ComponentOne

Xamarin.iOS Controls

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com

Sales: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh , PA USA Office)

Table of Contents

Getting Started with Xamarin.iOS Controls

ComponentOne Xamarin.iOS is a collection of iOS UI controls developed by GrapeCity. Xamarin.iOS Edition has been optimized for iOS development. For existing Xuni users, the new architecture brings many new features listed below:

- **Enhanced performance**
  The new controls should generally perform better than the old controls (sometimes doubling performance). By specifically focusing on the Xamarin architecture, the controls cut out some intermediary logic and are optimized for the platform. Since they're entirely in C#, so you can also expect a more consistent experience.

- **Designer support**
  The new controls should also support Xamarin's designers for iOS and Android applications. This makes it much easier to construct your Android XML or iOS Storyboards using these controls.

- **New control features**
  The controls have been rethought for the new architecture with the combined experience of Xamarin, Wijmo, as well as ComponentOne controls. Some controls have a number additional features (such as FlexGrid).

Breaking Changes for Xuni Users

New Package Names

The packages have changed their prefix if you're coming from Xuni. For instance,

Xuni.iOS.Calendar now corresponds to **C1.iOS.Calendar**

We have also moved to a more consistent naming scheme for our controls based on the following pattern:

C1.[Platform].[ControlName]

For example, FlexGrid is available in **C1.Xamarin.Forms.Grid**

Additionally, FlexChart, FlexPie, and ChartCore have all been consolidated into one single package instead of three different packages. To use FlexChart or FlexPie, you now need to add a single package developed for the platform of your choice:

C1.iOS.Chart

Namespace Changes

We've made some changes to the namespace of the current controls, which are in line with the changes in package names. For example, Xuni.iOS.FlexGrid now corresponds to C1.iOS.Grid.

Minor API Changes

There are some minor changes in API between ComponentOne Xamarin Edition and Xuni. These should mostly amount to additions, slight change in syntax, and use of prefix 'C1' instead of 'Xuni' in class and object names. For FlexChart, however, the control is very actively growing in terms of API, so missing features are intended to be added in the future.

NuGet Packages

The following NuGet packages are available for download:

Package Name	Description
C1.CollectionView	This is the dependency package for the control NuGet packages and is automatically installed when any dependent package is installed.
C1.iOS.Calendar	Installing this NuGet package adds all the references that enable you to use the Calendar control in your Xamarin.iOS application.
C1.iOS.Core	This is the dependency package for the control NuGet packages and is automatically installed when any dependent package is installed.
C1.iOS.Chart	Installing this NuGet package adds all the references that enable you to use the FlexChart and FlexPie controls in your Xamarin.iOS application.
C1.iOS.Grid	Installing this NuGet package adds all the references that enable you to use the FlexGrid control as well as CollectionView interface in your Xamarin.iOS application.
C1.iOS.Gauge	Installing this NuGet package adds all the references that enable you to use the Gauge control in your Xamarin.iOS application.
C1.iOS.Input	Installing this NuGet package adds all the references that enable you to use the Input controls in your Xamarin.iOS application.

Redistributable Files

Xamarin.iOS Edition, developed and published by GrapeCity, inc., can be used to develop applications in conjunction with Microsoft Visual Studio, Xamarin 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.

Control	Redistributable File
Calendar	C1.iOS.Calendar.dll
CollectionView	C1.CollectionView.dll
Core	C1.iOS.Core.dll
FlexChart	C1.iOS.Chart.dll
FlexGrid	C1.iOS.Grid.dll
Gauge	C1.iOS.Gauge.dll
Input	C1.iOS.Input.dll

System Requirements

Xamarin.iOS Edition can be used in applications written for the following mobile operating systems:

- iOS 10 and above (recommended)

Requirements

- Xamarin Platform 3.0.0.482510 and above
- Visual Studio 2017

Windows System Requirements

- Windows 8.1 and above

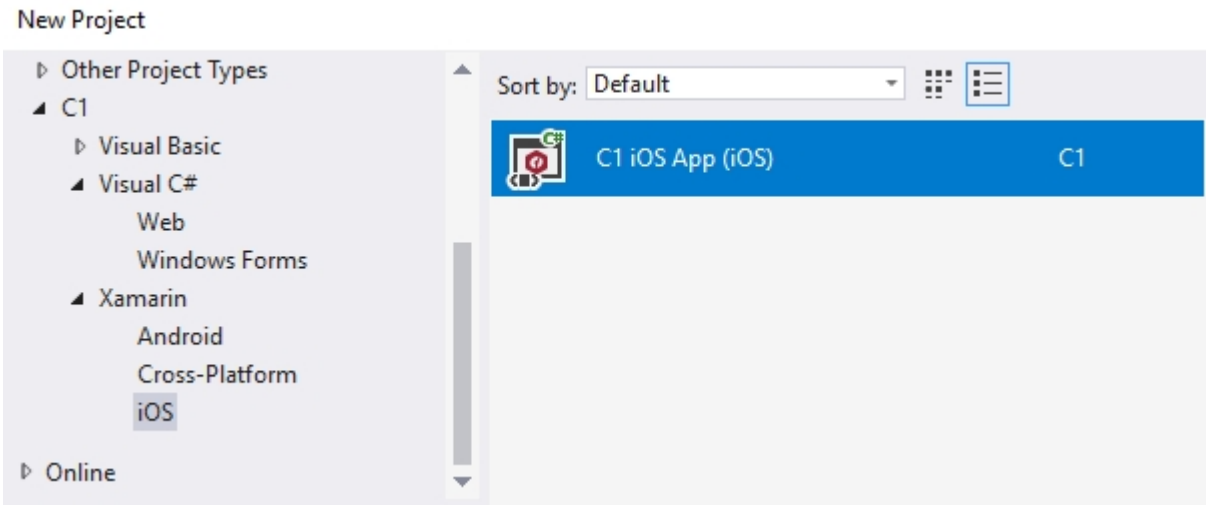
Mac System Requirements

- Visual Studio for Mac
- MacOS 10.12
- Xcode 8 and above

Xamarin iOS Project Templates

When you install **C1StudioInstaller.exe** on your system, we provide different project templates that are available for Xamarin.Forms, Android, and iOS. These templates help you to easily create a Xamarin project with renderer initializers already in place to make use of controls to make development easier. These templates will also help the new users to get started with Xamarin.Forms using C1 Xamarin controls.

- **Xamarin.Forms**
  - Android
  - Cross-Platform
  - iOS



These project templates are similar to the Xamarin.Forms template available in Visual Studio. However, our templates provide a simple mobile application that includes our control references. These generated applications contains all the necessary project files and assembly references to reduce manual work in project creation.

The table below list the C1 Xamarin.Forms project templates.

Project Template (Visual C#)	Description
C1 Android App (Android)	Creates a Xamarin.Android application using ComponentOne Studio for Xamarin. It includes the necessary project files, assembly references, and license information.
C1 Cross-Platform App (Xamarin.Forms Portable)	Creates a Xamarin.Forms application using a Portable Class Library and ComponentOne Studio for Xamarin. It includes the necessary project files, assembly references, and license information.
C1 EntityFramework App	Creates a Xamarin.Forms project using a data grid, Entity Framework, and a SQLite database that allows adding, removing, and updating records.
C1 iOS App (iOS)	Creates a Xamarin.iOS application using ComponentOne Studio for Xamarin. It includes the necessary project files, assembly references, and license information.

Creating a New Xamarin.iOS App

This topic demonstrates how to create a new Xamarin.iOS app in Visual Studio or Xamarin Studio. See the System requirements before proceeding. To download and install Xamarin Studio, visit http://xamarin.com/download.

To know more about Xamarin.iOS, visit:

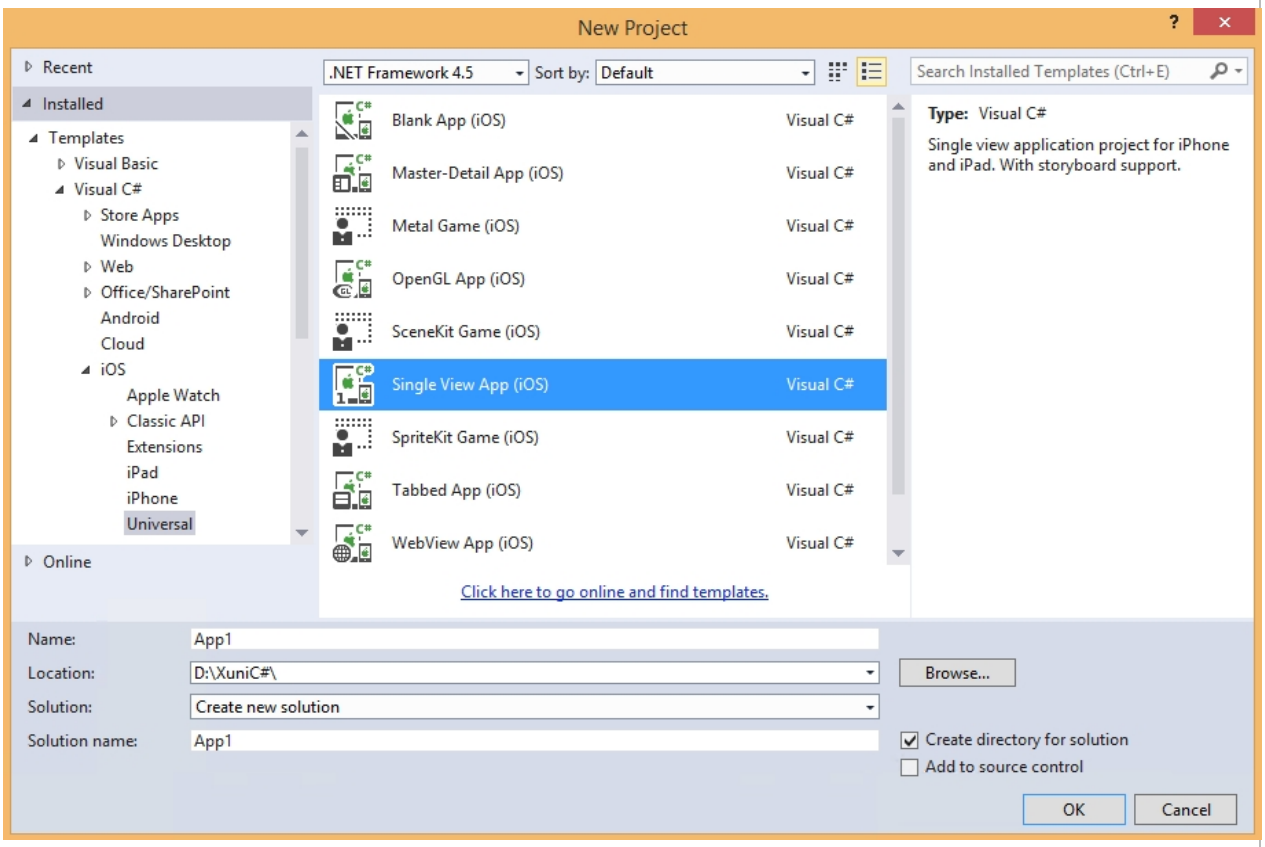
https://developer.xamarin.com/guides/ios/getting_started/

Complete the following steps to create a new Xamarin.iOS App:

Visual Studio (Windows)

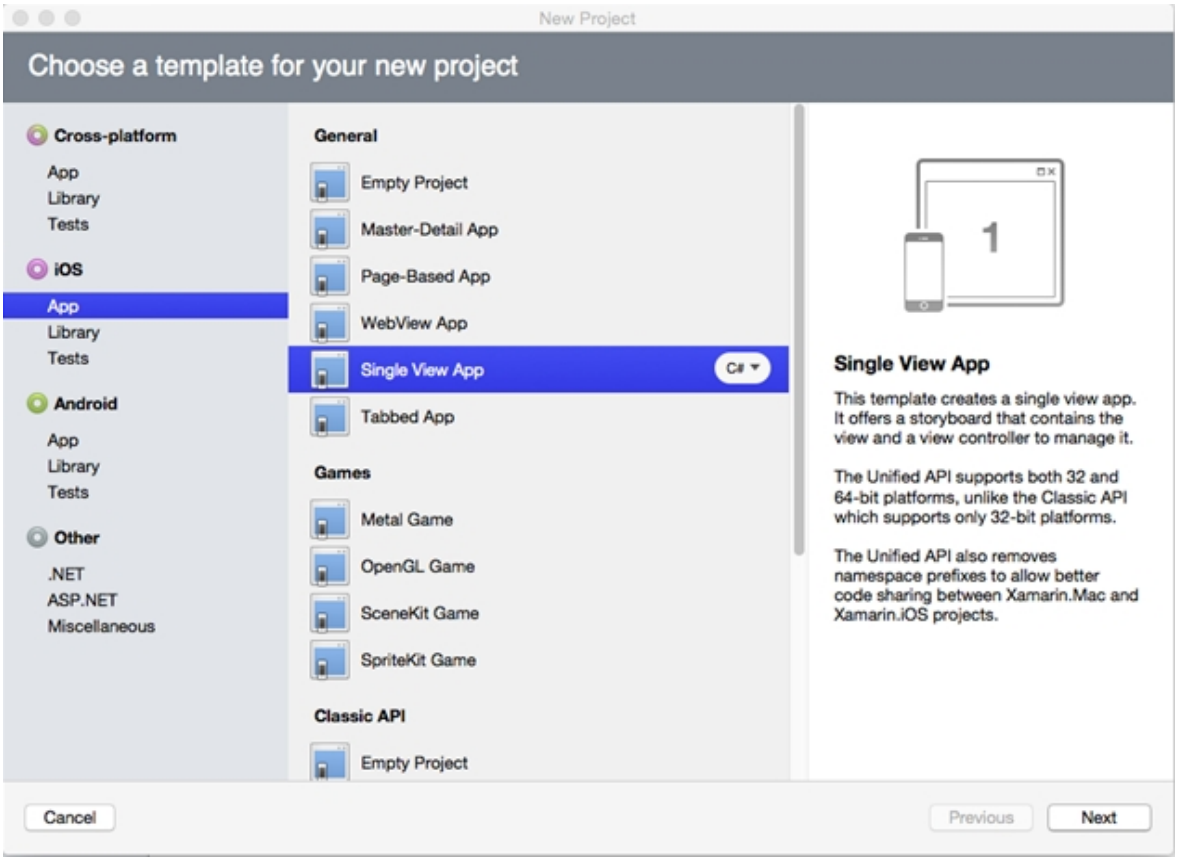
1. Select **File | New | Project**.
2. Under installed templates, select **Visual C# |Universal**.

3. In the right pane, select **Single View App (iOS)**.
4. Type a name for your app and select a location to save it.
5. Click OK.

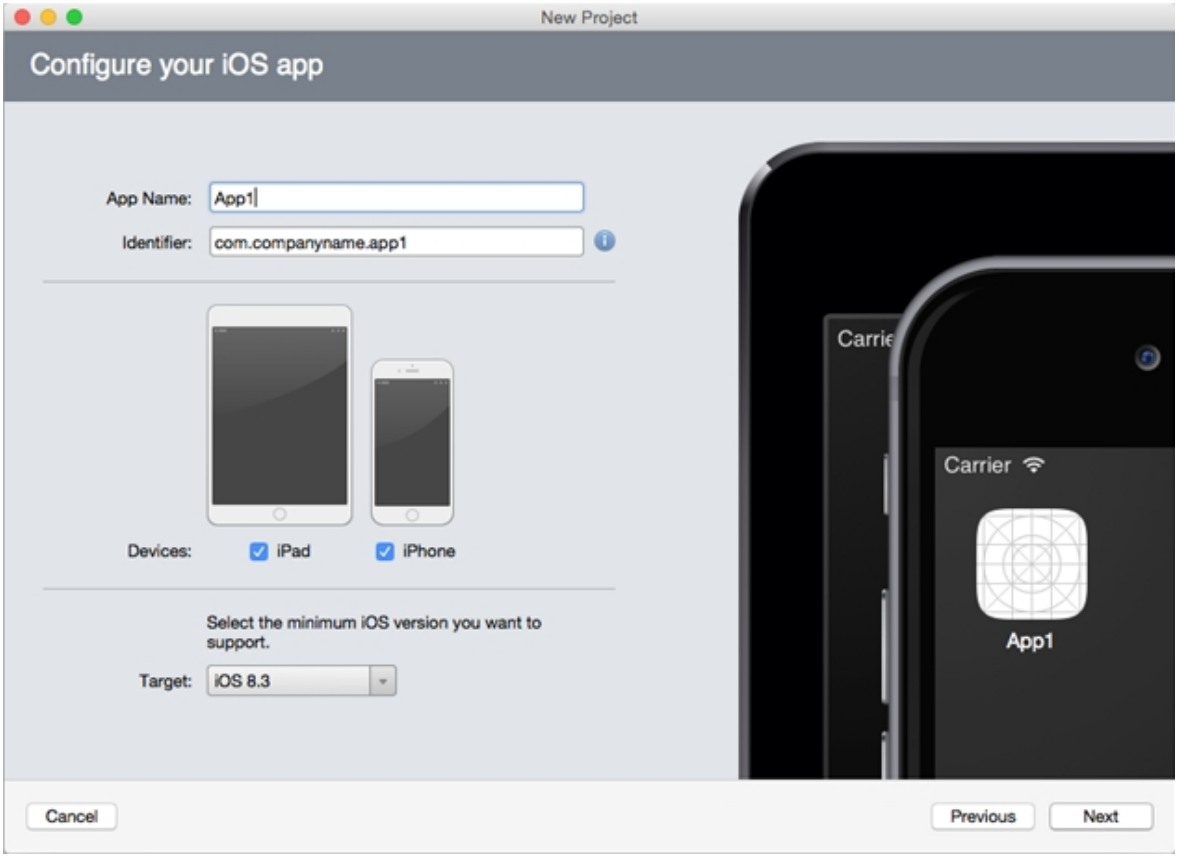


Visual Studio for Mac (macOS)

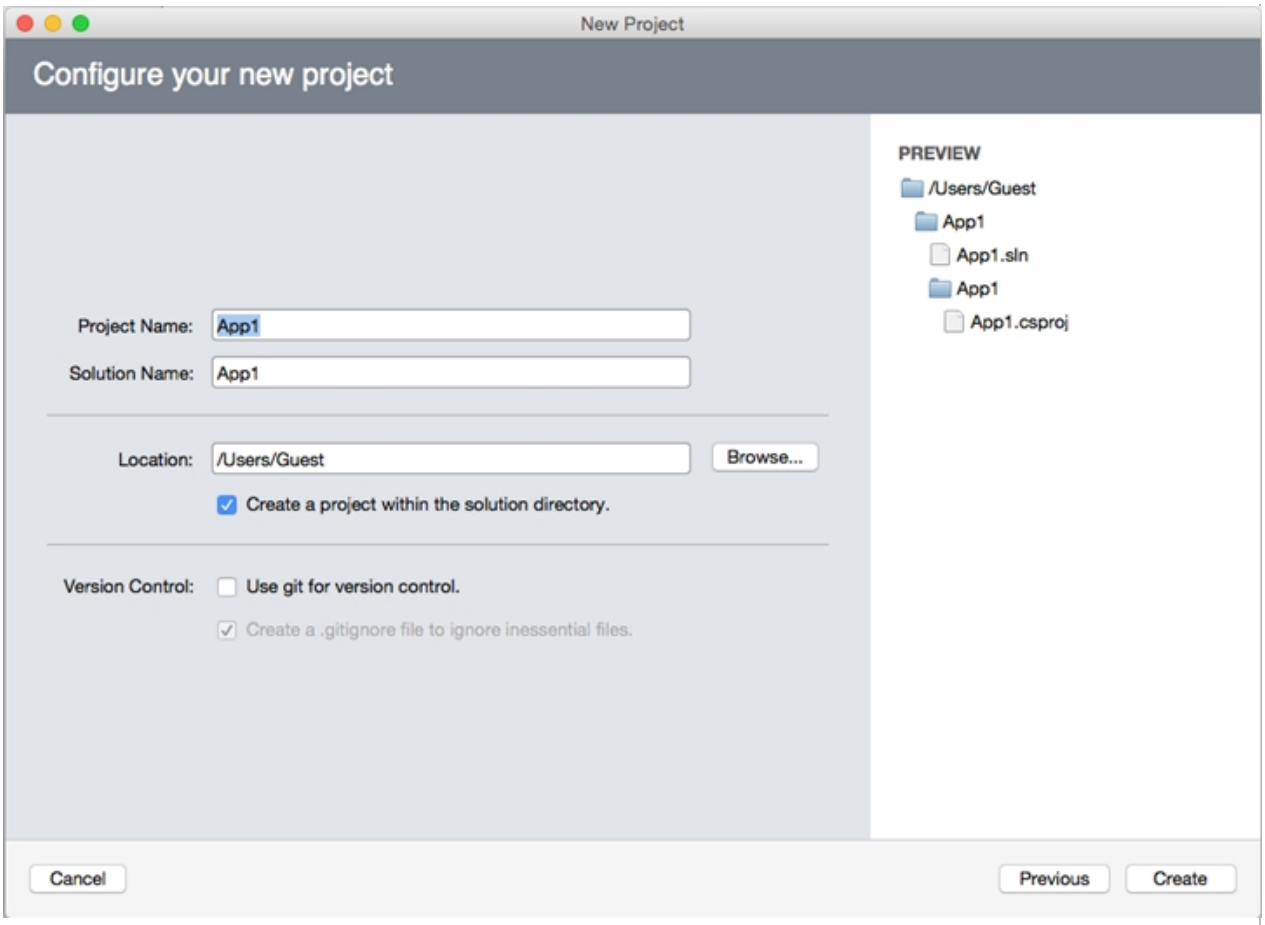
1. Select **File** | **New Solution**.
2. Select **iOS** | **App**.
3. In the right pane, select **Single View App**.
4. Click **Next**.



