPanel and PivotGrid components. • **PdfViewer -** This sample demonstrates how to open a local PDF file in PdfViewer. • **ReportViewer** - Shows how to get started with ASP.NET MVC ReportViewer control. The sample shows one page with how-to's for some of the most important report viewer tasks. • **Periodic Table** - Shows how to use Sunburst control to make a periodic table.
MultiRowExplorer	This sample demonstrates working of MultiRow control and its features. The MultiRow control extends conventional grid layouts by using multiple rows to represent each data item.
MvcExplorer	This sample includes implementation for the following controls. • FlexGrid • FlexPie • FlexChart • FlexRadar • TreeView • Linear Gauge • TreeMap • Input Controls • Radial Gauge • Sunburst Chart
OlapExplorer	This sample demonstrates how to bind data to an Olap control. It includes the following data binding methods. • Remote Bind • DataEngine Service(DataEngine) • DataEngine Service(DataSource) • DataEngine Service(SSAS)
ASPNETCore	The ASPNETCore folder includes the following samples implemented using ASP.NET Core 2.0 framework.

	• FinancialChartExplorer • FlexSheetExplorer • FlexViewerExplorer • HowTo • MultiRowExplorer • RazorPagesExplorer • MvcExplorer • OlapExplorer

The list of available VB samples is as follows:

Sample	Description
HowTo	The samples included in HowTo folder are basic getting started guide for our popular controls. It includes the sample for the following applications. • **FlexChart** - This sample demonstrates how to get started with ASP.NET MVC's FlexChart controls. • **FlexGrid** - This sample demonstrates how to get started with ASP.NET MVC's FlexGrid controls. • **FlexSheet** - The sample shows one page with how-to's for the most important excel-like tasks. • **Input -** This sample demonstrates how to get started with MVC's Input controls. • **Olap -**The sample shows how to use the PivotEngine, PivotPanel and PivotGrid components. • **PdfViewer -** This sample demonstrates how to open a local PDF file in PdfViewer. • **ReportViewer** - Shows how to get started with ASP.NET MVC ReportViewer control. The sample shows one page with how-to's for some of the most important report viewer tasks.

Release History

The release history contains information on all the new controls, breaking changes, improvements, new features and bug fixes for the ASP.NET MVC Controls since the previous release. Choose a release version to learn more:

2017 v3

ASP.NET MVC

Enhancements

- Added TreeMap control.
- Updated MVC project templates for ASP.NET Core 2.0 Framework.
- Updated the Project Wizard by adding group templates (Standard, Model Binding, Ajax Binding, Spreadsheet).
- Improved accessibility by adding ARIA 1.1 built-in support.
- [FinancialChart] Added new Financial Chart type 'PointAndFigure'.
- Made popup ignore Escape key while IME mode is active.
- Added time zone offset date format parts ('z', 'zz', 'zzz') to Globalize.formatDate.
- Added a Popup.removeOnHide property to control whether the Popup should be removed from the DOM or just hidden when the popup closes.
- [ReportViewer] Hide the Parameters tab if all parameters are hidden.
- Added a MultiSelect.showSelectAllCheckbox property to display a "Select All" checkbox above the items, so users can select/de-select all items with a single click.
- Added a new FlexGrid.itemValidator property to improve validation support, especially for unbound grids (bound grids can be validated using the CollectionView.getError property which provides the same functionality).
- Added a MultiSelect.selectAllLabel property to customize the label shown next to the "Select All" checkbox displayed when the showSelectAllCheckbox property is set to true.
- Added some configuration properties to the wijmo.olap.PivotEngine class: serverTimeOut: the timeout value for retrieving results from the server, serverPollInterval: the poll interval for getting progress reports from the server, serverMaxDetail: the maximum number of detail records to retrieve from the server.
- Added several new properties to make the Calendar control more customizable: formatYearMonth, formatDayHeaders, formatDays, formatYear, and formatMonths. All these properties represent format strings used to format different parts of the Calendar in month and year view.

- Added two new properties to improve FlexGrid keyboard accessibility: keyActionTab and keyActionEnter. These properties allow you to customize the behavior of special keys so the grid becomes more accessible or more compatible with Excel.
- Added a new PivotField.sortComparer property to allow customization of the sort order in dimension fields. This is similar to the CollectionView's sortComparer property, except it applies to pivot dimensions (grid headers) as opposed to measures (summary data).