5. Type a name for your app and select a location to save it..



6. Click **Next**.



7. Click **Create**.

Adding NuGet Packages to your App

To install NuGet

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

To add Xamarin References to your App

In order to use Xamarin controls on iOS references have to be added to your project. Complete the following steps to add Xamarin references to your project.

Visual Studio (PC)

1. Open a pre-existing Mobile App or create a new Mobile App (see Creating a New Xamarin.iOS App).
2. From the Project menu, select Manage NuGet Packages. The Manage NuGet Packages dialog box appears.
3. Click Online and then click GrapeCity.
4. Click Install next to C1.iOS.ControlName (for example C1.iOS.Chart). This adds the references for the Xamarin control.
5. Click **I Accept** to accept the license and then click Close in the Manage NuGet Packages dialog box.

Visual Studio for Mac

1. Open a pre-existing Mobile App or create a new Mobile App (see Creating a New Xamarin.iOS App).
2. In the Solution Explorer, right click the project and select Add | Add Packages. The Add Packages dialog appears.
3. From the drop down menu in the top left corner, select GrapeCity. The available Xamarin packages are displayed.
4. Select the package C1.iOS.ControlName and click the Add Package button. This adds the references for the Xamarin control.

To manually create a Xamarin feed source

Complete the following steps to manually add Xamarin NuGet feed URL to your NuGet settings in Visual Studio or Xamarin Studio and install Xamarin.

Visual Studio (PC)

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 **Add** button in top right corner. A new source is added under Available package sources.
4. Set the Name of the new package source. Set the Source as http://nuget.grapecity.com/nuget/.
5. Click OK. The feed has now been added as another NuGet feed source.

To install Xamarin using the new feed

1. Open a pre-existing Mobile App or create a new Mobile App (see Creating a New Xamarin.iOS App).
2. Select Project | Manage NuGet Packages. The Manage NuGet Packages dialog box appears.
3. Click Online and then click Xamarin. The available packages are displayed in the right pane.
4. Click Install next to C1.Xamarin.Forms.ControlName (for example C1.Xamarin.Forms.Chart). This updates the references for the Xamarin control.
5. Click **I Accept** to accept the ComponentOne license for Xamarin and then click Close in the Manage NuGet Packages dialog box.

Visual Studio for Mac

1. From the Projects menu, select Add Packages. The Add Packages dialog appears.
2. From the drop down menu on the top left corner, select Configure Sources. The Preferences dialog appears.
3. In the left pane, expand Packages and select Sources.
4. Click the Add button. The Add Package Source dialog appears.
5. Set the Name of the new package source. Set the URL as http://nuget.grapecity.com/nuget/.
6. Click the Add Source button. The Xamarin feed has now been added as another NuGet feed source.
7. Click OK to close the Preferences dialog.

To install Xamarin using the new feed

1. Open a pre-existing Mobile App or create a new Mobile App (see Creating a New Xamarin.iOS App).
2. In the Solution Explorer, right click the project and select Add | Add Packages. The Add Packages dialog

appears.
3. From the drop down menu on the top left corner, select Xamarin. The available Xamarin packages are displayed.
4. Select the package C1.[Platform].[ControlName] and click the Add Package button. This adds the references for the Xamarin control.

Licensing

ComponentOne Xamarin Edition contains runtime licensing, which means the library requires a unique key to be validated at runtime. The process is quick, requires minimal memory, and does not require network connection. Each application that uses ComponentOne Xamarin Edition requires a unique license key. This topic gives you in-depth instructions on how to license your app. For more information on GrapeCity licensing and subscription model, visit https://www.componentone.com/Pages/Licensing/.

To know the licensing process in details, see the following links

- Licensing App using GrapeCity ComponentOne Extension
- Licensing you app using website

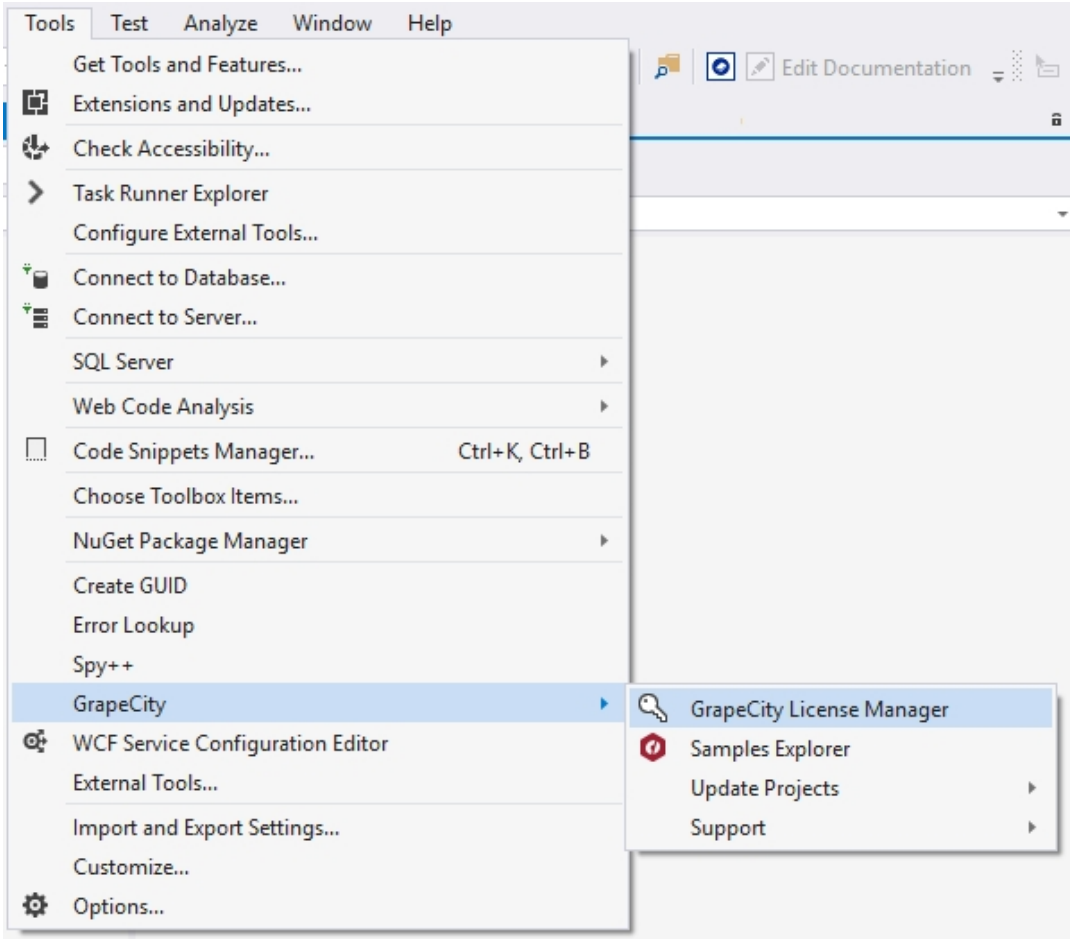
License App using GrapeCity ComponentOneMenu Extension

If you are using ComponentOne controls with Visual Studio 2017 or above, you can use the GrapeCity ComponentOneMenu Extension to **License applications**, open **Sample Explorer**, **update projects** and contact **Technical Support**. After installation, it gets available with various options as shown below.

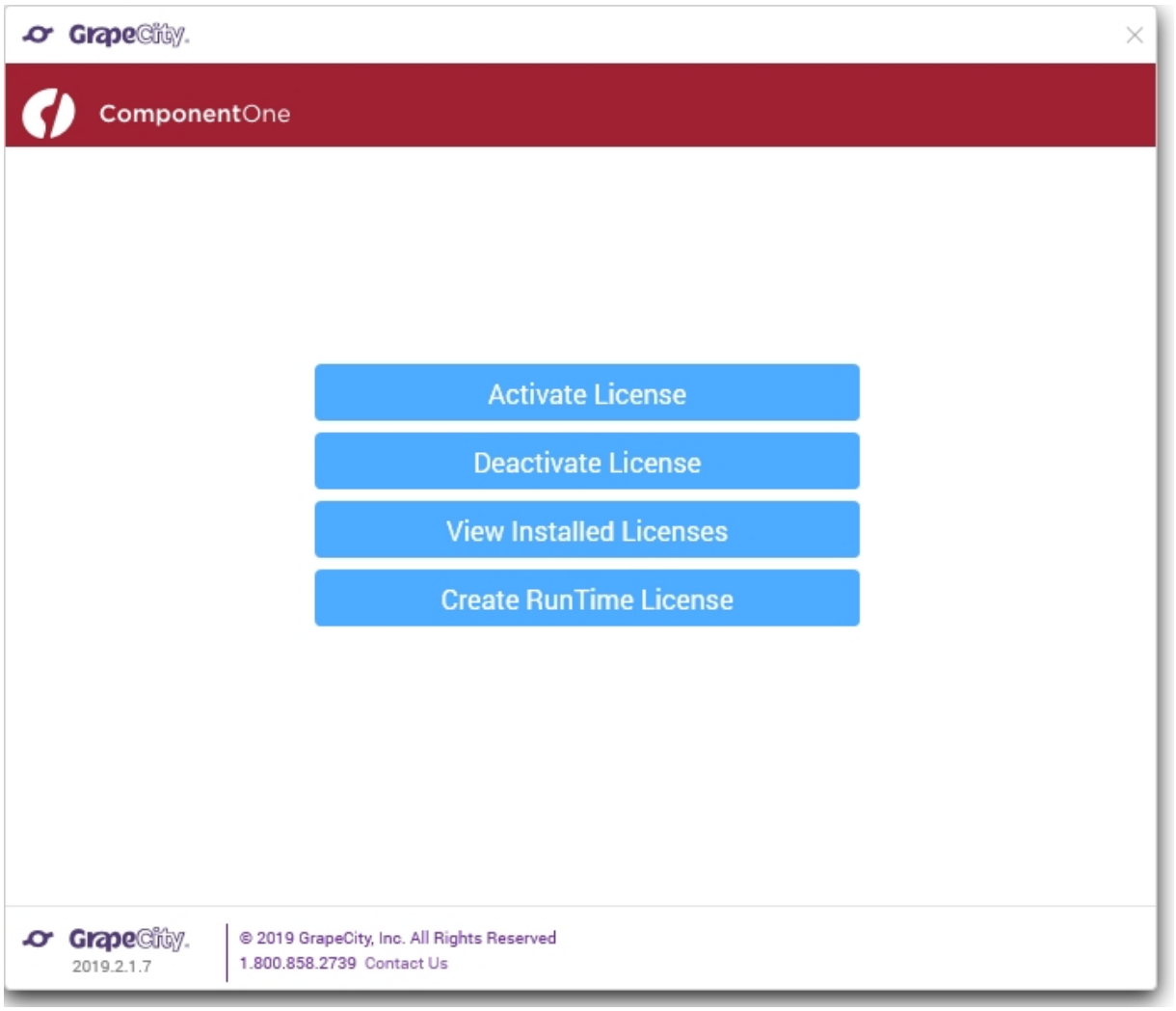
GrapeCity License Manager Extension for Visual Studio

To use GrapeCity License Manager Extension, follow these steps:

1. From the **Tools** menu, select **GrapeCity**. You will see four options as shown in the image below.

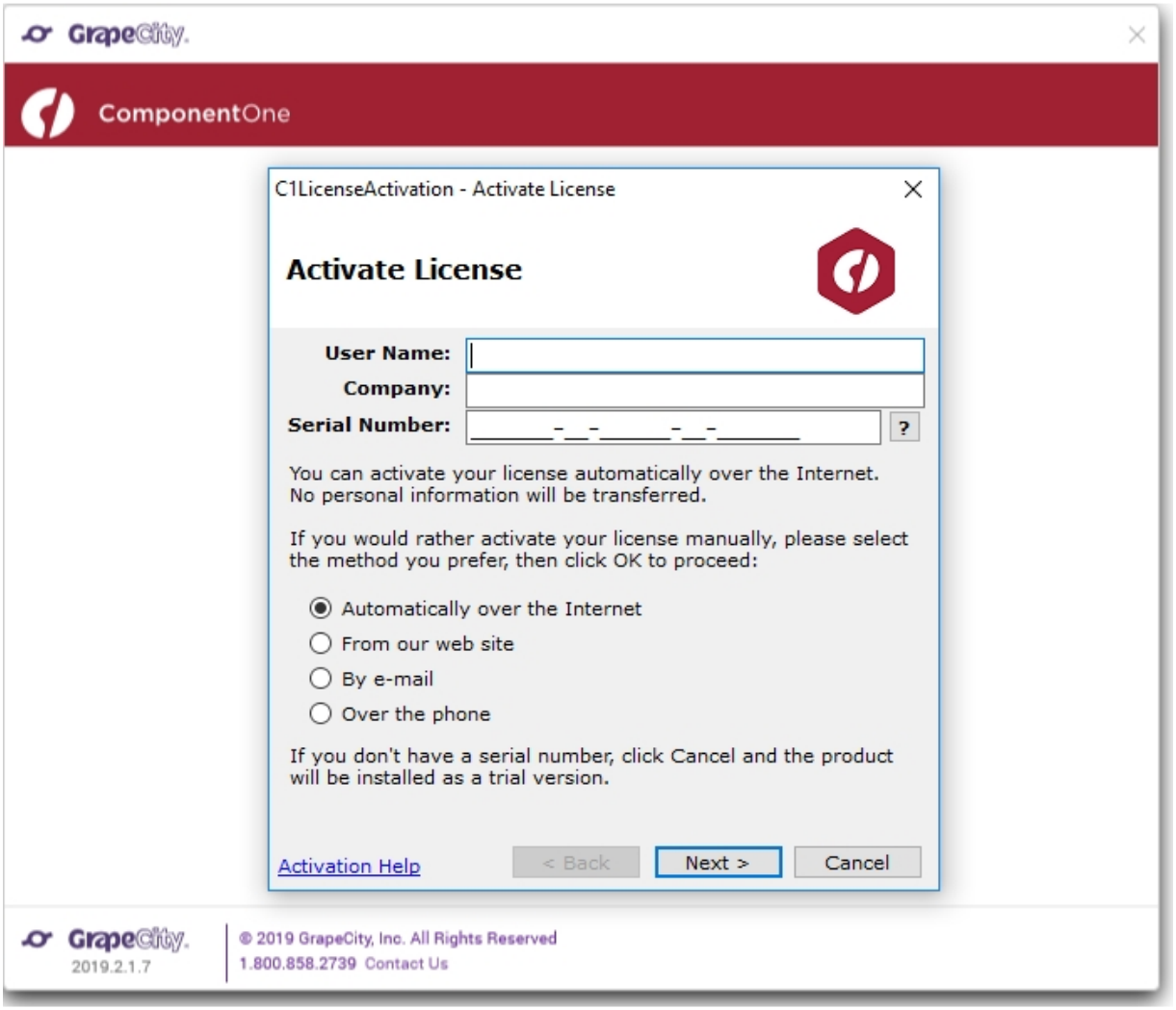


2. Select **GrapeCity License Manager**. The License Manager window appears as shown in the image below:

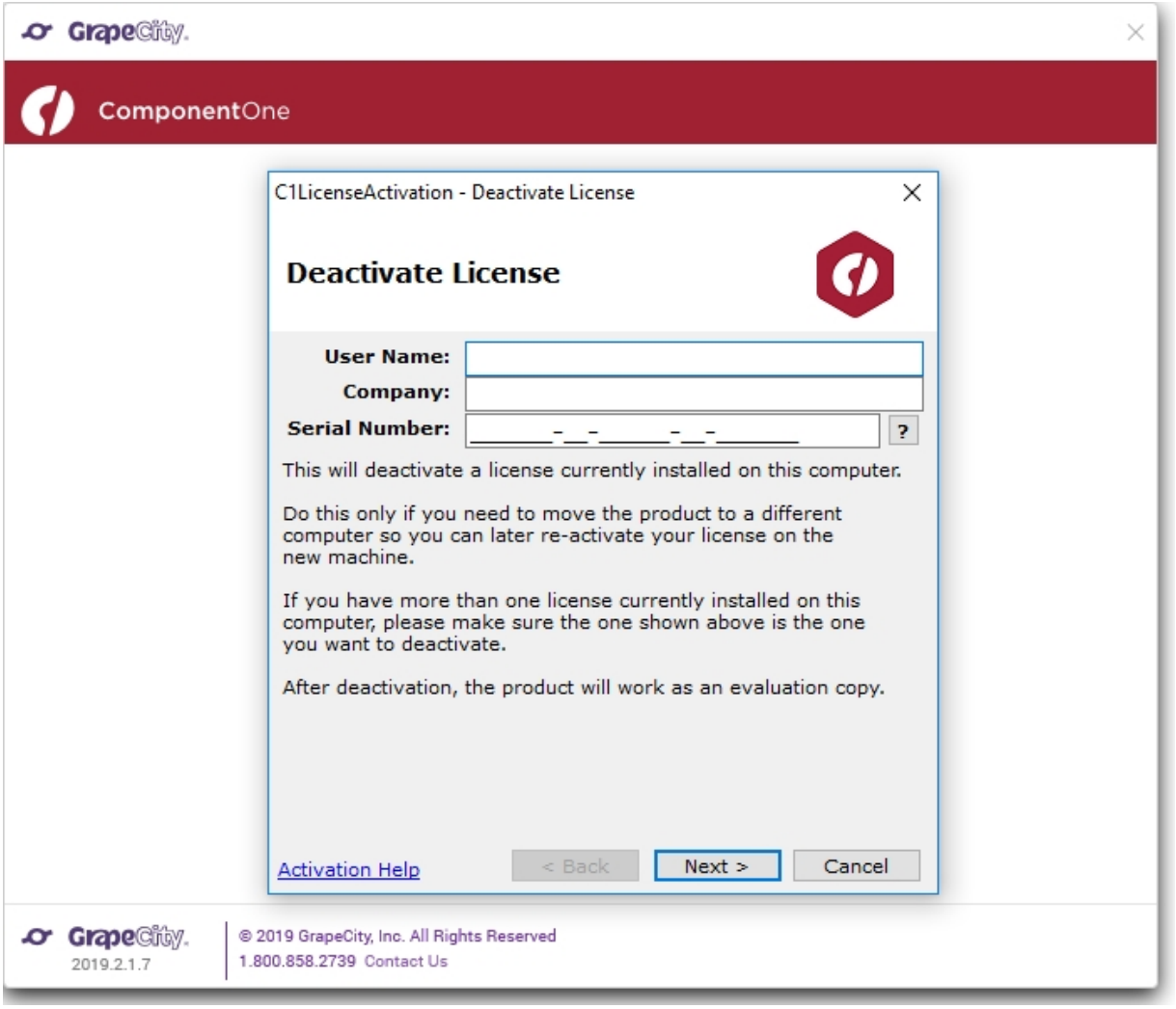


The **GrapeCity License Manager** displays the following options:

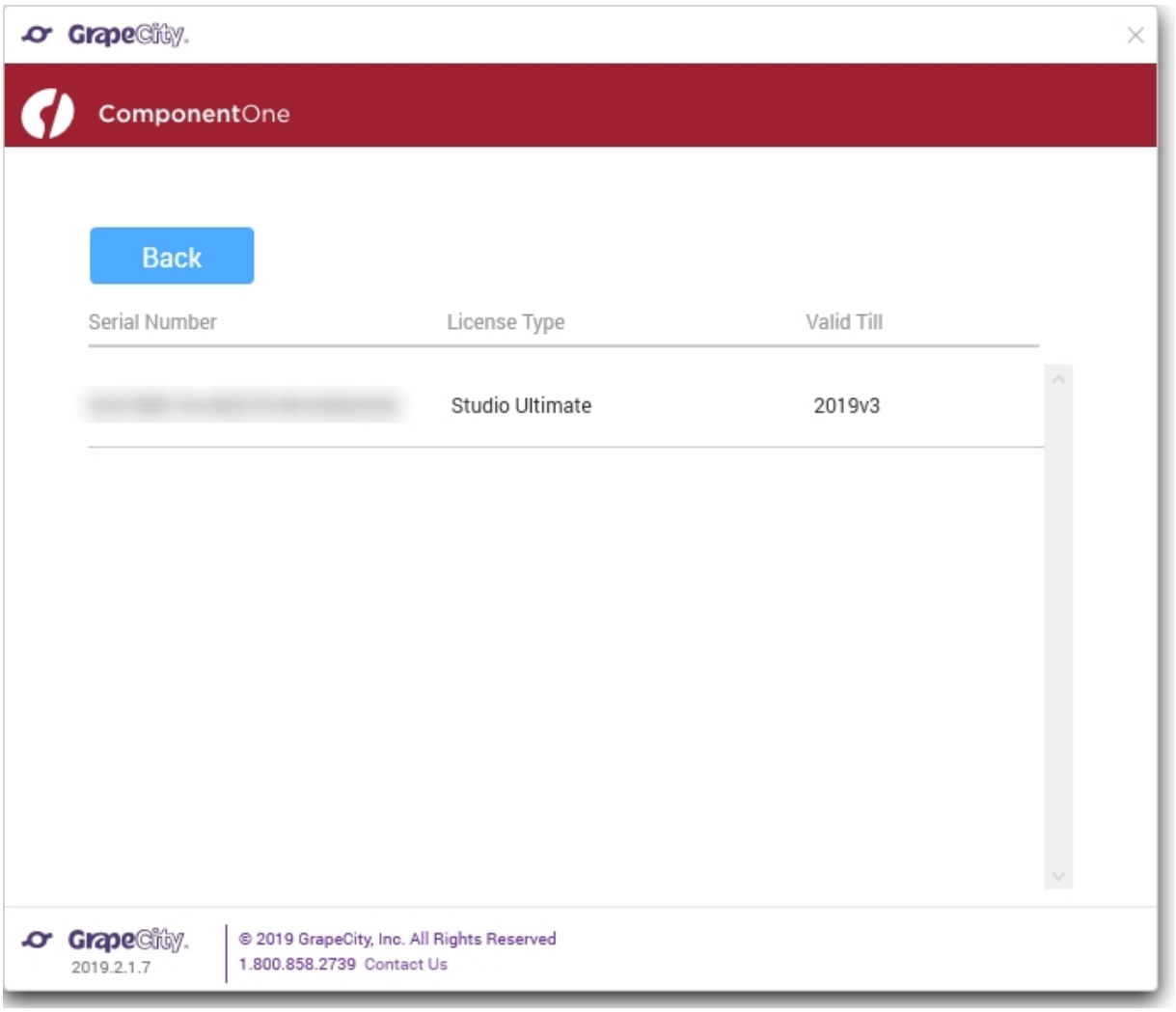
- **Activate License -** It allows the users to activate the License (**Serial Number**) using the Internet, ComponentOne website, e-mail, or over phone. On clicking this option, **C1LicenseActivation - Activate License** application window appears as shown below. Users can follow the wizard to do the license activation.