CollectionView

Enhancements

- Added forceRefresh method in RemoteCollectionView.

Bugs

- Fixed an issue where user encounters run-time error when enable property "group-by".
- Fixed an issue of unit test CollectionViewServiceTests.VirtualScrollingTest.

OLAP

Bugs

- Fixed an issue where JavaScript error is displayed while setting Fields of DataEngine to multiple fields.
- Removed "auto-generate-fields" attribute from PivotEngine tag helper.

Scaffolder

Bug Fixes

- Fixed an issue where 'FontSize' of Header style for all Charts can be set by Scaffold but it does not take effect in the application.

FlexGrid

Enhancements

- Added KeyAction.CycleOut Enum item.

Bugs

- Fixed an issue where value in filter dropdown could not display, after delete the record in the grid during filtering.
- Fixed an issue where user could not cancel the Editing operation by using the Esc key in Unobtrusive Validation sample.
- Fixed an issue where focus does not shifts to the next cell when pressing Tab key on a cell having custom editor.
- Fixed an issue where console error occurs after clicking any cell in custom editors page.

InpuTime

Bug Fixes

- Fixed an issue where data in the dropdown could not show correctly when binding with customize items.
- Fixed an issue where data in the dropdown could not show correctly when min/max value is set in InputTime control.

InpuDate

Bug Fixes

- Fixed an issue where JavaScript error occurs when value binding is used in InputDate/InputDateTime control of ASP.NET Core project.

Project Templates

Bug Fixes

- Fixed an issue where data could not load and 404 Not Found error occurred when Ajax template is host in IIS.
- Fixed an issue where error occurs after user publish the Spreadsheet Core 2.0 template project.
- Fixed an issue where C# template icons were displayed when choosing the VB template for ComponentOne ASP.NET MVC Application Wizard.

Back to Top

2017 v2

Core

Enhancements

- Added MultiAutoComplete control.
- Added TreeView control.
- Added Finance sample.
- Added FilterPanel sample.
- Updated C1 MVC templates with new structure.
- Added business application template. (Spreadsheet)
- Responsive application template with server binding. (FlexGrid and Menu)
- Responsive application template with AJAX binding. (FlexGrid and Menu)
- Refactoring Scaffolder for inserting MVC controls.
- Added gradient color support in FlexChart.

Bugs

- Fixed an issue where C1CollectionItemConverterAttribute is used for a collection, the separator is missing between the collection item.
- Fixed an issue where exception occurs when using InputDate in a list.
- Fixed an issue where error occurred while changing the "SourceCollection" of CollectionView.
- Fixed an issue where FlexGrid CellTemplate cells does not hide when cell edit mode is completed.

PdfViewer

Enhancements

- Added search feature.
- Added license verification process.

CollectionView

Bugs

- Fixed an issue where OnClientPageChanged and OnClientPageChanging events were not getting invoked after applying paging.
- Fixed an issue of unit test CollectionViewServiceTests.VirtualScrollingTest.

FlexViewer

Enhancements

- Added rotate document and rotate page tool.
- Added mouse mode tool.
- Added ZoombySelection and Magnifier tool.
- Added ThersholdWidth property.
- Added ZoomMode property for FlexViewer.

Bug Fix

- Fixed an issue where JavaScript error occurred while working with Print dialog.
- Fixed forward and backward functionality for MobileViewer.

OLAP

Enhancements

- Renamed Error property of PivotEngine to OnClientError event.
- [PivotEngine] Added ViewDefinition for ASP.NET Core application.
- Added new aggregate functions in Olap Server.

Bugs

- Fixed an issue where JavaScript error is displayed while setting Fields of DataEngine to multiple fields.
- Removed "auto-generate-fields" attribute from PivotEngine tag helper.

Input

Bugs

- Fixed OnClientFormatItem event of MultiSelect.
- Fixed an issue where drop-down list shows InputColor after setting "IsReadOnly" and "IsDroppedDown" to true.

FlexSheet

Bugs

- Removed NewRowAtTop property.

Sunburst Chart

Bug Fixes

- Fixed an issue where tooltip stops working while working with "ShowAnimation".

FlexGrid

Enhancements

- Refactoring customized editor for FlexGrid.

Bugs

- Fixed an issue where JavaScript error occurred when setting the ShowColumnFooters property for FlexGrid, which uses cell templates.

- Fixed an issue where Value Filter cannot select an item which contains HTML element.
- Fixed an issue where ComboBox when used as the Edit

FlexPie

Bug Fixes

- Fixed an issue where alpha of pie slices is always set to 0.7 even when setting a custom palette with alpha value as 1.

MultiSelect

Bug Fixes

- Fixed an issue where setting CheckedIndexes and CheckedValues together will throw an exception.

Back to Top

2017 v1

Core

Enhancements

- Added Polar & Radar Chart.
- Added Error Bar and Box-Whisker Series.
- Added Funnel Chart as new chart type in FlexChart.
- Added a new MultiRow control.
- Added scaffolding options for the new controls.
- Updated project templates for Visual Studio 2017

FlexViewer

Enhancements

- Added MobileViewer support for Report/PDF Viewer.
- Added ThersholdWidth property.
- Added ZoomMode property for FlexViewer.

Bug Fix

- Fixed an issue where pages in the Viewer cannot be scrolled using the "Mouse-Wheel" when 'Single Page View' is set.
- Fixed an issue where error is observed after clicked on the forward/backward button when continuous page is set.
- Fixed an issue where JavaScript error is occurred when 'ZoomMode' value is set as 'PageWidth' or 'WholePage'.
- [PDF] Fixed an issue where page setup is shown in mobile view and exception throw when page setup is set.
- Fixed an issue where export button throws an exception when you click on it using Internet Explorer.
- Fixed an issue where some inconsistency was observed in Export setting of PdfViewer.
- Fixed an issue where, if continuous page is set, error occurs when enter page number which is larger than total page count+1 in navigation text box.

OLAP

Bugs

- Fixed the issue where "Show Detail" in context menu should be hidden, when data source of PivotEngine is Cube.
- Fixed the problem related to DimensionType.

Input

Bugs

- Fixed OnClientFormatItem event of MultiSelect.

FlexSheet

Bugs

- Fixed an issue which was caused due to support of JSZip3.0

Sunburst Chart

Bug Fixes

- Fixed an issue where Sunburst slices are not rendered when the ChildItemsPath contains multiple property names.

Project Templates

Bug Fixes

- Fixed an issue where user was not able to create ComponentOne MVC 3 project in VisualStudio 2012 using C1 project template.

FlexGrid

Enhancements

- Refactoring customized editor for FlexGrid.

Bugs

- Fixed an issue where memory usage increases when navigating between web pages rendering C1FlexGrid.

Samples

Enhancements

- Samples now support Visual Studio 2017
- Updated FlexSheetExplorer Sample.

Bug Fixes

- Fixed an issue where error occurred when any row is selected and deleted in 'Custom Editors' sample page.
- Fixed an issue where pager show incorrect page count and can't navigate to another page in "Filtering" sample page of Core project.
- Fixed an issue where setting "stacked 100%" takes no effect in sample.
- Fixed an issue where JavaScript error is occurred when zero value is selected for TotalAngle.
- Fixed an issue where undo button on the toolbar was not working.
- Fixed an issue where JavaScript error occurs after importing the Excel file.

Back to Top

2016 v3.5

Breaking Changes

There is a breaking change in the **ServiceUrl** property of **ReportViewer** control. If you want to use external report services, set the **ServiceUrl** property to "http://c1webapi host/api/report" other than "http://c1webapi host/api". The breaking change is also valid for **PDFViewer** control.

Core

Enhancements

- Added support for Visual Studio 2017
- Updated project templates for Visual Studio 2017

FlexViewer

Bug Fix

- [Report] Fixed an issue where the print option was not working in any browser.
- [PDF] Fixed an issue where continious page and single page was not working correctly.

OLAP

Enhancements

- Added license support to OLAP MVC control.

Sunburst Chart

Bug Fixes

- Fixed an issue where Sunburst slices are not rendered when the ChildItemsPath contains multiple property names.

FlexReport Explorer

Bug Fix

- Fixed the issue where isRequired warning message is shown in console when clicking Customers near Stores report of SSRS.

Project Templates

Bug Fixes

- Fixed the issue where user cannot restore the application which is created by ".NET FW4.5"-->C1 ASP.NET Core MVC Application (.NET Framework).
- Fixed the issue where "project.json used by another process" error pops up when creating new project using c1mvc template.

Samples

Enhancements

- Samples now support Visual Studio 2017
- Updated FlexSheetExplorer Sample.