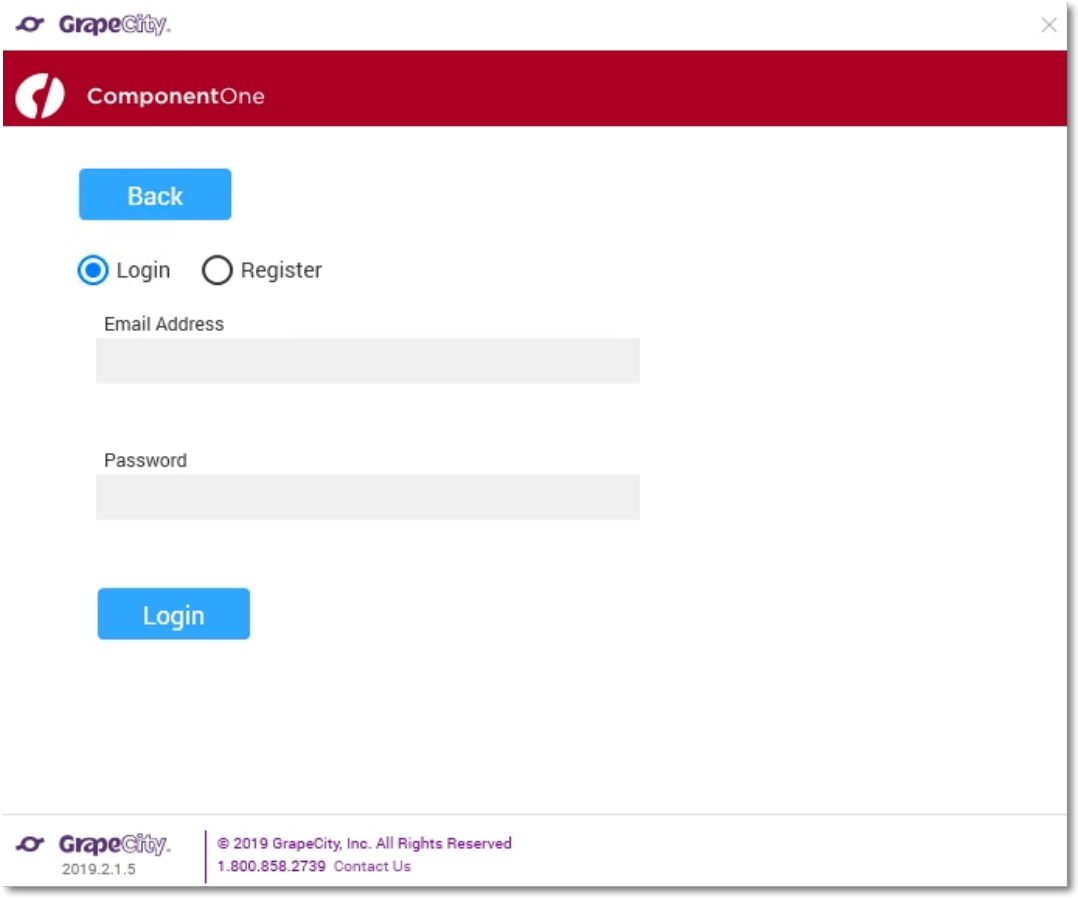
- **Deactivate License** – It allows the users to deactivate the License (**Serial Number**) using the Internet, ComponentOne website, e-mail, or over phone. On clicking this option, **C1LicenseActivation - Deactivate License** application will appear as shown below. Users can follow the wizard to do the license deactivation.



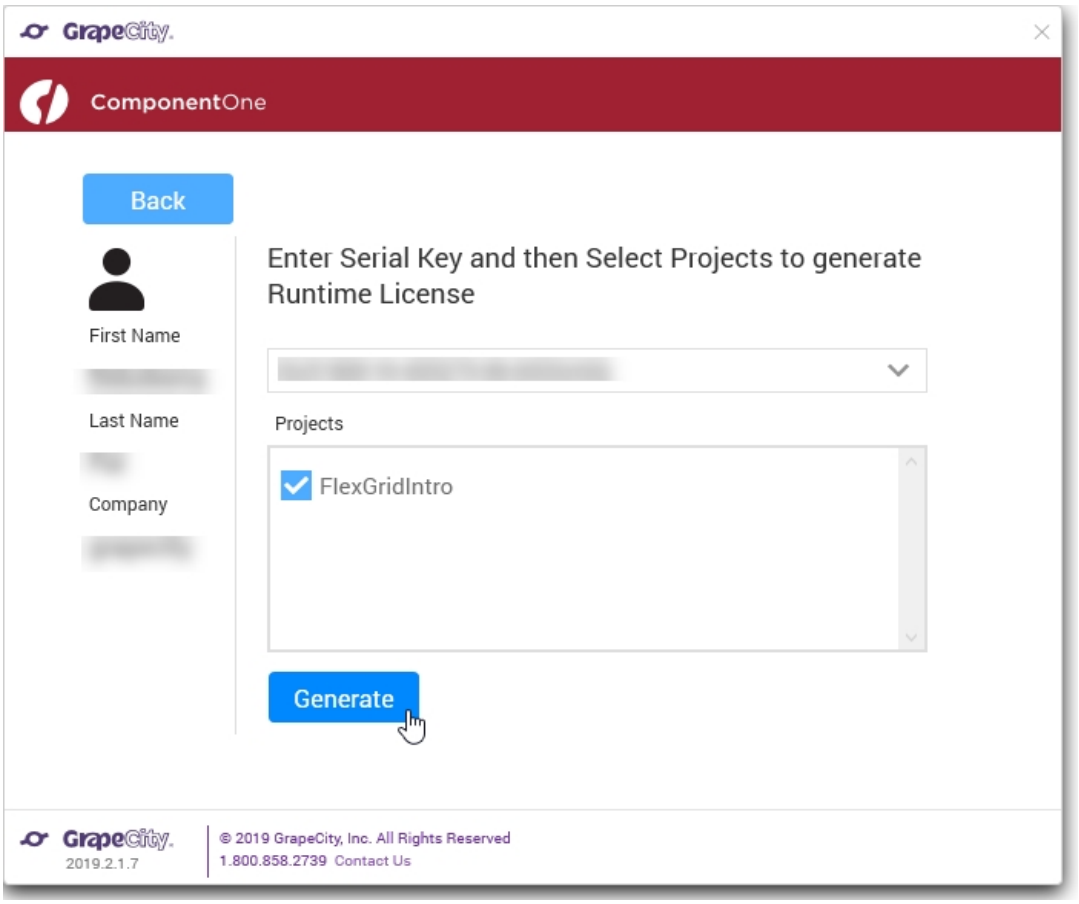
- **View Installed Licenses** - It allows the users to view the **Serial Number**, **License Type**, and **Validity** of the installed licenses on the system. The license can be viewed as shown below:



- **Create RunTime License** - It allows the users to generate runtime license for the project(s). On clicking this option, the following window appears:



To use this option, you need to **login** the **GrapeCity Account**. In case you don't have a GrapeCity Account, you can create it using the **Register** option. Once you Sign in, the following window appears:

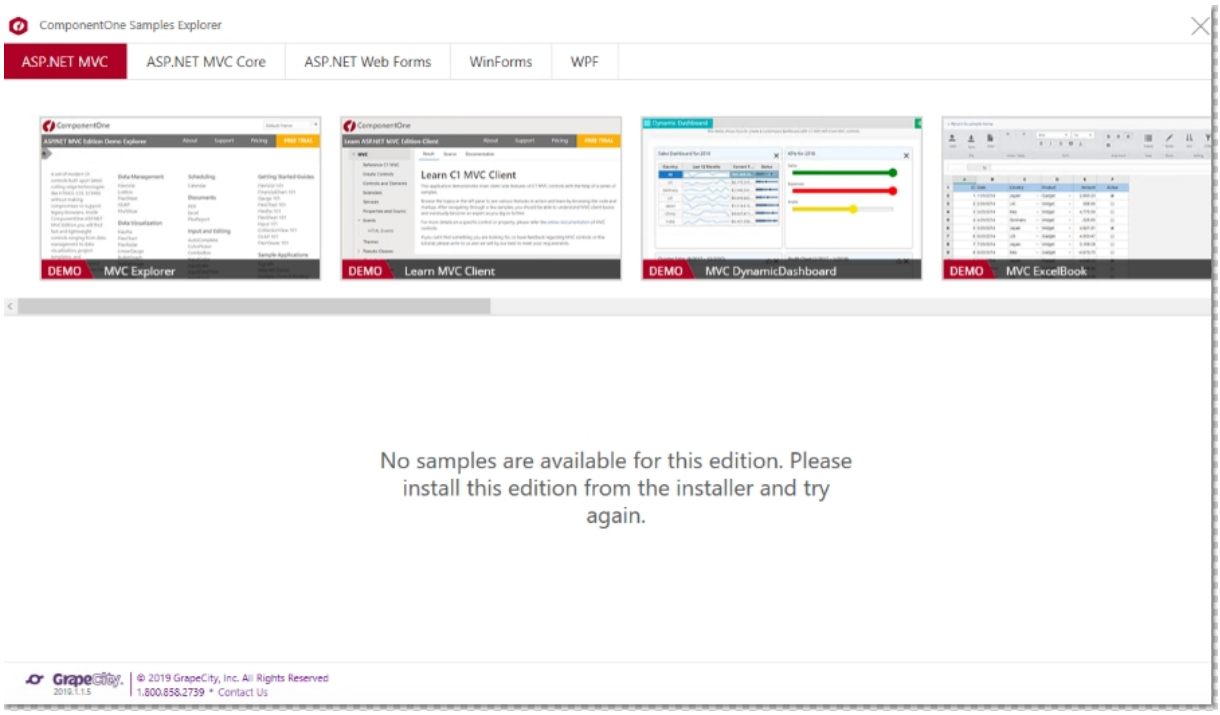


From the above window, you can select the license and the project for which license needs to be generated. On clicking the **Generate** button, a success message appears and a license file **GCDTLicenses.xml** is generated.

Samples Explorer

To view sample explorer, follow these steps:

1. From the **Tools** menu, select **GrapeCity**.
2. Select **Samples Explorer**. **ComponentOne Samples Explorer** window appears as shown in the image below:

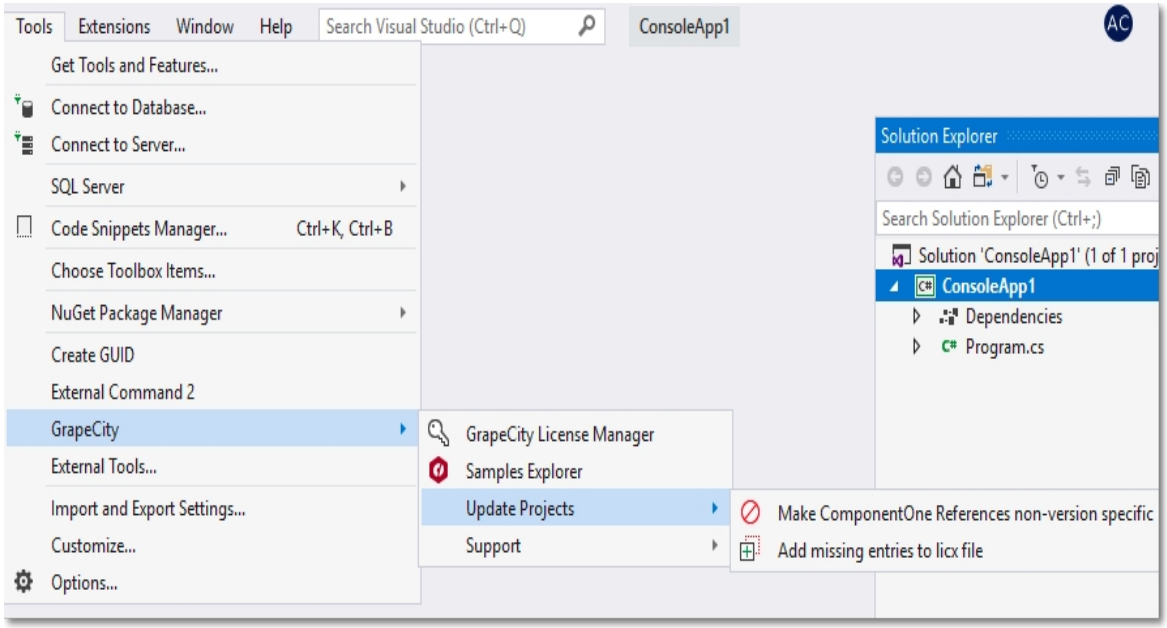


This option allows the users to open the Sample Explorer that has been installed on the system with the installer. From the displayed window, you can open any of the required samples or demos.

Update Projects

To update existing projects, follow these steps:

1. From the **Tools** menu, select **GrapeCity**.
2. Select **Update Projects**. Update Projects provides two options to update the project(s) as shown and listed below:

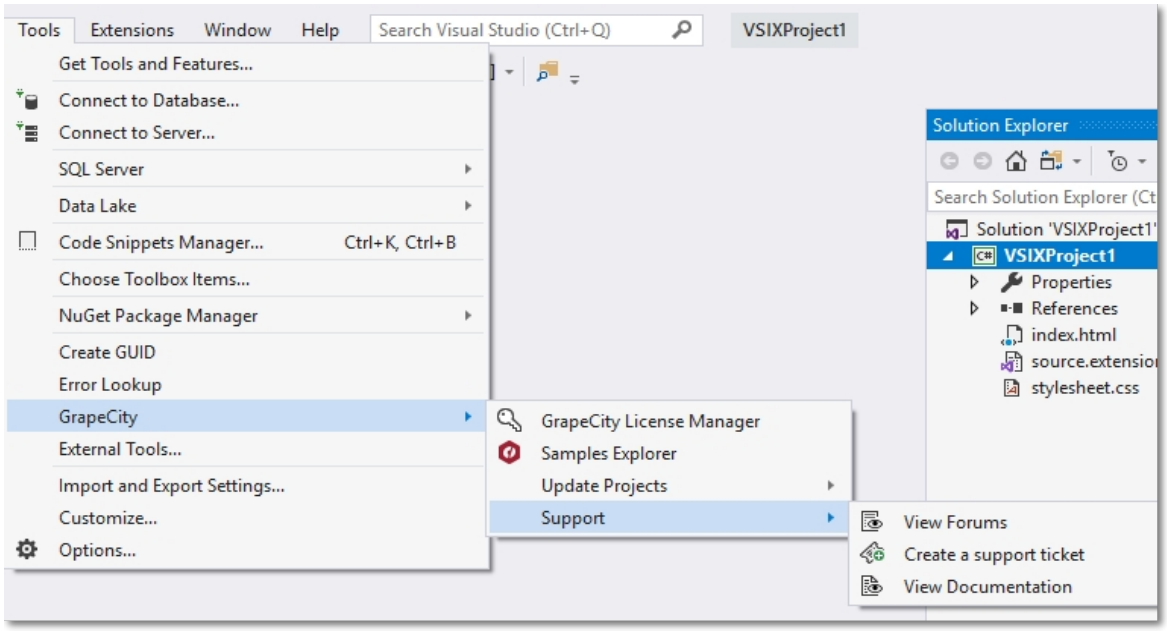


- **Make ComponentOne References non-version specific** - This option removes the version information from the Licenses.licx file and sets the **Specific Version** property of ComponentOne assemblies in the project(s) to **false**. On clicking this option, a window appears where you can select the **Project,** click **Update**, and click **Finish** to complete the update process.
- **Adding missing entries to Licx file -** This option adds missing license information to the license file (*. licx) in the project(s). On Clicking, a window appears where you can select the **Project,** click **Update**, and click **Finish** to update the license files.

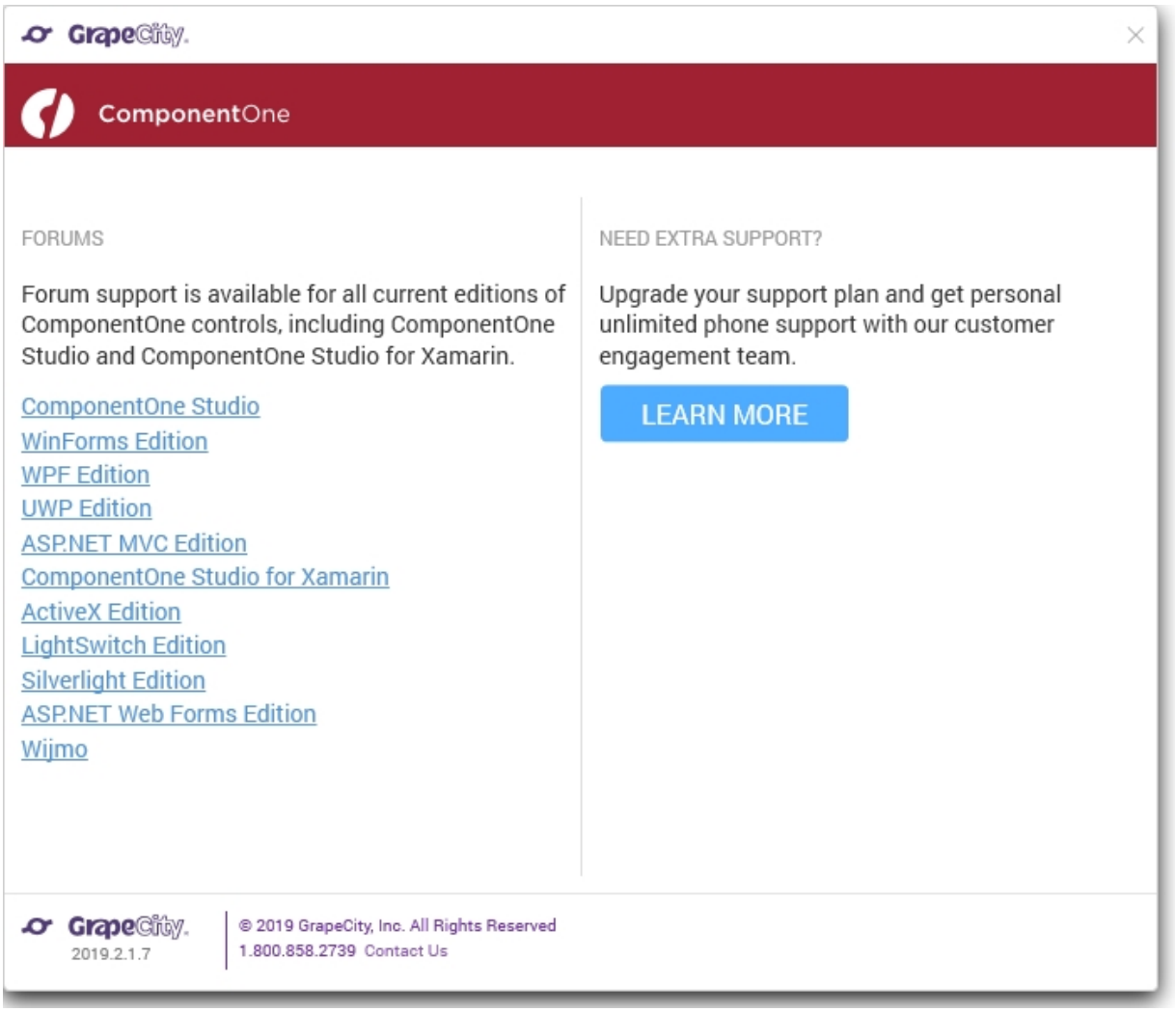
Support

To contact the Technical Support and view Documentation, follow these steps:

1. From the **Tools** menu, select **GrapeCity**.
2. Select **Support**. Support provides three options as shown and listed below.