Bug Fixes

- [FlexSheet101] Fixed the issue where JS error displays after clicking "Export/Import" in "FlexSheetExplorer-->Excel Service" sample.
- [MVCExplorer] Fixed the issue where user cannot export the data to pdf file in "MVCExplorer-->PDF-->OverView/RichGraphics" sample.
- [Input] Fixed an issue where JavaScript runtime errors occur when value less than min value is set in InputNumber.

Back to Top

2016 v3

Core

Enhancements

- Added Sunburst chart.
- Added OLAP control.
- Added PdfViewer control.

Bug Fix

- Fixed the issue where setting one control's C1EnableDeferredScripts to false makes other control's DeferredScripts take no effect.

FlexChart

Enhancement

- Added Waterfall series to FlexChart.

Bug Fix

- Fixed the issue where JS error is thrown after setting ChartType from "Column" to "SplineArea" in "MVCExplorer" sample.

FlexGrid

Bug Fix

- Fixed the issue where Javascript error is observed while editing in cell using custom editor.

Scaffolder

Enhancements

- Shortened the names for C1Scaffolder.
- Added Scaffolder for Sunburst chart.

Bug Fix

- [FlexChart] Fixed the issue where an extra dropdown is found for Binding X value in 'General' tab of FlexChart scaffold.

FlexViewer

Enhancements

- Implement press 'Esc' to exit full screen mode.
- Show full screen toolbar for a short time when change to full screen mode.
- [ReportViewer] Updated the export menu after loading the document source.
- [ReportViewer] Implemented Move button.
- [ReportViewer] Added First Page, Last Page, Backward History, and Forward History buttons.
- [ReportViewer] Added Full Page view, where page can enter in full screen mode without toolbar.

Bug Fixes

- [Report Viewer] Fixed the issue where the parameters in drop-down list with the checkbox cannot be changed.
- [ReportViewer] Fixed the issue where UI is distorted in 'Normal Page View' when report pages are navigated in 'Full Page View' and it is return back to 'Normal Page View'.
- [ReportViewer] Fixed the issue where 'Select Tool' of ReportViewer did not work while performing selecting on iPad Air.
- [ReportViewer] Fixed the issue where user is unable to print document in Firefox.
- [ReportViewer] Fixed the issue where exporting and printing do not work on iPad Air.

FlexReport Explorer

Bug Fix

- Fixed the issue where isRequired warning message is shown in console when clicking Customers near Stores report of SSRS.

Project Templates

Bug Fixes

- Fixed the issue where compiling error displays after rebuilding the application that is created by MVC template with "Add TS file" checked.
- Fixed the issue where "project.json used by another process" error pops up when creating new project using c1mvc template.

Samples

Bug Fixes

- [FlexChart][AxisScrollbar] The "readme.txt" file is displayed as a non-existing file when opened in Visual Studio.
- [FlexSheet101] Fixed the issue where error occurs when VB FlexSheet101 sample is run.
- [FlexSheet101] Fixed the issue where readme.txt file cannot be found in FlexSheet101.

Back to Top

2016 v2.5

Enhancements

- Added new samples for Unobtrusive Validation support in FlexGrid and Input.

Scaffolder

Bug Fixes

- Fixed an issue where it was suggested to add label for Content property of DataLabel.

- Fixed an issue where some needless error codes were generated by FlexChart Scaffold.

FlexViewer

Bug Fix

- Fixed an issue where setting a parameter creates a problem in Cascading Parameters report.

FlexSheet

Bug Fix

- Fixed an issue where it was required to add "AllowAddNew" property in Asp.NetCore project.

FlexViewer Sample

Bug Fix

- Fixed the issue where error occurred after choosing Report File or Report Name from dropdown when FlexViewer sample is deployed in IIS.

MVCExplorer

Bug Fixes

- Fixed wrong description in MvcExplorer FlexGrid Custom Footer page.
- Fixed the issue where User is unable to change values of existing data in FlexGrid.

Samples

Bug Fix

- Fixed an issue where FlexGrid should not be editable in the save page of GridInForm sample.

ReportExplorer

Bug Fixes

- Fixed an issue where console error was observed in FlexReportExplorer Sample.
- Removed map report under controls folder of FlexReportExplorer Sample.

FinancialChartExplorer

Bug Fix

- Fixed the issue where JavaScript runtime error occurs if any value that is less than min value or more than max is entered to inputNumbers.

Back to Top

2016 v2

Enhancements

- Updated C1.Web.MVC to C1.AspNetCore.Mvc for ASP.NET Core applications.
- Added InputDateTime control as per Wijmo 5
- Added a new Report Viewer control

- Updated Scaffolder VStemplate for InputDateTime.
- Updated the FlexViewExplorer sample for MVC5 and ASPNETCore.
- Added C1 MVC JS intellisense support,.
- Added Validation sample for InputDateTime control.

Core

Bug Fix

- Fixed the issue where integer variable's value was zero when project was created using Visual Studio C1 MVC template.

Scaffolder

Enhancements

- Update ComponentOne Scaffolder for ASP.NET Core.
- Added scaffolding for FlexChart
- Added scaffolding for FlexPie
- Added scaffolding for FlexSheet

Bug Fixes

- Fixed an issue where compiler error was generated in InputController.cs after adding Wijmo5 Input Scaffolding.
- Fixed an issue where error occurs if remote load and save are used.

ComboBox

Enhancements

- Added HeaderPath property on server side.

FlexGrid

Enhancements

- Support two new properties ImeEnabled and DropDownCssClass property.

FlexChart

Enhancements

- Added PlotArea support in FlexChart.
- Added support for Chart.Animation

Bug Fixes

- Fixed an issue where browser will crash after running FlexChart which contains "TrendLine" with AxisX/Y setting in ASPNETCore application.
- Fixed FlexChart cannot render correctly in MVCExplorer.
- Fixed an issue where thumb buttons in range selector are doubled.

FlexPie

Bug Fix

- Fixed FlexPie's SelectedIndex property.

InputTime

Enhancements

- Changed type of Step property in InputTime from double to integer.

FlexViewer

Bug Fixes

- Fixed an issue where Label on Page Thumbnails and PArameters panel is shown incorrect.
- Fixed an issue where changing the report quickly will throw exception in console.

FlexSheet

Enhancements

- Support Formula Bar as an extension of FlexSheet.

Input (Dropdown)

Enhancements

- Added AutoExpandSelection property.
- Added DropDownCssClass property.

FlexViewer Sample

Bug Fix

- Fixed the issue where error occurred after choosing Report File or Report Name from dropdown when FlexViewer sample is deployed in IIS.

MVCExplorer

Enhancements

- Added sample for DropDownCssClass property.

Bug Fixes

- Fixed compiling error in the "ASPNETCore>MVCExplorer>FlexChart
- Fixed the issue where popup was not working when open in MvcExplorer.
- Fixed an issue of sample project, data label at the bottom of FlexPie is covered.
- Fixed the status of checked items are wrong after clicking "Submit" in "MultiSelect>Form" MVCExplorer sample

Samples

Bug Fixes

- Removed _Pager partialviewer from samples.
- Added another section in DataMap sample for demonstrating multicolumn DataMap.
- Fixed the issue where an error was displayed after exporting FlexChart to image file in C1MVCFlexChartExport sample.

ASPNETCoreInput101

Bug Fix

- Fixed. Set the "value-changed" event to "inputDateTime_ValueChanged" in "..\ASPNETCore\HowTo\Input\ASPNETCoreInput101" sample.

FlexReportExplorer

Enhancement

- Increased the indent of the navigate bar in sample.

Back to Top

2016 v1.5

Core

Enhancements

- Added TypeScript IntelliSense support for MVC controls.

Bug Fixes

- Fixed the issue where integer variable's value was zero when project was created using Visual Studio C1 MVC template.
- Copyright information updated to 2016

CollectionView

No change

FlexViewer Sample

Bug Fix

- Fixed the issue where error occurred after choosing Report File or Report Name from dropdown when FlexViewer sample is deployed in IIS.

MVCExplorer

Bug Fixes

- Fixed the issue where 'And' and 'Or' radio buttons are not displayed properly in 'Disable Server Reading' sample.
- Fixed the issue where Sample description is inconsistent with the actual grid in 'Disable Server Reading' sample.
- Fixed the issue where RadialGauge value range is not shown correctly at Scaling in MvcExplorer sample.
- Fixed Download link on the demos page.