- **View Forums** - This option opens the window shown below, which has the redirecting links to the GrapeCity ComponentOne Forums.



You can login to access the community forums for all GrapeCity products and post queries/doubts related to the products. You can also find information on how to contact the support team by clicking **Learn More**.

- **Create a support ticket -** This option redirects you to the Support Portal where you can login and get in touch with the support team directly. You can also post queries/doubts related to the products to get personalized support.
- **View Documentation -** This option redirects you to the Documentation page where you can access all the documents related to all the GrapeCity products.

Note: If you are using Visual Studio 2015 or previous versions of Visual Studio, you can use the GrapeCity License Manager add-in. The add-in is available in Tools menu in Visual Studio. For detailed information on using the GrapeCity License Manager add-in, see Generate License using GrapeCity License Manager Add-in.

Licensing your app using website

ComponentOne Xamarin Edition users can license their app via the ComponentOne website. If you are using ComponentOne Xamarin Edition with Visual Studio on PC, you have the option to use the **GrapeCity License Manager Add-in**. For more information, see Licensing your app using GrapeCity License Manager Add-in.

How to License your app using the website

1. Open a pre-existing mobile application or create a new mobile application.
2. Add the required Xamarin Edition NuGet packages to your application through the NuGet Package Manager.

3. Visit https://www.grapecity.com/en/my-account/create-app-key.

   > You must create a GrapeCity 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. Select C# for the language.
6. In the **App Name** text box, enter the name of your application. This name should match the Default Namespace of your application. See Finding the Application Name to know how to find the name of your application.
7. Click the **Generate** button. A runtime license will be generated in the form of a string contained within a class.
8. Copy the license and complete the following steps to add it in your application.
   1. Open your application in Visual Studio.
   2. In the **Solution Explorer**, right-click the project YourAppName.
   3. Select **Add | New**. The **Add New Item** dialog appears.
   4. Under installed templates, select **C# | Class**.
   5. Set the name of the class as **License.cs** and click **OK**.
   6. In the class License.cs, create a new string to store the runtime license inside the constructor as shown below.

      C#

      ```
      public static class License
      {
          public const string Key = "Your Key";
      }
      ```

   7. From the Solution Explorer, open **AppDelegate.cs** and set the runtime license inside the **FinishedLaunching()** method as shown below.

      C#

      ```
      C1.iOS.Core.LicenseManager.Key = License.Key;
      ```

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

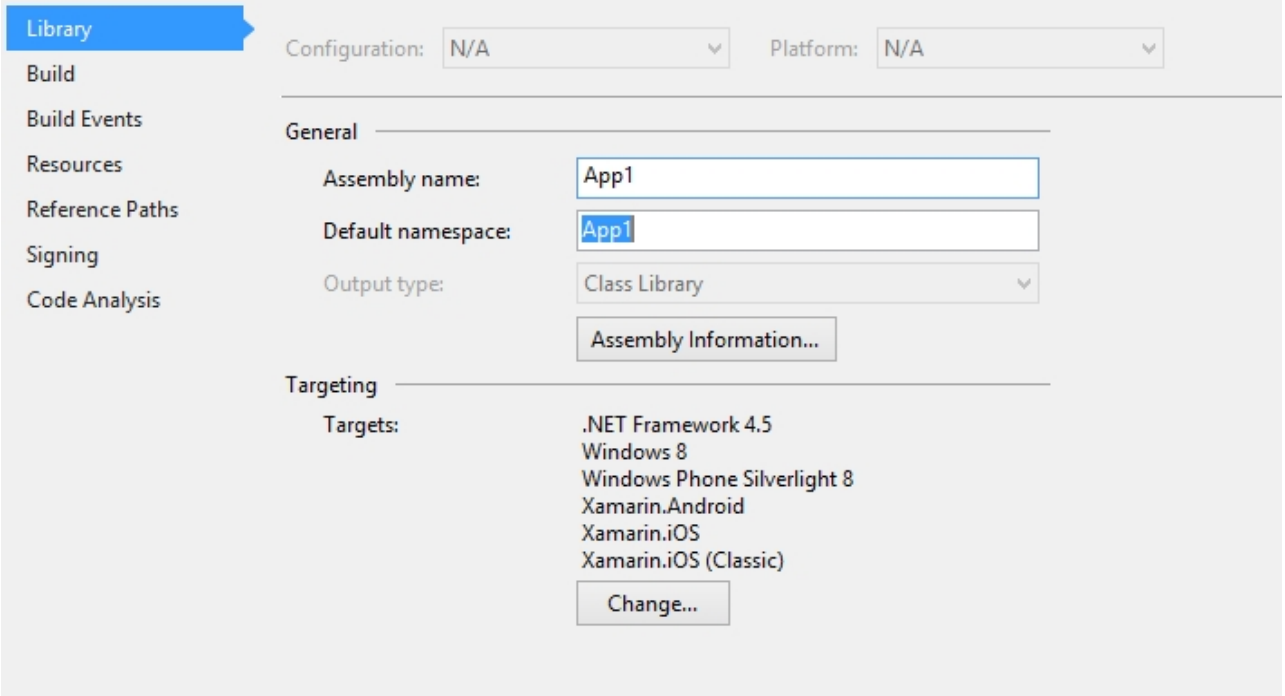
> The trial period is limited to 30 days, which begins when you generate your first runtime license. The controls will stop working after your 30-day trial period is over.

Finding the Application Name

ComponentOne Xamarin Edition licenses are unique to each application. Before you can generate a runtime license, you need to know the name of the application where the license will be used.

Visual Studio

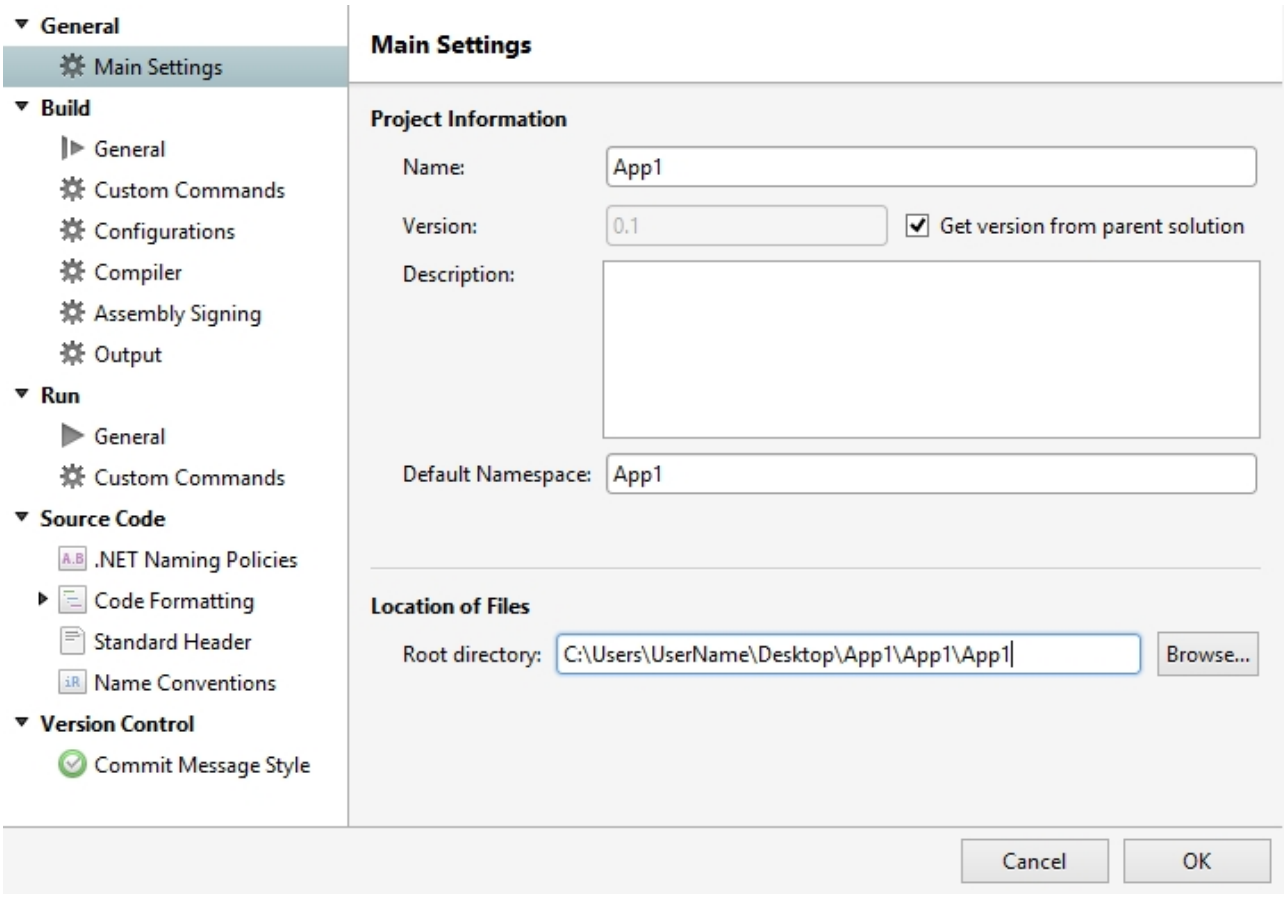
1. Open a pre-existing mobile application.
2. In the Solution Explorer, right-click the project **YourAppName** and select **Properties**.
3. Open the **Library** tab.
4. The application name is the same as the displayed **Default namespace**.



You need to generate a new runtime license in case you rename the assembly later.

Visual Studio for Mac

1. Open a pre-existing mobile application.
2. In the **Solution Explorer**, right click the project **YourAppName** and select Options.
3. The application name is displayed on the **Main Settings** tab.



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 website to explore more.

Some methods for obtaining technical support include:

- **Online Resources**

ComponentOne provides customers with a comprehensive set of technical resources in the form of Licensing FAQs, samples, demos, and videos, searchable online documentation 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 Submit a ticket. 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 us via an email within two business days.
- **Product Forums**
  Forums are available for users to share information, tips, and techniques regarding all the platforms supported by the ComponentOne Xamarin Edition, including Xamarin Forms, Xamarin.iOS and Xamarin.Android. ComponentOne developers or community engineers will be available on the forums to share insider tips and technique and answer users' questions. Note that a user account is required to participate in the Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing Xamarin Edition on their systems. 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 and PDF files can be downloaded for offline viewing. If you have suggestions on how we can improve our documentation, please send feedback to support forum.

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

Controls

Calendar

The C1Calendar control provides a calendar through which you can navigate to any date in any year. The control comes with an interactive date selection user interface (UI) with month, year and decade view modes. Users can view as well as select multiple dates on the calendar.

The C1Calendar provides the ability to customize day slots so that users can visualize date information on the calendar. In addition, you can also customize the appearance of the calendar using your own content and style.

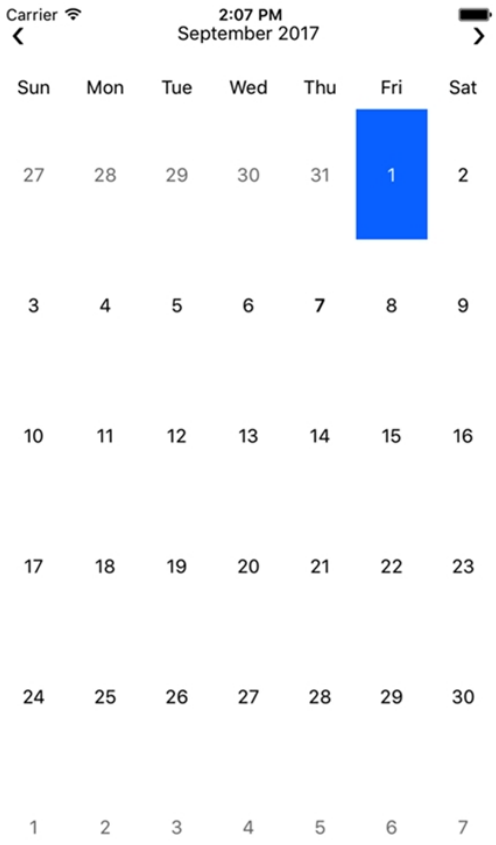
Key Features

- **Custom Day Content**: Customize the appearance of day slots by inserting custom content.
- **View Modes**: Tap header to switch from month mode to year and decade mode.
- **Appearance**: Easily style different parts of the control with heavy customizations.
- **Date Range Selection**: Simply tap two different dates to select all the dates in between.
- **Orientation**: Toggle the scroll orientation to either horizontal or vertical.
- **Animation**: By default, Calendar confers fast, animated transitions for enhanced user experience.

Quick Start: Display a C1Calendar Control

This section describes how to add a C1Calendar control to your iOS app and select a date on the calendar at runtime.

The following image shows how C1Calendar appears after completing the above steps.



Step 1: Initialize Calendar control in code

To initialize a **C1Calendar** control, open the ViewController file from the Solution Explorer and replace its code with the code below. This code overrides the **ViewDidLoad** method of the View controller in order to initialize a C1Calendar.

C#

```
using C1.iOS.Calendar;
using CoreGraphics;
using System;
using UIKit;

namespace CalendariOS
{
    public partial class ViewController : UIViewController
    {
        //Initialize Calendar controlS
        C1Calendar calendar = new C1Calendar();
        public ViewController(IntPtr handle) : base(handle)
        {
```

```
        }
        public override void ViewDidLoad()
        {
           base.ViewDidLoad();
           this.Add(calendar);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            calendar.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
             this.View.Frame.Width, this.View.Frame.Height);
        }
    }
}
```

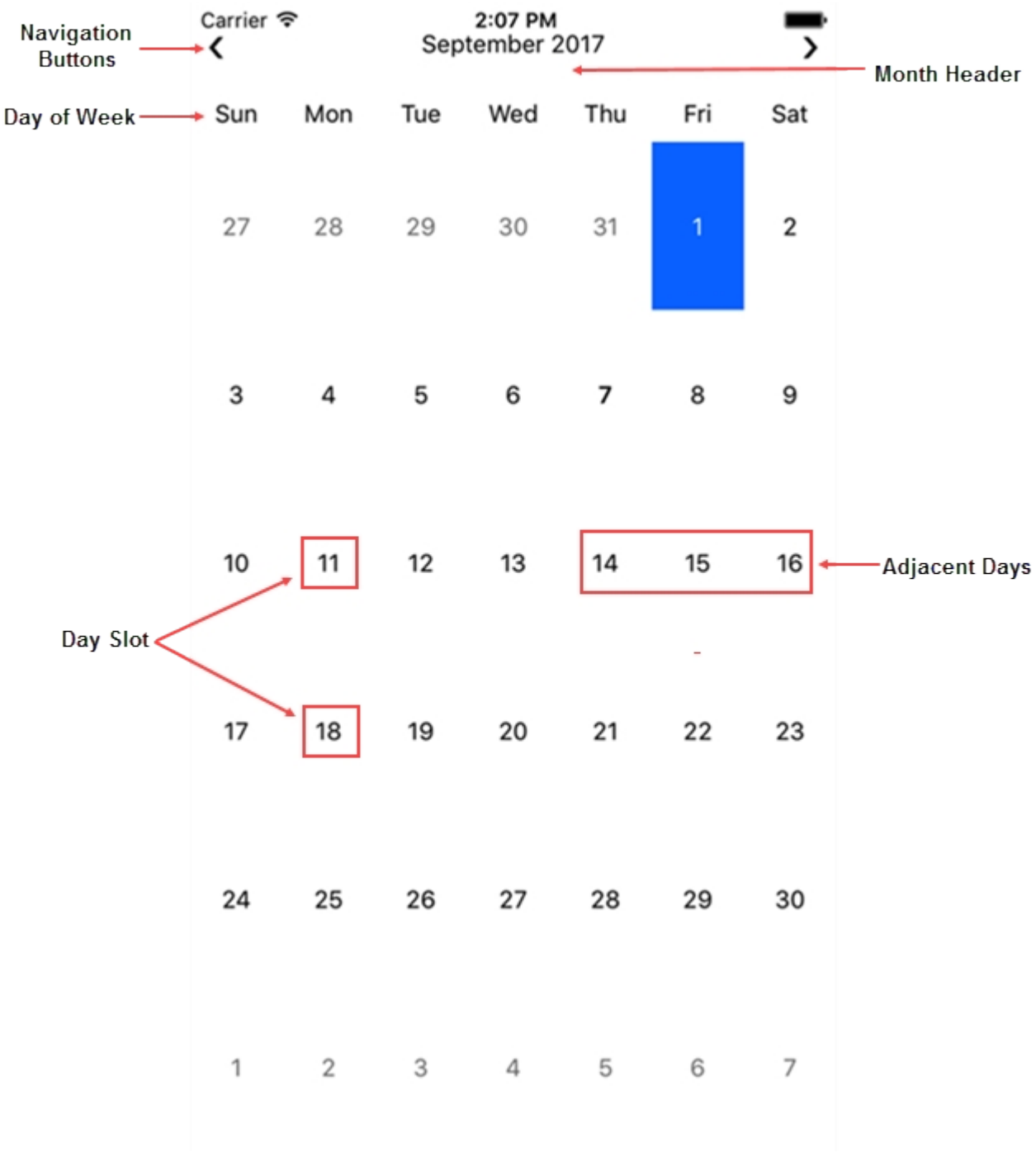
Step 2: Run the Application

Press **F5** to run the application.

Interaction Guide

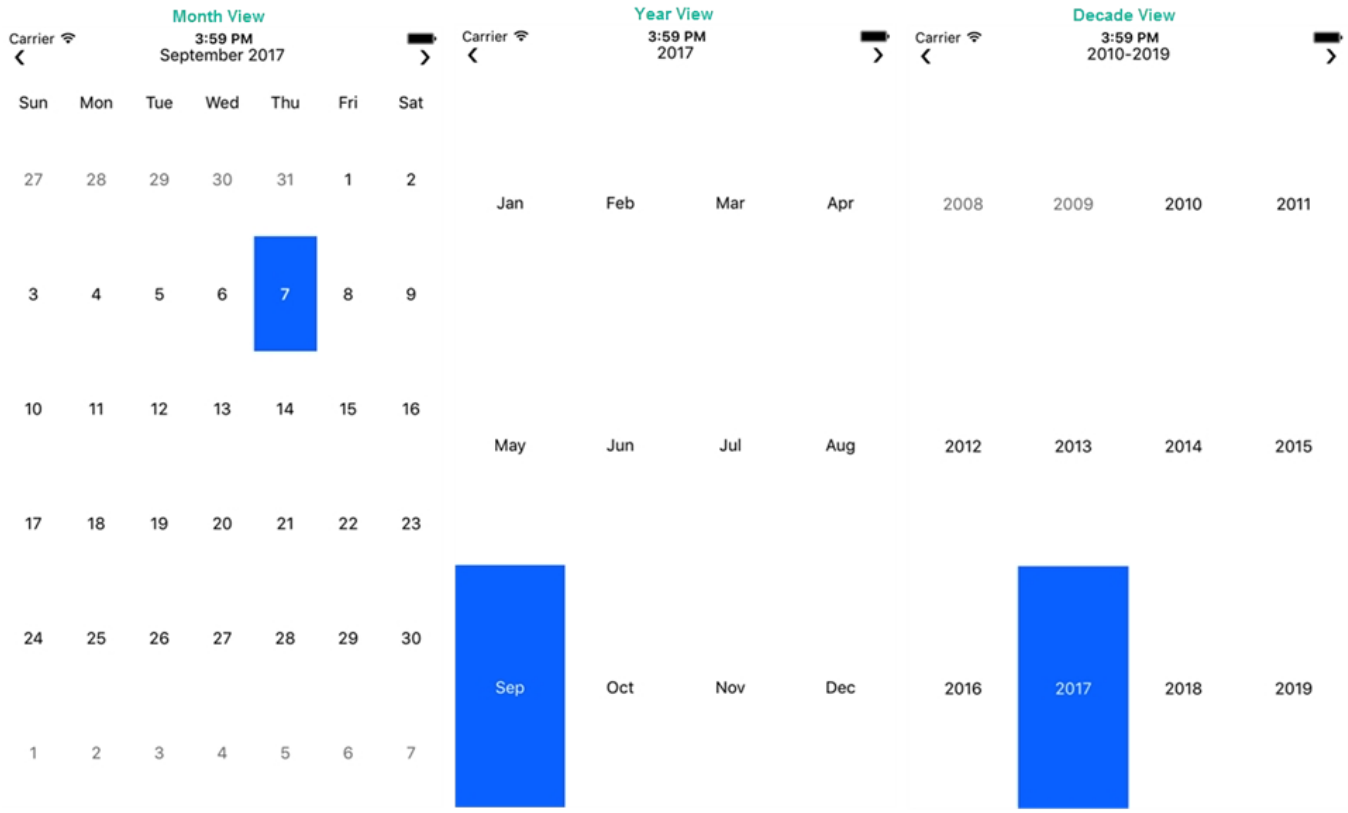
The C1Calendar control comprises of various functional parts such as Header, Day of Week, Day Slot, etc. These functional parts are illustrated below.

- **Header** - The C1 Calendar comprises a header that displays the current month, year or decade along with navigation buttons. Users can hide the default header and create their own headers as illustrated in Customizing Header.
- **Day Slot** - The C1 Calendar features day slots which can be customized to display custom content. See Customizing Day Content to understand how day slots can be used to insert and display custom content.
- **Day of Week -** The C1 Calendar's user interface displays seven days of week corresponding to respective dates.
- **Navigation Buttons -** The navigation buttons enables users to traverse the selected month or year, forward or backward.



View Modes

The C1Calendar for iOS supports month, year and decade views as shown in the image below. The calendar switches from month to year view when the user taps the month header. On tapping the year header, the calendar switches to decade view. The calendar switches back to the month view on tapping the header on the decade view, completing a full circle from Month<Year<Decade<Month view.

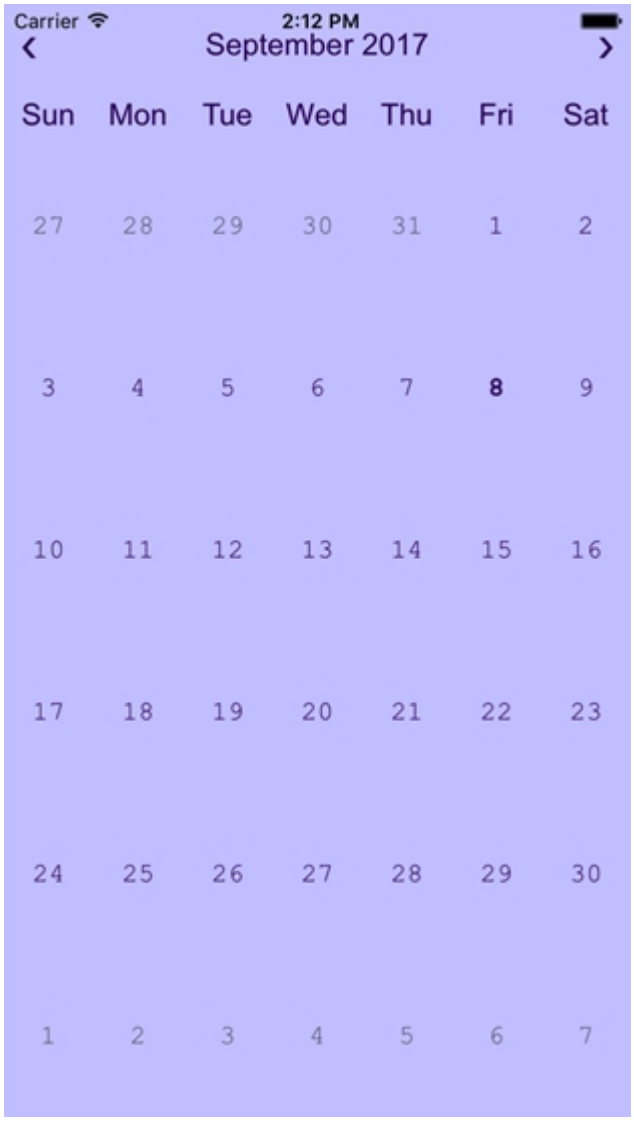


Features

Customizing Appearance

C1Calendar provides various built-in properties to customize calendar's appearance. You can use these properties to set calendar's background color, text color, header color, font size, header font size, selection background color, etc.

The image below shows a customized C1Calendar after setting these properties.



The following code example demonstrates how to set these properties in C#. This example uses the sample created in the Quick Start.

In Code

C#

```
using C1.iOS.Calendar;
using CoreGraphics;
using System;
using UIKit;

namespace CalendariOS
{
    public partial class ViewController : UIViewController
    {
        C1Calendar calendar = new C1Calendar();
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        public override void ViewDidLoad()
```

```
        {
            base.ViewDidLoad();
             calendar.ViewModeAnimation.ScaleFactor = 1.1;
             calendar.ViewModeAnimation.AnimationMode =
CalendarViewModeAnimationMode.ZoomOutIn;
             calendar.Font = UIFont.FromName("Courier New", 16);
             calendar.TodayFont = UIFont.FromDescriptor(UIFont.FromName("Courier New",
16).FontDescriptor.CreateWithTraits(UIFontDescriptorSymbolicTraits.Bold), 16);
             calendar.DayOfWeekFont = UIFont.FromName("Arial", 18);
             calendar.HeaderFont = UIFont.FromName("Arial", 18);
             calendar.TextColor = UIColor.FromRGB(51, 0, 102);
             calendar.BackgroundColor = UIColor.FromRGB(204, 204, 255);
             calendar.BorderColor = UIColor.Black;
             this.Add(calendar);
        }
        public override void ViewDidLayoutSubviews()
        {
             base.ViewDidLayoutSubviews();
             calendar.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
              this.View.Frame.Width, this.View.Frame.Height);
        }
    }
}
```

Customizing Header

The C1Calendar control shows a default header that displays the current month or year and navigation buttons. However, users can hide or remove the default header by setting the showHeader property to No, and apply a custom header.

The following image shows a C1Calendar with a custom header.

The following code example shows how to apply a custom header to the C1 control in C#. The example uses the sample created in the Quick Start.

1. Replace the content of the **ViewController.cs** file with the code given below.

C#

```
using C1.iOS.Calendar;
using CoreGraphics;
using Foundation;
using System;
using UIKit;

namespace CalendariOS
{
    public partial class ViewController : UIViewController
    {
        C1Calendar Calendar = new C1Calendar();
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();

            var model = new ViewModePickerViewModel(async row =>
            {
                switch (row)
```

```
                {
                    case 0:
                        await
Calendar.ChangeViewModeAsync(CalendarViewMode.Month);
                        break;
                    case 1:
                        await
Calendar.ChangeViewModeAsync(CalendarViewMode.Year);
                        break;
                    case 2:
                        await
Calendar.ChangeViewModeAsync(CalendarViewMode.Decade);
                        break;
                }
            });
            ViewModePicker.Model = model;
            Calendar.ViewModeChanged += OnViewModeChanged;
            Calendar.DisplayDateChanged += OnDisplayDateChanged;
            UpdateHeaderLabel();
        }

        async partial void TodayButton_TouchUpInside(UIButton sender)
        {
            await Calendar.ChangeViewModeAsync(CalendarViewMode.Month,
DateTime.Today);
            Calendar.SelectedDate = DateTime.Today;
        }

        private void OnViewModeChanged(object sender, EventArgs e)
        {
            switch (Calendar.ViewMode)
            {
                case CalendarViewMode.Month:
                    ViewModePicker.Select(0, 0, true);
                    break;
                case CalendarViewMode.Year:
                    ViewModePicker.Select(1, 0, true);
                    break;
                case CalendarViewMode.Decade:
                    ViewModePicker.Select(2, 0, true);
                    break;
            }
        }

        private void OnDisplayDateChanged(object sender, EventArgs e)
        {
            UpdateHeaderLabel();
        }

        private void UpdateHeaderLabel()
        {
```

```
            HeaderLabel.Text = Calendar.DisplayDate.ToString("Y");
        }

        public class ViewModePickerViewModel : UIPickerViewModel
        {
            Action<nint> _selected;
            public ViewModePickerViewModel(Action<nint> selected)
            {
                _selected = selected;
            }

            public override nint GetComponentCount(UIPickerView picker)
            {
                return 1;
            }

            public override nint GetRowsInComponent(UIPickerView picker, nint
component)
            {
                return 3;
            }

            public override string GetTitle(UIPickerView picker, nint row, nint
component)
            {
                switch (row)
                {
                    default:
                    case 0:
                        return NSBundle.MainBundle.LocalizedString("Month View",
"");
                    case 1:
                        return NSBundle.MainBundle.LocalizedString("Year View",
"");
                    case 2:
                        return NSBundle.MainBundle.LocalizedString("Decade
View", "");
                }
            }

            public override void Selected(UIPickerView pickerView, nint row,
nint component)
            {
                _selected(row);
            }
        }
    }
}
```

Customizing Day Content

The C1Calendar control allows users to add custom content to day slot. For this, all you need to do is subscribe the **DaySlotLoading** event and apply custom content such as images in the background of these day slots. This feature allows users to display weather related information on the calendar.

The image below shows a C1Calendar after adding custom content to day slots. The calendar displays weather related information through various icons.

The following code example shows how to add custom content to day slots to the C1Calendar control in C#. The example uses the sample created in the Quick Start.

1. Replace the content of the ViewController.m file with the code given below.

Objective-C

```
using Foundation;
using System;
using UIKit;
using C1.iOS.Calendar;


namespace C1Calendar101
{
    public partial class CustomDayController : UIViewController
    {
        public CustomDayController(IntPtr handle) : base(handle)
        {
        }

        public override void LoadView()
```

```
        {
            base.LoadView();
            Calendar.DaySlotLoading += OnDaySlotLoading;
            Calendar.Refresh();
        }

        public override void WillAnimateRotation(UIInterfaceOrientation
toInterfaceOrientation, double duration)
        {
            base.WillAnimateRotation(toInterfaceOrientation, duration);
            Calendar.Refresh();
        }

        private void OnDaySlotLoading(object sender,
CalendarDaySlotLoadingEventArgs e)
        {
            // add weather image for certain days in the current month
            var today = DateTime.Today;
            if (e.Date.Year == today.Year &&
                e.Date.Month == today.Month &&
                e.Date.Day >= 14 && e.Date.Day <= 23)
            {
                var iv = new UIImageView();
                var tv = new UILabel();
                var slot = new UIStackView(new UIView[] { tv, iv });
                slot.Axis = InterfaceOrientation ==
UIInterfaceOrientation.Portrait || InterfaceOrientation ==
UIInterfaceOrientation.PortraitUpsideDown ? UILayoutConstraintAxis.Vertical :
UILayoutConstraintAxis.Horizontal;

                tv.Text = e.Date.Day.ToString();

                UIImage image;
                switch (e.Date.Day % 5)
                {
                    default:
                    case 0:
                        image = UIImage.FromFile(@"Images/Cloudy.png");
                        break;
                    case 1:
                        image = UIImage.FromFile(@"Images/PartlyCloudy.png");
                        break;
                    case 2:
                        image = UIImage.FromFile(@"Images/Rain.png");
                        break;
                    case 3:
                        image = UIImage.FromFile(@"Images/Storm.png");
                        break;
                    case 4:
                        image = UIImage.FromFile(@"Images/Sun.png");
                        break;
```

```
                }
                image =
image.ImageWithRenderingMode(UIImageRenderingMode.AlwaysTemplate);
                iv.UserInteractionEnabled = true;
                iv.TintColor = UIColor.Black;
                iv.ContentMode = UIViewContentMode.ScaleAspectFit;
                iv.Image = image;
                e.DaySlot = slot;
            }
        }
    }
}
```

Orientation

The C1Calendar appears in default horizontal orientation. However, you can change the orientation of the calendar to Vertical by using the Orientation property. The orientation property can be set using CalendarOrientation enumeration in code so that the calendar's orientation changes to Vertical.

The following code example demonstrates how to set the Orientation in C#. This code example uses the sample created in the Quick Start.

In Code

C#

```
Calendar.Orientation = CalendarOrientation.Vertical;
```

Custom Selection

You can customize the default behavior of the C1Calendar control to select specific dates. For instance, consider a scenario where you wish to select only the weekdays on tapping two dates in different workweeks. For this, you simply need to subscribe the SelectionChanging event and apply selection condition in the handler.

The following image shows a C1Calendar that only selects weekdays and deselect weekends on tapping two different dates in different workweeks.

The following code example shows how to customize selection in C1Calendar control in C#. The example uses the sample created in the Quick Start.

1. Replace the content of the ViewController.cs file with the code given below.

C#

```
using C1.iOS.Calendar;
using System;
using UIKit;

namespace CalendariOS
{
    public partial class ViewController : UIViewController
    {
        public ViewController(IntPtr handle) : base(handle)
        {
        }

        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            Calendar.SelectionChanging += OnSelectionChanging;
        }
        public override void ViewDidUnload()
        {
            base.ViewDidUnload();
            Calendar.SelectionChanging -= OnSelectionChanging;
```

```
        }

        private void OnSelectionChanging(object sender,
CalendarSelectionChangingEventArgs e)
        {
            foreach (var date in e.SelectedDates.ToArray())
            {
                if (date.DayOfWeek == DayOfWeek.Saturday || date.DayOfWeek ==
DayOfWeek.Sunday)
                    e.SelectedDates.Remove(date);
            }
        }

    }
}
```

CollectionView

CollectionView is a powerful cross-platform data management component that is designed to perform common data transformations and virtualization operations like sorting, grouping, filtering, editing, incremental-loading and refreshing. It provides a series of interfaces which are consumed and manipulated by the controls, such as grid, and these interfaces are implemented by collection views. CollectionView provides the interfaces so that the built-in features of the control are available for you to provide the data in a straightforward way without needing to synchronize the sources.

C1CollectionView library provides two implementations of ICollectionView, namely C1CollectionView and C1CursorCollectionView. **C1CollectionView** is an in-memory collection view and **C1CursorCollectionView** is an abstract base class for a range of remote-source collections.

Key Features

- Provides filtering, grouping and sorting on a data set.
- Can be used with the data collection controls, such as grid.
- Provides currency for master-detail support for X.iOS apps.
- Based on the .NET implementation of ICollectionView.
- Performs on demand incremental loading and refreshing of data using C1CursorCollectionView.

The **C1CollectionView** implements several interfaces to provide current record management, custom sorting, filtering, grouping, editing, incremental loading, and refreshing.

Quick Start

This section describes how to add a **FlexGrid** control to your Xamarin.iOS application and add data to it using the CollectionView class. For more information on how to create a new Xamarin.iOS application, see Creating a New Xamarin.iOS App.

This topic comprises of three steps:

- **Step 1: Create a data source for FlexGrid**
- **Step 2: Add a FlexGrid control**
- **Step 3: Run the Application**

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

	Id	Country	Amount	Active
	0	Germany	294.2	☑
	1	Greece	344.64	☐
	2	Italy	402.72	☐
	3	Japan	637.5	☐
	4	UK	227.77	☑
	5	US	509.72	☐
	6	Germany	303.56	☐
	7	Greece	974.86	☐
	8	Italy	553.39	☑
	9	Japan	423.03	☐

Step 1: Create a data source for FlexGrid

1. Add a new class file to the Xamarin.iOS application (Name: `Customer.cs`).
2. Add the following code to the `Customer.cs` file. We are using **Customer** class to represent data in the FlexGrid control.

Customer.cs

```
public class Customer
    {
        int _id, _countryID;
        string _first, _last, _address, _postalCode, _email;
        static Random _rnd = new Random();
        static string[] _firstNames =
"Gil|Oprah|Xavier|Herb|Charlie|Larry|Steve".Split('|');
        static string[] _lastNames =
"Orsted|Frommer|Jammers|Krause|Neiman".Split('|');
        static string[] _countries = "Brazil|Colombia|Egypt|United
States|Japan|Thailand".Split('|');

        static string[] _emailServers = "gmail|yahoo|outlook|aol".Split('|');
        static string[] _streetNames =
"Main|Broad|Grand|Panoramic|Green|Golden|Park|Fake".Split('|');
        static string[] _streetTypes = "ST|AVE|BLVD".Split('|');
        static string[] _streetOrientation = "S|N|W|E|SE|SW|NE|NW".Split('|');

        public Customer()
            : this(_rnd.Next(10000))
        {
        }
        public Customer(int id)
        {
            ID = id;
            First = GetString(_firstNames);
            Last = GetString(_lastNames);
```

```
            Address = GetRandomAddress();
            PostalCode = _rnd.Next(100000, 999999).ToString();
            CountryID = _rnd.Next() % _countries.Length;
            Email = string.Format("{0}@{1}.com", (First + Last.Substring(0,
_rnd.Next(1, Last.Length)).ToLower()), GetString(_emailServers));
        }
        public int ID
        {
            get { return _id; }
            set
            {
                if (value != _id)
                {
                    _id = value;
                }
            }
        }
        public string Name
        {
            get { return string.Format("{0} {1}", First, Last); }
        }

        public string Address
        {
            get { return _address; }
            set
            {
                if (value != _address)
                {
                    _address = value;
                }
            }
        }
        public string Country
        {
            get { return _countries[_countryID]; }
            set { _countries[_countryID] = value; }
        }

        public int CountryID
        {
            get { return _countryID; }
            set
            {
                if (value != _countryID)
                {
                    _countryID = value;
                }
            }
        }
        public string PostalCode
        {
            get { return _postalCode; }
            set
```

```
        {
            if (_postalCode != value)
            {
                _postalCode = value;
            }
        }
    }
    public string First
    {
        get { return _first; }
        set
        {
            {
                _first = value;
            }
        }
    }

    public string Last
    {
        get { return _last; }
        set
        {
            if (_last != value)
            {
                _last = value;
            }
        }
    }
    public string Email
    {
        get { return _email; }
        set
        {
            {
                _email = value;
            }
        }
    }

    static string GetString(string[] arr)
    {
        return arr[_rnd.Next(arr.Length)];
    }
    // Provide static list.
    public static ObservableCollection<Customer> GetCustomerList(int count)
    {
        var list = new ObservableCollection<Customer>();
        for (int i = 0; i < count; i++)
        {
            list.Add(new Customer(i));
        }
        return list;
    }
    private static string GetRandomAddress()
```

```
        {
            if (_rnd.NextDouble() > 0.9)
            {
                return string.Format("{0} {1} {2} {3}", _rnd.Next(1, 999),
GetString(_streetNames), GetString(_streetTypes), GetString(_streetOrientation));
            }
            else
            {
                return string.Format("{0} {1} {2}", _rnd.Next(1, 999),
GetString(_streetNames), GetString(_streetTypes));
            }
        }
    }
```

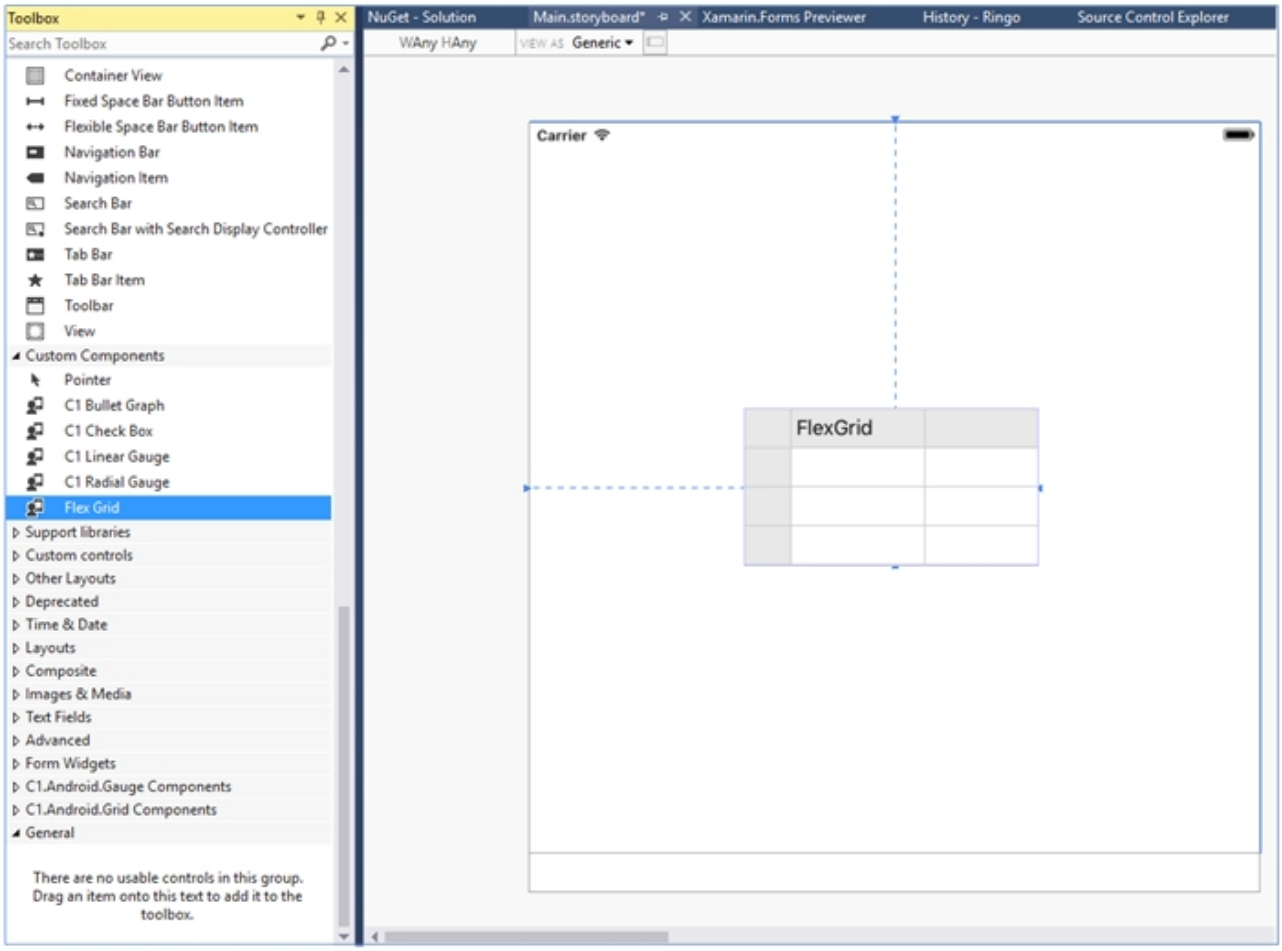
Back to Top

Step 2: Add a FlexGrid control

Complete the following steps to initialize a FlexGrid control in C#.