HowTo Samples

Bug Fixes

- Fixed the issue in SignalR sample, where changing some columns value to some format, in another sheet becomes "0".
- Fixed the issue where inconsistent behavior was observed in "StockChart" sample in some cases.
- Fixed the issue where 'Country' column was read-only which is inconsistent with the sample description.

Samples

Bug Fixes

- Fixed the issue where a user opens MVC_ASPNET5Gauge101 sample will pop up link to Team Foundation Server window.
- Fixed the issue where the dialog use to flicker after rendering popup sample.
- Removed useless _Pager partial viewer from samples
- Fixed the issue where an error was displayed after exporting FlexChart to image file in C1MVCFlexChartExport sample.

Back to Top

2016 v1

Core

Enhancements

- Added FlexSheet control.
- Added Pager control.
- Added jszip cdn to Visual Studio Template for FlexSheet.

CollectionView

No change

FinancialChart

Enhancements

- Added new Overlays and Indicators for financial charts.

FlexGrid

Enhancements

- Added Scaffolder for FlexGrid control.
- Added jszip.js to FlexGrid resource.

Bug Fixes

- Fixed the issue where Condition Filter does not work correctly while using AND/OR operator in ID column.

FlexChart

No change

Gauge

BugFixes

- Fixed the issue where Gauge value shows as '100' even though its value is set as '0' in all gauges types.

Input

Enhancements

- Added Scaffolder for input controls.

Bug Fixes

- Fixed the issue where dropdown button of the AutoComplete is no longer visible when the value which does not exist in the drop-down list is entered, although isEditable is 'False'.

MVCExplorer

Enhancements

- Line gauges are added/covered in MvcExplorer.

Bug Fixes

- Fixed the issue where popup dialog closes immediately without showing any message on clicking "Facebook" or "Google".
- Fixed the issue where Filtering does not work correctly in FlexGrid's Filter sample of MVCExplorer.
- Fixed the issue where SelectedValue and SelectedItem properties could not be set from Model.
- Fixed the issue where the "isDroppedDownChanged" event does not fire when 'Context Menu' is opened.
- Fixed the issue where web browser shows error message for Scaling page in Gauge sample.
- Fixed the issue where Custom Editors do not work correctly in Date column.
- Fixed the issue where the selected items of Stacking and ChartTypes do not work correctly at the first time in FlexChart's Selection.
- Fixed the issue where web browser shows error message on importing an excel file.

FinancialChartExplorer

Bug Fixes

- Fixed the issue where DateValue in ToolTip shows incorrect format.
- Fixed the issue where inconsistent behavior is observed in 'Fibonacci Tool' of FinancialChart's Analytics sample.

HowTo Samples

Bug Fixes

- Fixed the issue where 'ColumnPicker' icon is lost after deploying the ColumnPicker sample in IIS.
- Fixed the issue where'C1Icon' is lost after deploying the FlexChartGroup sample in IIS.
- Fixed the issue where Filter value cannot be displayed correctly in value filter of Discount column.
- Fixed the issue where JavaScript error is displayed in "HowTo\FlexGrid\C1MVCExcelImportExport" sample.
- Fixed the issue where the cells do not synchronize data in two browsers after editing one cell at the same time.
- Fixed the issue where error occurs when the 'Save Column Layout To Server' button is clicked after deploying the sample in IIS.
- Fixed the issue where error occurrs when 'C1MVCFinance' sample is run.
- Fixed the issue where JavaScript error is observed after applying filter with decimal value in max price.
- Fixed the issue where images are not displayed after deploying the FlexChartEditableAnnotationLayer sample in IIS.

Back to Top

2015 v3.5

Core

No change

CollectionView

No change

FlexGrid

Bug Fixes

- Fixed the issue where 'System.ArgumentException' occurs on filtering by condition when any filter condition is chosen and filter value is empty.

FlexChart

No change

Input

Breaking Change

- Added the following helper methods in ControlBuilderFactory<TModel> class to support nullable values in C1Input controls:
  - InputColorFor(Expression<Func<TModel, Color?>> expression)
  - InputDateFor(Expression<Func<TModel, DateTime?>> expression)
  - InputNumberFor(Expression<Func<TModel, decimal?>> expression)
  - InputNumberFor(Expression<Func<TModel, double?>> expression)
  - InputNumberFor(Expression<Func<TModel, int?>> expression)
  - InputNumberFor(Expression<Func<TModel, long>> expression)
  - InputNumberFor(Expression<Func<TModel, long?>> expression)
  - InputNumberFor(Expression<Func<TModel, float>> expression)
  - InputNumberFor(Expression<Func<TModel, float?>> expression)
  - InputNumberFor(Expression<Func<TModel, short>> expression)
  - InputNumberFor(Expression<Func<TModel, short?>> expression)
  - InputTimeFor(Expression<Func<TModel, DateTime?>> expression)