Add a FlexGrid control in StoryBoard

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. From the Toolbox under the **Custom Components** tab, drag the FlexGrid onto the ViewController.



Initialize FlexGrid control in code

To initialize FlexGrid control, open the ViewController file from the **Solution Explorer** and replace its content with the code below. This overrides the **ViewDidLoad** method of the View controller in order to initialize FlexGrid.

C#

```
public class QuickStart : UIViewController
    {
        public QuickStart()
        {
        }
        public override void DidReceiveMemoryWarning()
        {
            base.DidReceiveMemoryWarning();
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();

            var data = Customer.GetCustomerList(100);
            _collectionView = new C1SortCollectionView<Customer>(data);

            _grid.ItemsSource = _collectionView;
            _grid.SelectionMode = GridSelectionMode.Cell;

            var _navbarHeight = this.NavigationController.NavigationBar.Frame.Height +
UIApplication.SharedApplication.StatusBarFrame.Height;
            _grid.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y+_navbarHeight,
this.View.Frame.Width, this.View.Frame.Height-_navbarHeight);
            this.View.AddSubview(_grid);
        }
        private C1SortCollectionView<Customer> _collectionView;
    }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

Filtering

Filtering refers to refining data sets into simply what a user needs, without including other data that can be repetitive or irrelevant. The ISupportFiltering interface enables support for filtering in data controls, such as grids. It implements the FilterAsync method so that you can call the filtering operation in the CollectionView without having to cast to the specific interface. The **FilterAsync** method filters the data according to the filter value provided with the help of filter button.

The following image shows filtering in the FlexGrid control using CollectionView class.

In Code

The following code examples shows how to implement filtering in the FlexGrid control.

Filtering.cs

```
C#
```

```
using Foundation;
using System;
using UIKit;
using C1.iOS.Grid;
using C1.CollectionView;
namespace CollectionView.iOS
{
    public partial class ViewController : UIViewController
    {
        public FlexGrid Grid;
        public FullTextFilterBehavior fullTextFilter;
        public UITextField Filter;
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            Grid = new FlexGrid();
            Filter = new UITextField();
            Filter.BorderStyle = UITextBorderStyle.RoundedRect;
            Filter.BackgroundColor = UIColor.LightGray;
            Title = "Filtering";

            var data = Customer.GetCustomerList(100);
            Grid.ItemsSource = data;
            //Filter FlexGrid's CollectionView based on Jammers string
            ((C1CollectionView<object>)Grid.CollectionView).FilterAsync("Jammers");
            fullTextFilter = new FullTextFilterBehavior();
            fullTextFilter.HighlightColor = UIColor.Blue;
            //attach UITextField filter for live user entry filtering
            fullTextFilter.Attach(Grid);
            fullTextFilter.FilterEntry = Filter;

            this.View.AddSubview(Grid);
            this.View.AddSubview(Filter);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            Filter.Frame = new CoreGraphics.CGRect(10, 50, this.View.Frame.Size.Width -
20, 30);
            Grid.Frame = new CoreGraphics.CGRect(0, 95, this.View.Frame.Size.Width,
this.View.Frame.Size.Height - 55);
        }
    }
}
```

Grouping

Grouping refers to combining rows based on column values. When grouping is applied to a grid, it automatically sorts

the data, splits it into groups, and adds collapsible group rows above or below each group. Similarly, the ISupportGrouping interface enables support for grouping within data controls, such as grids. It implements GroupAsync method so that you can call the grouping operation in the CollectionView without having to cast to the specific interface. The **GroupAsync** method groups the CollectionView according to the specified group fields, group path, or group descriptions.

The image below shows how the FlexGrid appears, after the grouping is applied to column **Country**.

The following code examples demonstrate how to set this property in C#. These examples use the sample created in the Quick Start section.

In Code

C#

```
using UIKit;
using Foundation;
using C1.iOS.Grid;
using CoreGraphics;
using C1.CollectionView;
namespace CollectionView.iOS
{
    public class Grouping : UIViewController
    {
        public Grouping()
        {
        }
        public override void DidReceiveMemoryWarning()
        {
            // Releases the view if it doesn't have a superview.
            base.DidReceiveMemoryWarning();

            // Release any cached data, images, etc that aren't in use.
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();

            // Perform any additional setup after loading the view
            View.BackgroundColor = UIColor.White;
            Title = "Grouping";
            _grid = new FlexGrid();
            var data = Customer.GetCustomerList(100);
            _collectionView = new C1CollectionView<Customer>(data);
            _collectionView.GroupAsync("Country");

            _grid.ItemsSource = _collectionView;

            _grid.AllowDragging = GridAllowDragging.Both;
            _grid.AllowResizing = GridAllowResizing.Both;
            _grid.SelectionMode = GridSelectionMode.Cell;
            _grid.RowHeaders.Columns.Clear();

            var _navbarHeight = this.NavigationController.NavigationBar.Bounds.Height
+ UIApplication.SharedApplication.StatusBarFrame.Height;

            _grid.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y +
_navbarHeight, this.View.Frame.Width, this.View.Frame.Height - _navbarHeight);
            this.View.AddSubview(_grid);
        }
        private FlexGrid _grid;
        private C1CollectionView<Customer> _collectionView;
    }
}
```

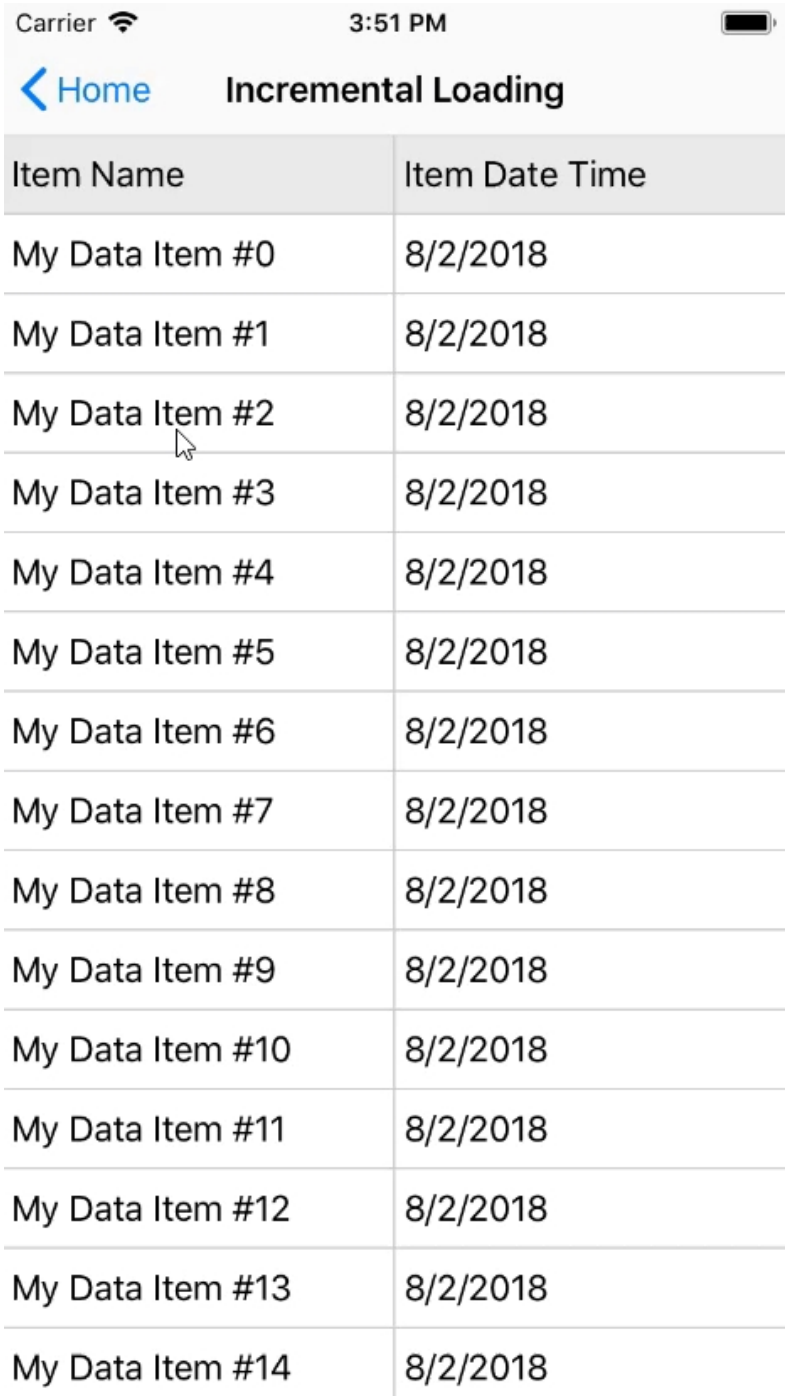
Incremental Loading

Incremental loading or on-demand loading is a powerful feature for mobile applications where data is loaded in chunks as the user scroll down a list in real time. Xamarin CollectionView supports incremental loading for data bound controls, such as FlexGrid and UITableView.

Adding incremental loading to ListView control is accomplished in two basic steps.

1. Implement your own **CollectionView** class, for example, OnDemandCollectionView, that extends C1CursorCollectionView class and overrides GetPageAsync. Once, this is completed you need to add logic that loads the data in pages or chunks.
2. Extend UITableView, FlexGrid, or another UI control to determine when the user has reached the bottom of a page.

The following GIF image shows incremental loading in the FlexGrid control.



The following code demonstrates how to implement incremental loading using CollectionView class.

C#

```
using System;
using UIKit;
using Foundation;
using C1.iOS.Grid;
using C1.CollectionView;
using CoreGraphics;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace CollectionView.iOS
{
    public class IncrementalLoading : UIViewController
    {
        public IncrementalLoading()
        {
        }
        public override void DidReceiveMemoryWarning()
        {
            // Releases the view if it doesn't have a superview.
            base.DidReceiveMemoryWarning();
            // Release any cached data, images, etc that aren't in use.
        }

        public override void ViewDidLoad()
        {
            base.ViewDidLoad();

            // Perform any additional setup after loading the view
            View.BackgroundColor = UIColor.White;
            Title = "Incremental Loading";

            var _collectionView = new OnDemandCollectionView();
            _grid = new FlexGrid();
            _grid.ItemsSource = _collectionView;
            _grid.LoadItemsOnDemand(_collectionView);
            for(int _column = 0; _column <= _grid.Columns.Count - 1; _column++)
            {
                _grid.Columns[_column].Width = GridLength.Star;
            }
            _grid.AllowDragging = GridAllowDragging.Both;
            _grid.AllowResizing = GridAllowResizing.Both;
            _grid.RowHeaders.Columns.Clear();

            var _navbarHeight = this.NavigationController.NavigationBar.Frame.Height +
UIApplication.SharedApplication.StatusBarFrame.Height;
            _grid.Frame = new
CGRect(this.View.Frame.X,this.View.Frame.Y+_navbarHeight,this.View.Frame.Width,this.View.Frame.Height-
_navbarHeight);
            this.View.AddSubview(_grid);
        }
        private FlexGrid _grid;
    }

    public class OnDemandCollectionView : C1CursorCollectionView<MyDataItem>
    {
        private bool hasMoreItems = true;

        public override bool HasMoreItems
        { get { return hasMoreItems; } }
```

```
        const int MAX = 100;
        int current;
        public int PageSize { get; set; }
        public OnDemandCollectionView()
        {
            PageSize = 20;
        }
        protected override async Task<Tuple<string, IReadOnlyList<MyDataItem>>> GetPageAsync(int
startingIndex, string pageToken, int? count = null, IReadOnlyList<SortDescription> sortDescriptions =
null, FilterExpression filterExpression = null, CancellationToken cancellationToken =
default(CancellationToken))
        {
            //No items to load further
            if (startingIndex + PageSize >= MAX)
            {
                hasMoreItems = false;
            }

            var newItems = new List<MyDataItem>();
            await Task.Run(() =>
            {
                for (int i = 0; i < this.PageSize; i++)
                {
                    //To mimick a long operation such as fetching data from a web-service
                    Thread.Sleep(100);
                    newItems.Add(new MyDataItem(startingIndex + i));
                    current++;
                }
            });
            return new Tuple<string, IReadOnlyList<MyDataItem>>("token not used", newItems);
        }
    }
    public class MyDataItem
    {
        public string ItemName { get; set; }
        public DateTime ItemDateTime { get; set; }
        public MyDataItem(int index)
        {
                this.ItemName = "My Data Item #" + index.ToString();
                this.ItemDateTime = DateTime.Now;
        }
    }
    public static class ListViewEx
    {
        public static void LoadItemsOnDemand<T>(this FlexGrid _flexGrid,
         C1CursorCollectionView<T> collectionView) where T : MyDataItem
        {
            if (collectionView.HasMoreItems)
            {
                collectionView.LoadMoreItemsAsync();
            }
        }
    }
}
```

Sorting

The ISupportSorting interface supports ascending and descending sorting for data controls, such as grids. It implements SortAsync method to allow you to call the sorting operation in the CollectionView without having to cast

to the specific interface. The SortAsync method sort the CollectionView according to the specified sort path and direction. To sort columns at run time, you can simply tap the column header of a particular column when using the FlexGrid control.

The image below shows how the FlexGrid appears, after you set the **SortAsync** method.

The following code examples demonstrate how to set this property in C#. These examples use the sample created in the Quick Start section.

In Code

C#

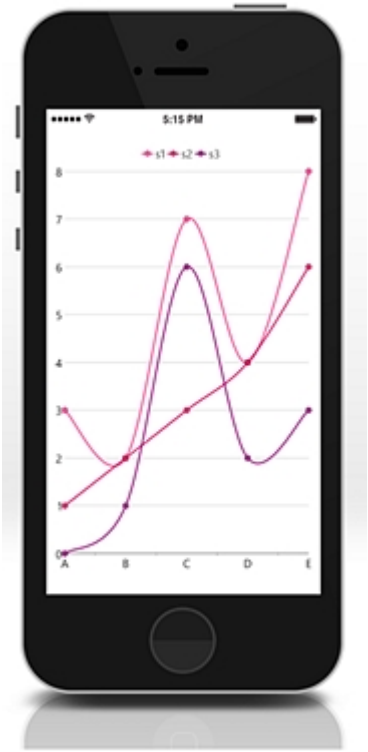
```
using CoreGraphics;
using UIKit;
using Foundation;
using C1.CollectionView;
using C1.iOS.Grid;
using System.Linq;

namespace CollectionView.iOS
{
    public class Sorting : UIViewController
    {
        public Sorting()
        {
        }
        public override void DidReceiveMemoryWarning()
        {
            // Releases the view if it doesn't have a superview.
            base.DidReceiveMemoryWarning();

            // Release any cached data, images, etc that aren't in use.
        }

        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            // Perform any additional setup after loading the view
            View.BackgroundColor = UIColor.White;
            Title = "Sorting";
            _grid = new FlexGrid();

            var data = Customer.GetCustomerList(100);
            _collectionView = new C1SortCollectionView<Customer>(data);

            var sort = _collectionView.SortDescriptions.FirstOrDefault(sd =>
sd.SortPath == "Name");
            var direction = sort != null ? sort.Direction : SortDirection.Descending;
            _collectionView.SortAsync("Name", direction == SortDirection.Ascending ?
SortDirection.Descending : SortDirection.Ascending);
            _grid.ItemsSource = _collectionView;
            _grid.SelectionMode = GridSelectionMode.Cell;
            _grid.RowHeaders.Columns.Clear();
            var _navbarHeight = this.NavigationController.NavigationBar.Frame.Height
+ UIApplication.SharedApplication.StatusBarFrame.Height;
            _grid.Frame = new CGRect(this.View.Frame.X,
this.View.Frame.Y+_navbarHeight, this.View.Frame.Width, this.View.Frame.Height-
```

```
_navbarHeight);
            this.View.AddSubview(_grid);
        }

        private FlexGrid _grid;
        private C1SortCollectionView<Customer> _collectionView;
    }
}
```

FlexChart

The FlexChart control allows you to represent data visually in your iOS mobile 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.

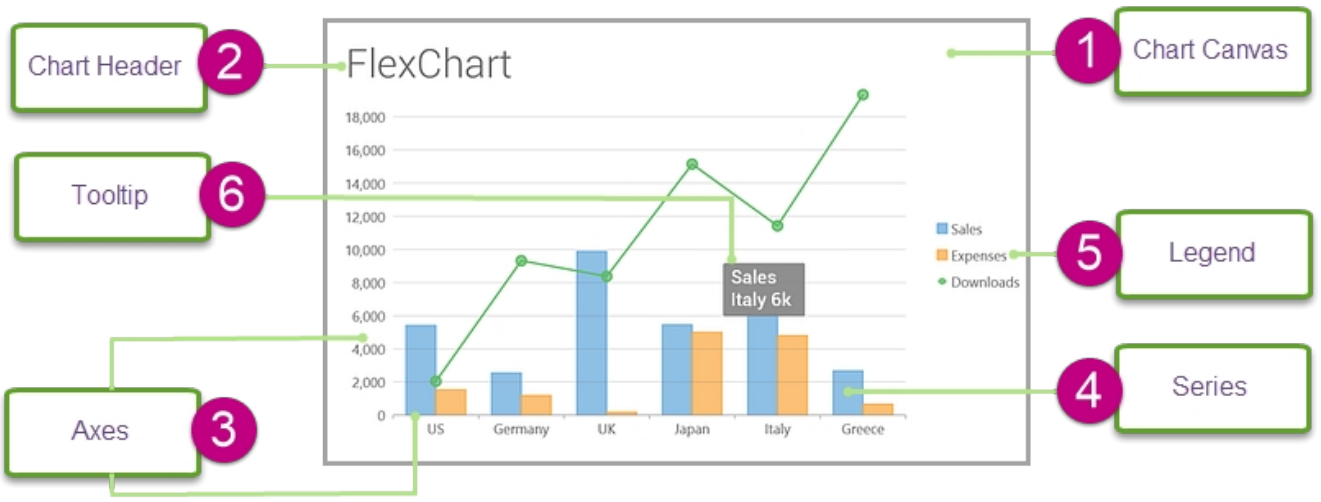


Key Features

- **Chart Type**: Change a line chart to a bar chart or any other chart type by setting a single property. FlexChart supports nine different chart types.
- **Touch Based Labels**: Display chart values using touch based labels.
- **Multiple Series**: Add multiple series on a single chart.

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 case of adding multiple series to FlexChart, each series of the chart are of the default chart type selected for that chart. However, you can set chart type for each series in code.

In Code

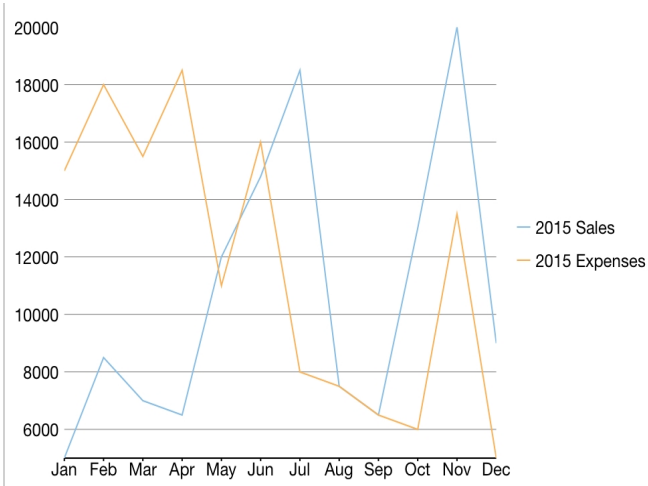
C#

```
chart.ChartType = ChartType.Area;
```

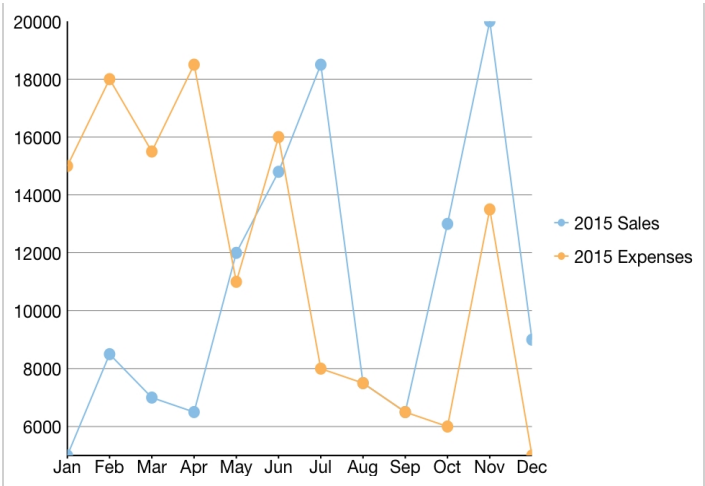
Line and LineSymbol chart

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

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



Line Chart

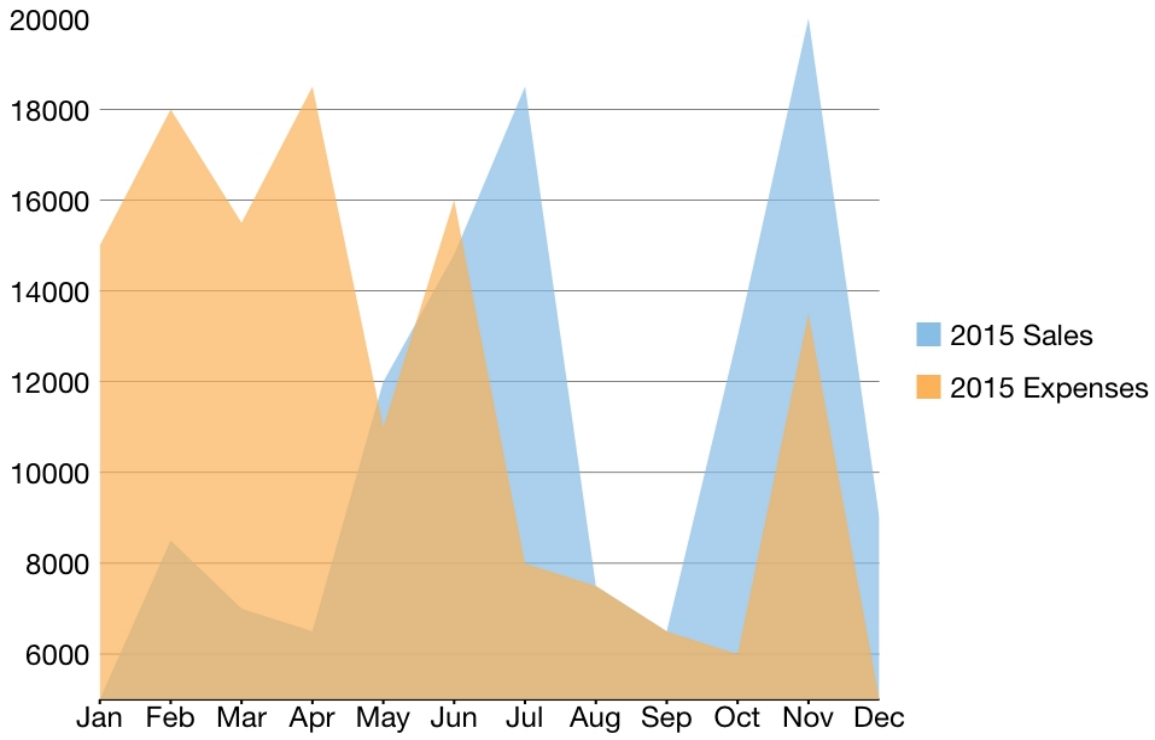


LineSymbol Chart

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.

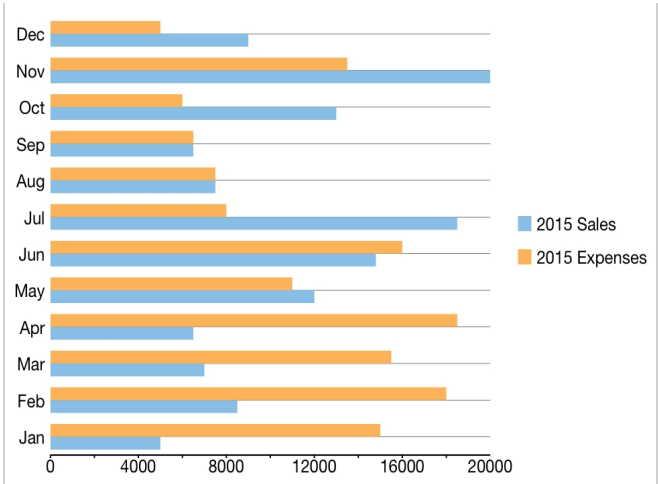
These charts are commonly used to show trends between associated attributes over time.



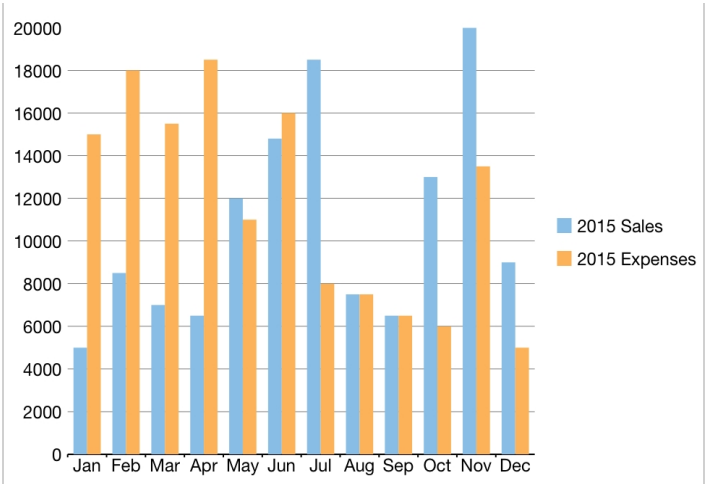
Bar and Column chart

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

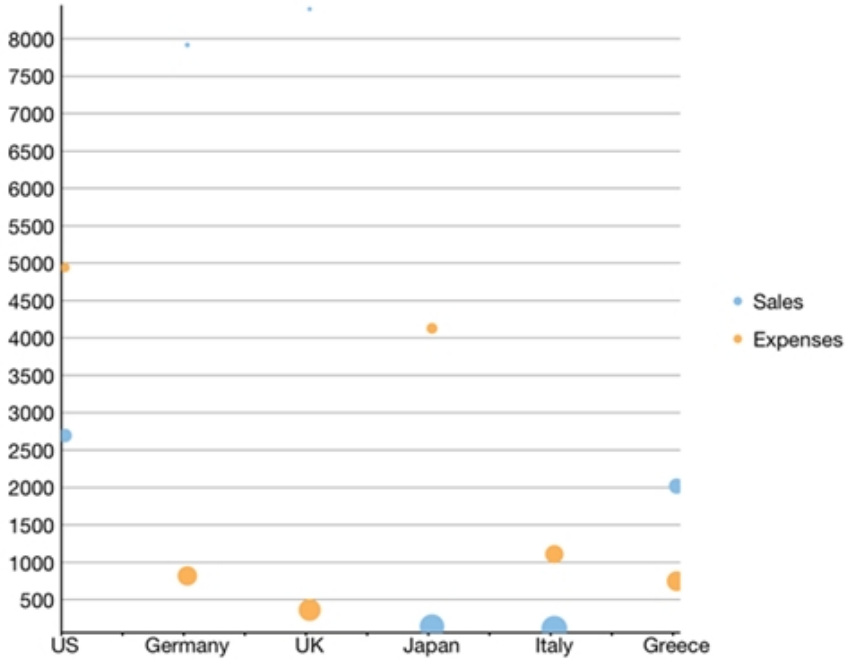


Column Chart

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.

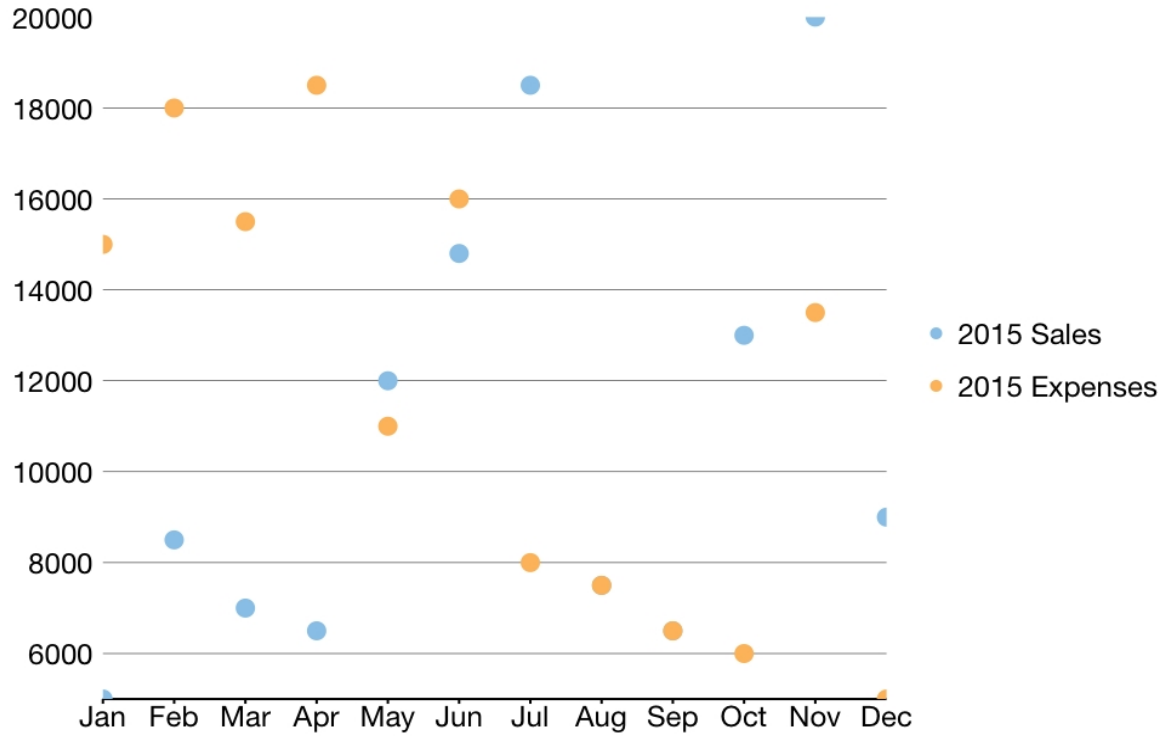
These charts are used to compare entities based on their relative positions on the axis as well as their size.



Scatter

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.



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 values of data. 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

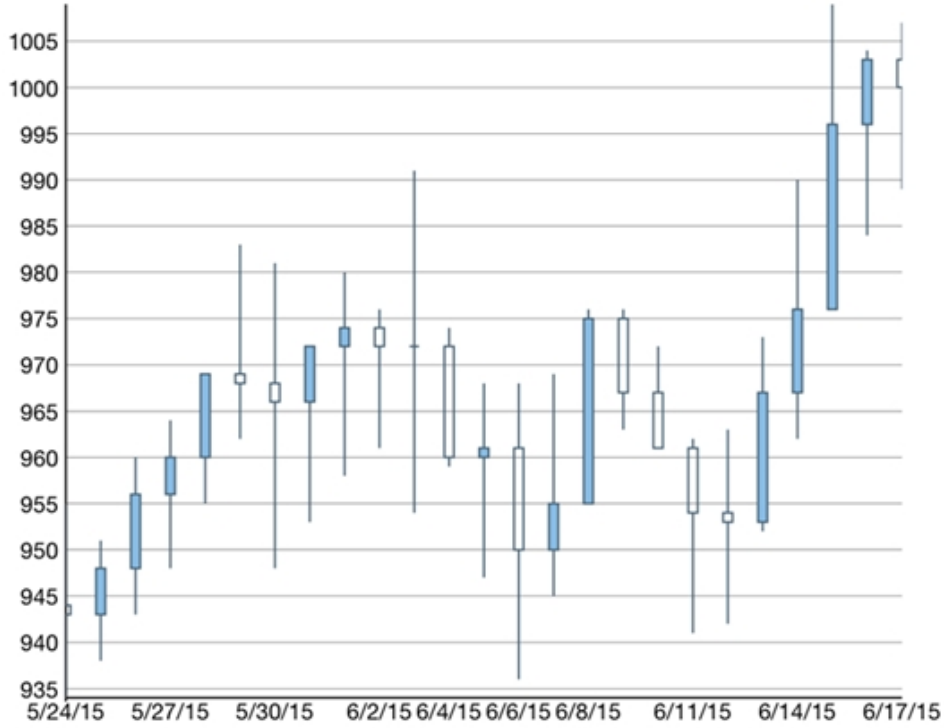
The Candlestick chart is made up of the following elements: **candle**, **wick**, and **tail**.

- **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.

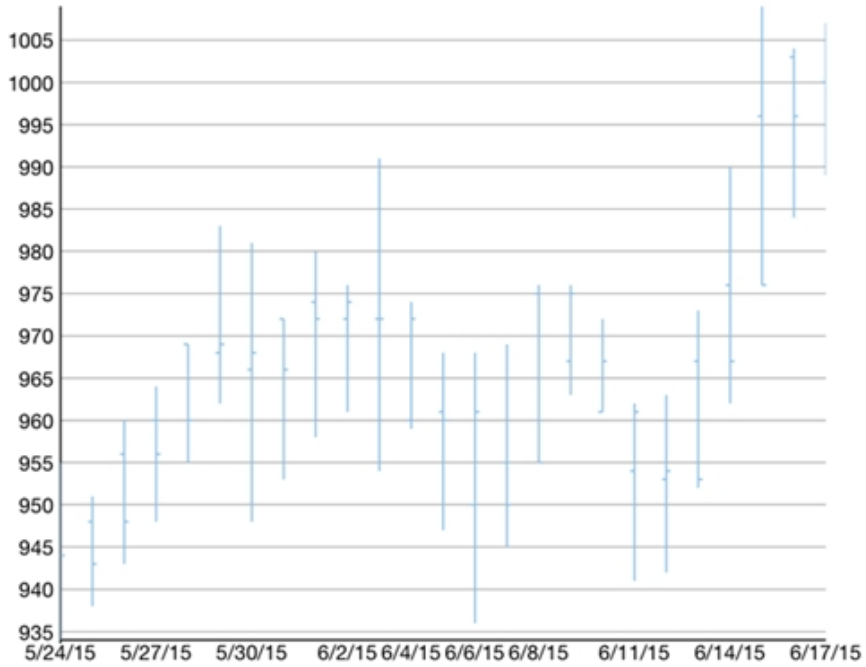
- **x:** Determines the date position along the x axis.
- **high:** Determines the highest price for the day, and plots it as the top of the candle along the y axis.
- **low:** Determines the lowest price for the day, and plots it as the bottom of the candle along the y axis.
- **open:** Determines the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

The following image shows a candlestick chart displaying stock prices.



High Low Open Close chart

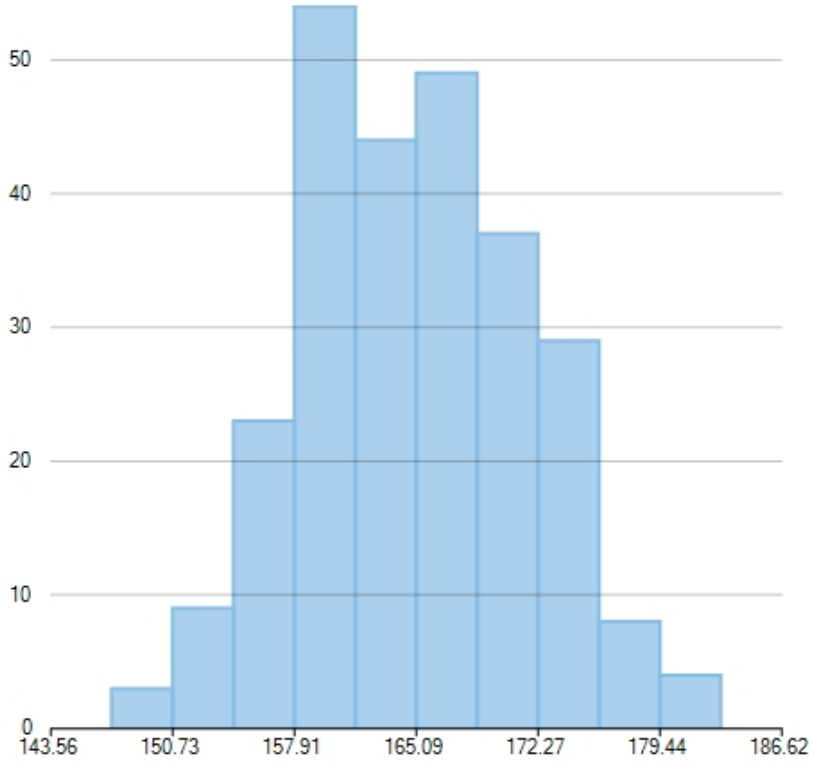
HiLoOpenClose are financial charts that combine four independent values to supply high, low, open and close data for a point in a series. In addition to showing the high and low value of a stock, the Y2 and Y3 array elements represent the stock's opening and closing price respectively.



Histogram Chart

Histogram chart plots the frequency distribution of data against the defined class intervals or bins. These bins are created by dividing the raw data values into a series of consecutive and non-overlapping intervals. Based on the number of values falling in a particular bin, frequencies are then plotted as rectangular columns against continuous x-axis.

These charts are commonly used for visualizing distribution of numerical data over a continuous, or a certain period of time.

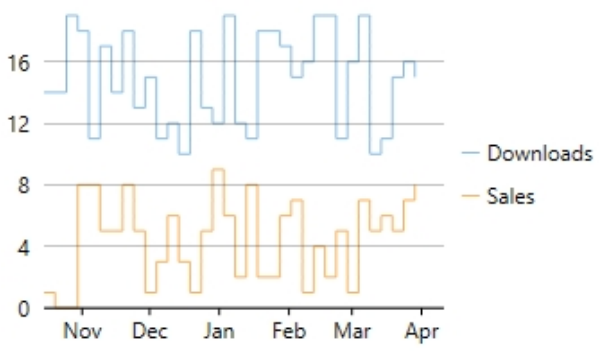


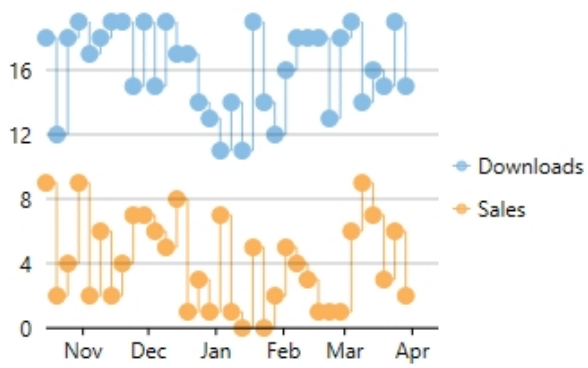
Step Chart

Step charts use horizontal and vertical lines to present data that show sudden changes along y-axis by discrete amount. These charts help display changes that are sudden and irregular but stay constant till the next change. Step charts enable judging trends in data along with the duration for which the trend remained constant.

Consider a use case where you want to visualize and compare weekly sales and units downloaded of a software. As both of these values vary with discrete amounts, you can use step chart to visualize them. As shown in the image below, apart from depicting the change in sales these charts also show the exact time of change and the duration for which sales were constant. Moreover, you can easily identify the magnitude of respective changes by simply looking at the chart.

FlexChart supports Step chart, StepSymbols chart, and StepArea or filled step chart. The following table gives detailed explanation of these chart types.

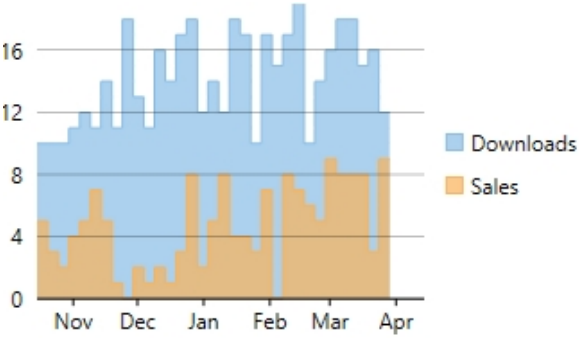
 **Step Chart**	Step chart is similar to the Line chart, except that Line chart uses shortest distance to connect consecutive data points, while Step chart connects them with horizontal and vertical lines. These horizontal and vertical lines give the chart step-like appearance. While the line charts depict change and its trend, the Step charts also help in judging the magnitude and the intermittent pattern of the change.



StepSymbols Chart

StepSymbols chart combines the Step chart and the Scatter chart. FlexChart plots data points by using symbols and connects those data points with horizontal and vertical step lines.

Here, the data points are marked using symbols and, therefore, help mark the beginning of an intermittent change.



StepArea Chart

StepArea chart combines the Step chart and the Area chart. It is similar to Area chart with the difference in the manner in which data points are connected. FlexChart plots the data points using horizontal and vertical step lines, and then fills the area between x-axis and the step lines.

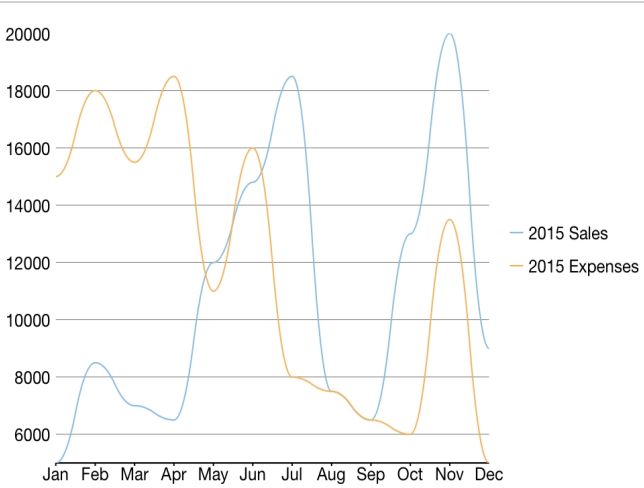
These are based on Step charts, and are commonly used to compare discrete and intermittent changes between two or more quantities. This gives the chart stacked appearance, where related data points of the multiple series seem stacked above the other.

For example, number of units downloaded and sales of a software for a particular time duration can be easily compared as shown in the image.

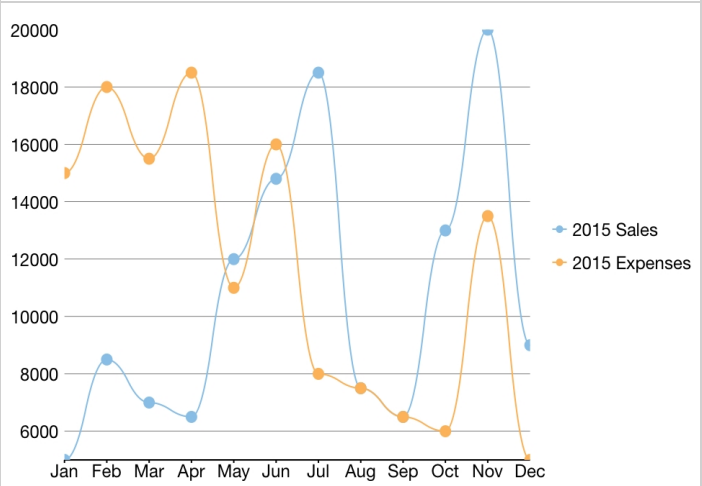
Spline and SplineSymbol chart

A Spline chart is a combination of line and area charts. It draws a fitted curve through each data point and its series can be drawn independently or stacked. It is the most effective way of representing data that uses curve fittings to show difference of values. A SplineSymbol chart is similar to Spline chart except that it represents data points using symbols.

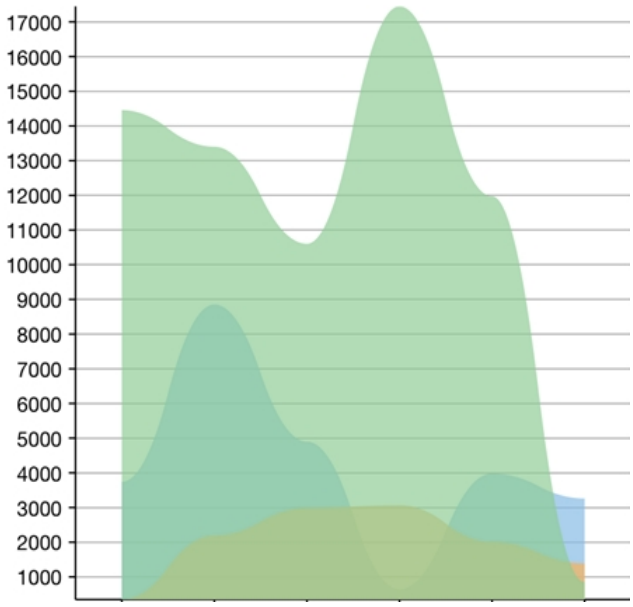
These charts are commonly used to show trends and performance over time, such as product life-cycle.



Spline Chart



SplineSymbol Chart



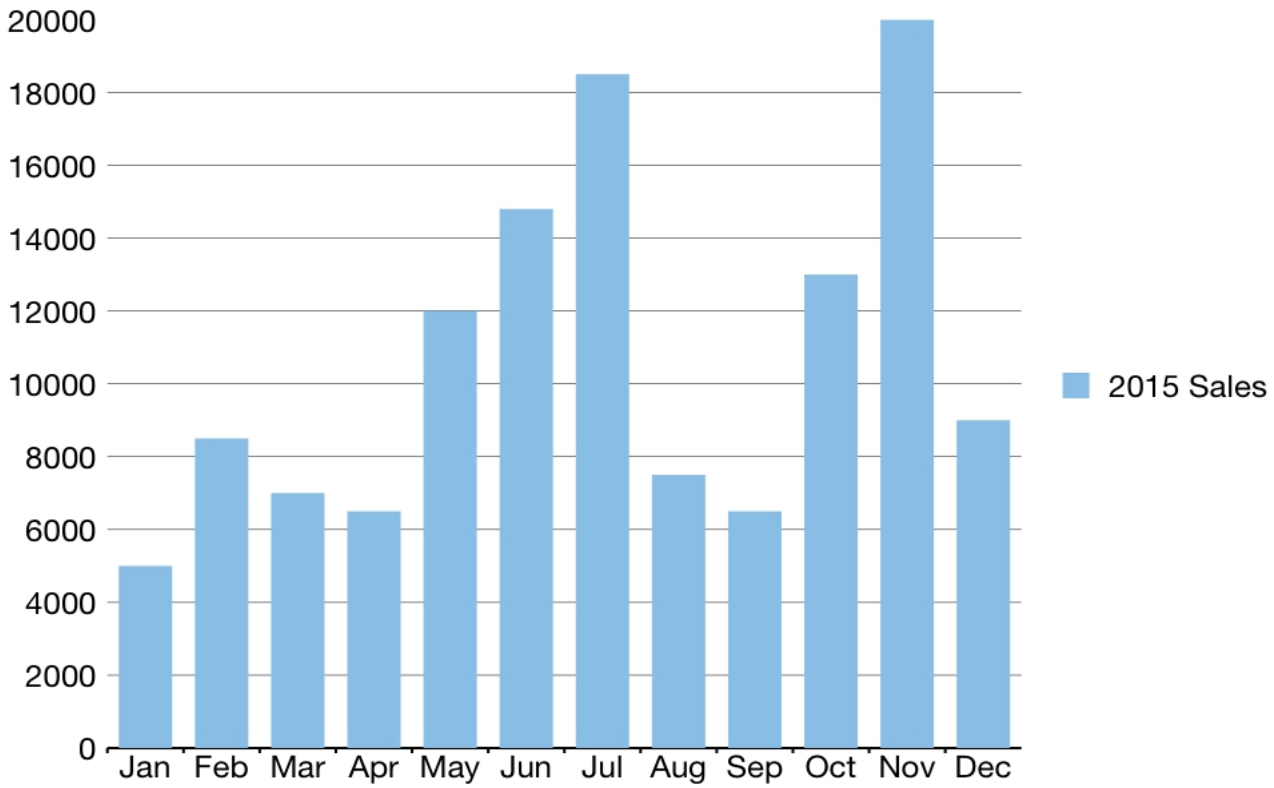
SplineArea Chart

Quick Start: Add Data to FlexChart

This section describes how to add a FlexChart control to your iOS app and add data to it. This topic comprises of three steps:

- **Step 1: Create a data source for FlexChart**
- **Step 2: Add a FlexChart control**
- **Step 3: Run the Application**

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



Step 1: Create a data source for FlexChart

Add a new class to serve as the data source for the FlexChart control.

FlexChartDataSource.cs