MVCExplorer

- Updated the Annotation sample for FlexChart.
- Added CustomMerging, Client Excel load/save, and Pdf export samples to Explorer sample of FlexGrid.
- Added Scaling, Panning/Zooming, PlotAreas, EditableAnnotation, and AxisScrollBar samples to Explorer sample of FlexChart.
- Added Scaling sample to explorer sample of gauges.

FinancialChartExplorer

- Fixed the issue where error occurs on setting Range Mode to 'ATR' when Reversal Amount is set to '7' in FinancialChart-Kagi sample.

2015 v3

Core

Enhancements

- Provided support for ASP.NET 5 Release Candidate.
- Added Financial Charts

- Added FlexChart and Financial chart annotations.
- Added ASP.NET 5 Beta 7 support to C1 MVC controls.
- Refactored client side extender.
- Fixed the issue where multiple issues occured while running MVCExplorer sample with trial version.

CollectionView

Enhancements

- Updates MVC VS template to add support for finance.

Bug Fixes

- Fixed the issue so that querydata event doesn't fire when do CRUD(Create/Update/Delete/Edit) operation with DisableServerRead lis set to true.

FlexGrid

Bug Fixes

- Fixed an issue so that 'cellEditEnding' event fires when custom editors are used.
- Fixed an issue so that "undefined" is not displayed in the cell of a new row when celltemplate is applied to mvc flexgrid.
- Fixed an issue so that Loading event and loaded event fire in page load of Hierarchical grid.
- Fixed an issue so that 'tab' key action works properly in child grid and javascript error does not occur.

FlexChart

Enhancements

- Added Annotations in FlexChart .

Bug Fixes

- Fixed the issue that FlexChart Palette property works correctly.

Input

AutoComplete

Bug Fixes

- The 'isContentHtml' property works correctly now.

FlexGrid

InputNumber

- Fixed the issue so that the separator between textbox and dropdown button does not disappear in Input control.

Menu

Enhancements

- Added Owner property to Menu control in order to support contextmenu.

Bug Fixes

- The 'isContentHtml' property works correctly now.

Multi Select

Bug Fixes

- Fixed the issue that "CheckedValues" and "CheckedIndexes" work correctly in MultiSelect.
- The "MaxHeaderItems(0)" property of MultiSelect works as default value.

MVCExplorer

- Changed Default document mode of IE browser to IE7.
- Added collectionview101 sample link to Home page.
- InputTime's value property works properly in "Custom Time" sample.
- Fixed the issue, wherein Input Date Custom Editors is displayed although it is exit from edit mode when no changes in custom editor.
- Fixed the issue that FlexGrid "CustomEditor" sample does not throw js error in IE.
- Dropdown lists item are shown correctly after selecting combobox item in ComboBox or ItemTemplate sample.

TagHelperExplorer

- Added FlexPie101, Gauge101 andInput101 sample links to Home Page.
- Fixed the issue that FlexGrid "CustomEditor" sample does not throw js error in IE.

2015 v2.5

Core

Enhancements

- Provided support for ASP.NET 5 beta 7.

FlexGrid

Enhancements

- Added support for DetailRow functionality.
- Added support for DataMap functionality.

Bug Fixes

- Fixed the issue where filter does not work properly on using value filter after applying search.

Input

Enhancements

- Added Multi-select control.
- Added Popup control.

MVC Explorer

- Fixed the issue where nothing shows in FlexGrid on switching the source and sample tabs after performing scrolling operation in FlexGrid.

2015 v2

Core

Enhancements

- Added MVC 6 compatibility support to MVC controls.
- Added TagHelpers support to MVC controls.

Bug Fixes

- Fixed issue where JS occurs after running some FlexGrid of MVCExporer sample in IE9.

FlexChart

Enhancements

- Added LineMarker Extender for FlexChart.
- Added event property "OnClientRendering" to FlexChart ChartSeries.
- Added Drag mode support for LineMarker.
- Added Trendline to FlexChart.

Bug Fixes

- Fixed an issue where JS error occurs when creating an empty FlexChart.
- Fixed the "Financial Chart" sample, where data disappeared after zoom operation/ChartType settings.
- Fixed unexpected behaviors found in chart.
- Fixed issue so that the threshold Tooltip works correctly in the Flex.

FlexGrid

Enhancements

- Added a FlexGrid.defer Resizing property to defer row and column resizing until the user releases the mouse.
- Added SortRowIndex property to the FlexGrid.
- Added SortMemberPath property to flexgrid's column.
- Added group panel support for flexgrid.
- Enhanced FlexGrid filtering feature.

Bug Fixes

- Fixed the issue that an exception is thrown when performed sorting or filterings of no-generic data.
- Fixed issue so that the data in filter dialog is display correctly in some condition.
- Fixed an issue so that the collapse/expand icons is displayed correctly on applying column grouping in Group Panel sample.
- Fixed the issue so that the filter result is correct on setting different values in the value filter editor twice.
- Fixed an issue so that the total page count remains unchanged after performing default value filtering for numeric value.
- Fixed issue so that the filter works correctly if filter by "End" or "Discount"column is applied in MvcExplorer sample.
- Fixed issue so that the license error is observed after trial license is expired.
- Fixed issue so that the JS error doesn't occur on filtering grid using "CellTemplate"and "FrozenRows" setting.
- Fixed the FlexGid's filtering sample, in which same filter value are shown in the value filter editor of "Discount"/"End" column.
- Filter editor is closed while applying filtering operation on column with cell template in IE.
- Fixed issue so that the FlexGrid control is updated when returning a partial view from the controller method called through an AJAX call.
- Fixed issue so that the data is displayed on performing drag virtual scrolling's thumb button to bottom after sorting operation.

- Fixed the issue so that the JS error will not be thrown after editing the cell from the first row to the last in "Custom Editors".
- Fixed issue so that the Javascirpt script error is not observed when scrolling the web page after opening the custom drop down and selecting an item.
- Fixed an issue so that the Checkbox cell doesn't need to be checked twice when checking the checkbox column for second time in FlexGrid.
- Fixed issue so that the filter result is correct on performing "ESC"operation in the filter editor.
- Fixed issue so that JS error is not thrown when one FlexGrid is regarded as Cell Template of another FlexGrid.
- Fixed issue that the data disappears after clicking "Apply" by "Amount"/"Amount2" in MVCExplorer sample.

FlexPie

Bug Fixes

- Fixed the issue where SelectedItemOffset does not take effect when SelectedItemPosition is set to None.
- Fixed the issue where some text appears on top left corner of the screen.

Gauge

Enhancements

- Added Origin property to Gauge

Bug Fixes

- Fixed issue where Error occurs on triggering the OnClientPropertyChanged event in LinearGauge and RadialGauge ranges

Input

Enhancements

ListBox

- Added CheckedMemberPath and OnClientItemChecked property to listbox to support multiple select feature.

Bug Fixes

ComboBox

- Fixed an issue where server error occurs on setting "select-value"/"select-item" of some controls.

InputDate

- Fixed issue so that the border of DropDown-Calendar in InputDate are displayed correctly.

InputNumber

- Fixed issue so that the dropdown list changes to textbox.

MVC Explorer

Bug Fixes

- Fixed FlexGrid BatchEditing issue.
- Made flexgrid readonly in "StarSizing" samples.
- Fixed FlexGrid grouping by date issue so that group header is rendered correctly.
- Fixed an issue so that the "FlexGrid" link is clicked easily on the index page.
- Fixed FlexGrid so that the 'System.FormatException' doesn't occur on generating new items count while menu

dropdown is open on the sample page.
- Javascirpt error is not observed while browsing Editing sample in MVCExplorer.
- [ListBox] Fixed an issue where "JavaScript runtime error: 'selectedIndexChanged' is undefined" is observed in ListBox sample.
- Clicking "OK" in "Editing" sample works after every operation.
- Fixed an issue where “Error: Invalid character" Javascript error occurred after opening the drop down menu without selecting the item and scrolling the FlexGrid.
- [FlexChart] Fixed this issue where Javascript error was observed on hovering legend icon in "Hit test" sample of FlexChart.
- [FlexGrid] Fixed FlexGrid issue where grid rendered without data if switch source and sample tabs after scrolling operation in FlexGrid.