```
public class FlexChartDataSource
{
    private List<Month> appData;

    public List<Month> Data
    {
        get { return appData; }
    }

    public FlexChartDataSource()
    {
        // appData
        appData = new List<Month>();
        var monthNames =
"Jan,Feb,March,April,May,June,July,Aug,Sept,Oct,Nov,Dec".Split(',');
        var salesData = new[] { 5000, 8500, 7000, 6500, 12000, 14800, 18500, 7500, 6500,
13000, 20000, 9000 };
        var downloadsData = new[] { 6000, 7500, 12000, 5800, 11000, 7000, 16000, 17500,
19500, 13250, 13800, 19000 };
        var expensesData = new[] { 15000, 18000, 15500, 18500, 11000, 16000, 8000, 7500,
6500, 6000, 13500, 5000 };
        for (int i = 0; i < 12; i++)
        {
             Month tempMonth = new Month();
             tempMonth.Name = monthNames[i];
             tempMonth.Sales = salesData[i];
             tempMonth.Downloads = downloadsData[i];
             tempMonth.Expenses = expensesData[i];
             appData.Add(tempMonth);

        }
    }
}

public class Month
{
    string _name;
    long _sales, _downloads, _expenses;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }

    public long Sales
    {
        get { return _sales; }
        set { _sales = value; }
```

```
    }

    public long Downloads
    {
        get { return _downloads; }
        set { _downloads = value; }
    }
    public long Expenses
    {
        get { return _expenses; }
        set { _expenses = value; }
    }
}
```

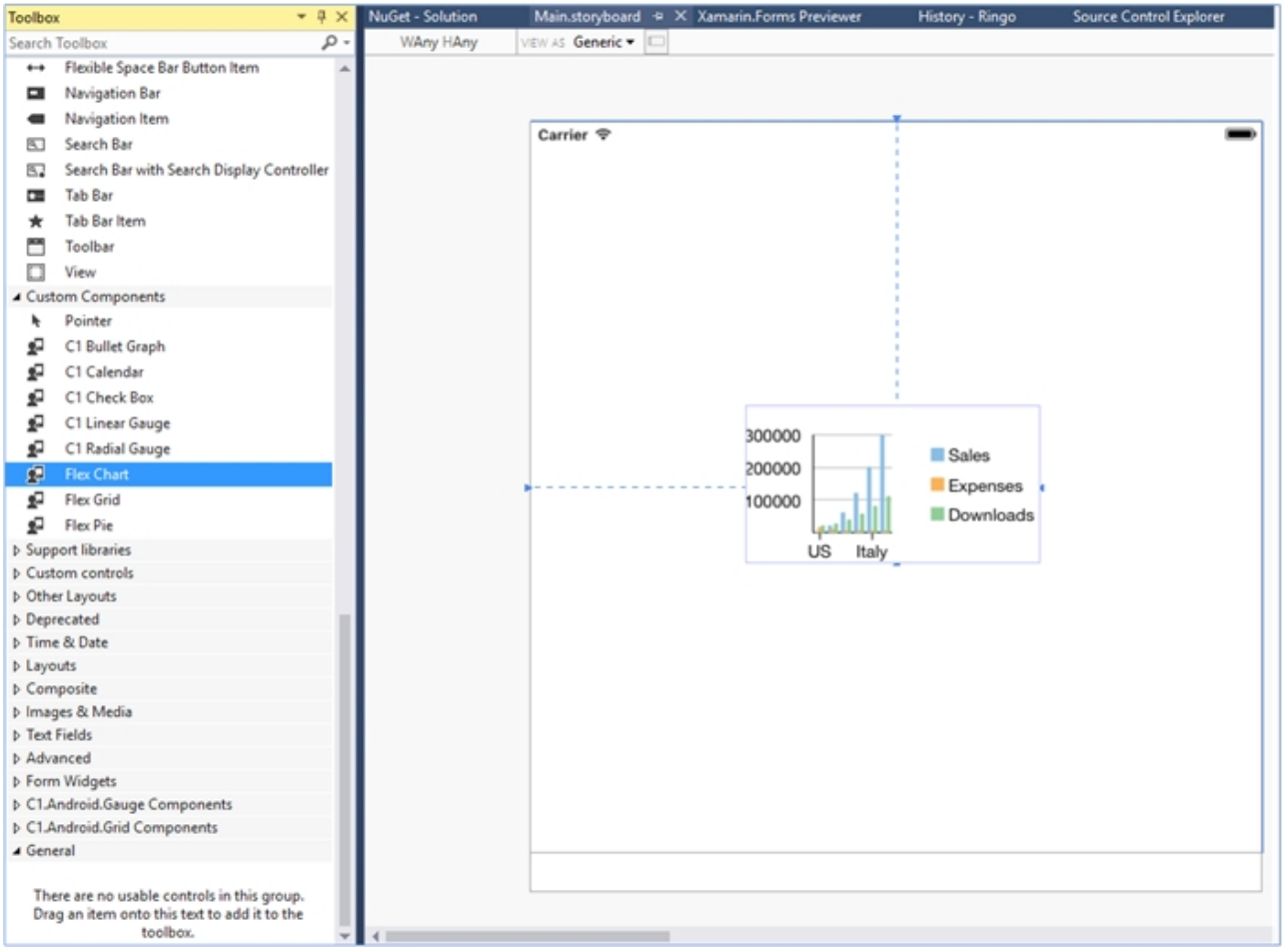
Back to Top

Step 2: Add a FlexChart control

Complete the following steps to initialize a FlexChart control in C#.

Add a FlexChart control in StoryBoard

1. In the **Solution Explorer**, click **MainStoryboard** to open the storyboard editor.
2. In your Toolbox under the **Custom Components** tab, drag a FlexChart onto your ViewController.



Initialize the FlexChart control in Code

To initialize the FlexChart control, open the ViewController file from the Solution Explorer and replace its content with the code below. This overrides the **ViewDidLoad** method of the view controller in order to initialize FlexChart.

C#

```
  public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            // Perform any additional setup after loading the view, typically from a nib.
            chart = new FlexChart();
            chart.BindingX = "Name";
            chart.Series.Add(new ChartSeries() { SeriesName = "Sales", Binding =
"Sales,Sales" });
            FlexChartDataSource SalesData = new FlexChartDataSource();
            chart.ItemsSource = SalesData.Data;
            this.Add(chart);
        }

        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            chart.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y + 80,
                                     this.View.Frame.Width, this.View.Frame.Height - 80);
        }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

Animation

FlexChart allows you to enable animation effects using one of the two ways, either on loading when the chart is drawn or on updating when the chart is redrawn after modifications. It supports animation in charts through C1Animation class available in the C1.iOS.Core namespace.

You can also set the duration of the animation in chart using Duration property of the C1Animation class and interpolate the values of animation using Easing property of the C1Animation class, which accepts values from the C1Easing class. This class supports a collection of standard easing functions such as CircleIn, CircleOut, and Linear.

- **CircleIn:** Easing function that starts slow and speeds up in the form of a circle.
- **CircleOut:** Easing function that starts fast and slows down in the form of a circle.
- **Linear:** Easing function with constant speed.

C#

```
C1Animation animate = new C1Animation();
// set update animation duration
animate.Duration = new TimeSpan(3000 * 10000);
```

```
// interpolate the values of animation
animate.Easing = C1Easing.Linear;
```

In addition to easing functions of C1Easing class, FlexChart supports built in easing functions of Xamarin.Forms.Easing class. For more information, refer Xamarin Easing Class.

You can show animation while loading or updating a chart. To show animation while loading, use LoadAnimation property of the ChartBase class. This property gets the load animation from the object of C1Animation class to display the animation at the time of loading chart. Similarly, to animate the chart when underlying data collection changes on adding, removing, or modifying a value, you can use UpdateAnimation property of the ChartBase class.

C#

```
// set the loading animation
chart.LoadAnimation = animate;
```

You can apply the loading animation effect by setting the AnimationMode property which accepts values from AnimationMode enumeration. This enumeration supports four different animation modes: All, None, Series, and Point.

- **All**: All plot elements animate at once from the bottom of the plot area.
- **None**: Does not display any animation.
- **Series**: Each series animates one at a time from the bottom of the plot area.
- **Point**: The plot elements appear one at a time from left to right.

C#

```
// set the animation mode
chart.AnimationMode = AnimationMode.Series;
```

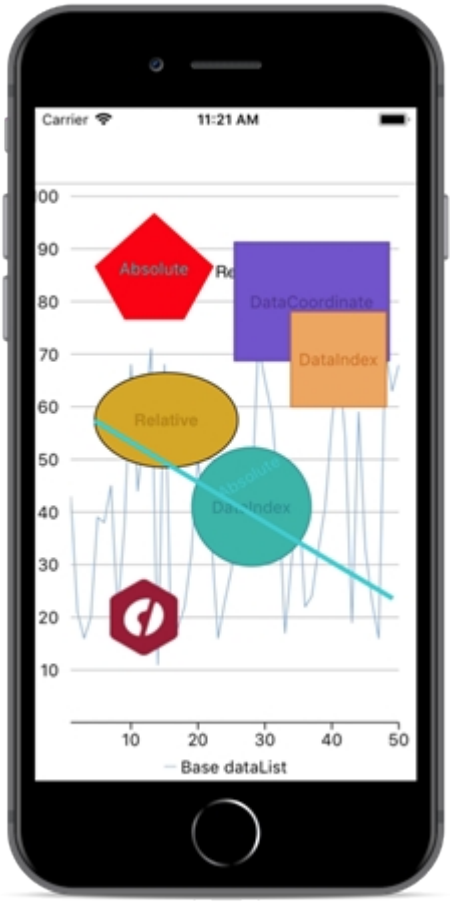
Annotations

Annotations are used to mark important news or events that can be attached to a specific data point on FlexChart. Annotations can also be used to place arbitrary elements such as images, shapes and text onto the chart. The FlexChart control supports various built-in annotations such as Polygon, Line, Ellipsis, Rectangle, Image and Text.

You can specify the position of an annotation on FlexChart by setting the **Position** property to Bottom, Center, Left, Right or Top. To specify the type of annotation on FlexChart, you can use the **Content** property and set its value to:

- **Absolute**: The coordinates of the annotation are specified by the annotation's shape data in pixels.
- **DataCoordinate**: The coordinates of the annotation are specified in data coordinates.
- **DataIndex**: The coordinates of the annotation are specified by the data series index and the data point index.
- **Relative**: The coordinates of the annotation are specified as a relative position within the control, where (0, 0) is the top left corner and (1, 1) is the bottom right corner.

The following image shows how FlexChart control appears after applying annotations to it.



Complete the following steps to apply annotations to the **FlexChart** control:

1. Add the following import statements in the AnnotationController.cs file.

C#

```
using System;using System.Collections.Generic;using UIKit;using
CoreGraphics;using C1.iOS.Chart;using C1.iOS.Chart.Annotation;using CoreText;
```

2. Replace the code in the AnnotationController.cs file with the following code.

C#

```
public class AnnotationViewModel {
 List <DataItem> _data;
 List <DataItem> _simpleData;
 Random rnd = new Random();

 public List <DataItem> Data {
  get {
   if (_data == null) {
    _data = new List < DataItem > ();
    for (int i = 1; i < 51; i++) {
     _data.Add(new DataItem() {
      X = i,
       Y = rnd.Next(10, 100)
     });
    }
   }
```

```
    return _data;
   }
  }

  public List <DataItem> SimpleData {
   get {
    if (_simpleData == null) {
     _simpleData = new List <DataItem> ();
     _simpleData.Add(new DataItem() {
      X = 1, Y = 30
     });
     _simpleData.Add(new DataItem() {
      X = 2, Y = 20
     });
     _simpleData.Add(new DataItem() {
      X = 3, Y = 30
     });
     _simpleData.Add(new DataItem() {
      X = 4, Y = 65
     });
     _simpleData.Add(new DataItem() {
      X = 5, Y = 70
     });
     _simpleData.Add(new DataItem() {
      X = 6, Y = 60
     });
    }
    return _simpleData;
   }
  }
}
public class DataItem {
 public int X {
  get;
  set;
 }
 public int Y {
  get;
  set;
 }
}
public partial class AnnotationController: UIViewController {
  FlexChart chart;
  public AnnotationController(): base("AnnotationController", null) {}
  public AnnotationController(IntPtr handle): base(handle) {
  }
  public override void ViewDidLoad() {
   base.ViewDidLoad();
   // Perform any additional setup after loading the view, typically from a nib.
   chart = new FlexChart();
   chart.LegendPosition = ChartPositionType.Bottom;
```

```
    chart.ChartType = ChartType.Line;
    chart.BindingX = "X";
    chart.Series.Add(new ChartSeries() {
     SeriesName = "Base dataList", Binding = "Y,Y"
    });
    chart.ItemsSource = new AnnotationViewModel().Data;
    this.Add(chart);

    Text text = new Text() {
     Content = "Relative", Location = new CGPoint(0.5, 0.5), Attachment =
AnnotationAttachment.Relative
    };
    text.AnnotationStyle = new ChartStyle() {
     FontSize = 15, Stroke = UIColor.Black, FontFamily =
UIFont.FromName("GenericSansSerif", 15)
    };

    Ellipse ellipse = new Ellipse() {
     Content = "Relative", Location = new CGPoint(0.4, 0.45), Width = 120, Height
= 80, Attachment = AnnotationAttachment.Relative
    };
    ellipse.AnnotationStyle = new ChartStyle() {
     Fill = UIColor.FromRGBA(0.85 f, 0.65 f, 0.12 f, 1.0 f), Stroke =
UIColor.FromRGBA(0.75 f, 0.55 f, 0.06 f, 1.0 f), FontFamily =
UIFont.ItalicSystemFontOfSize(10)
    };

    Circle circle = new Circle() {
     Content = "DataIndex", Radius = 50, SeriesIndex = 0, PointIndex = 27,
Attachment = AnnotationAttachment.DataIndex
    };
    circle.AnnotationStyle = new ChartStyle() {
     Fill = UIColor.FromRGBA(0.17 f, 0.70 f, 0.67 f, 1.0 f), Stroke =
UIColor.FromRGBA(0.13 f, 0.58 f, 0.58 f, 1.0 f), FontFamily =
UIFont.BoldSystemFontOfSize(10)
    };

    Rectangle rectangle = new Rectangle() {
     Content = "DataCoordinate", Width = 130, Height = 100, Location = new
CGPoint(37, 80), Attachment = AnnotationAttachment.DataCoordinate
    };
    rectangle.AnnotationStyle = new ChartStyle() {
     Fill = UIColor.FromRGBA(0.42 f, 0.35 f, 0.80 f, 1.0 f), Stroke =
UIColor.FromRGBA(0.29 f, 0.25 f, 0.57 f, 1.0 f), FontFamily =
UIFont.BoldSystemFontOfSize(10)
    };

    Square square = new Square() {
     Content = "DataIndex", Length = 80, SeriesIndex = 0, PointIndex = 40,
Attachment = AnnotationAttachment.DataIndex
    };
```

```
    square.AnnotationStyle = new ChartStyle() {
     Fill = UIColor.FromRGBA(0.96 f, 0.64 f, 0.38 f, 1.0 f), Stroke =
UIColor.FromRGBA(0.89 f, 0.54 f, 0.25 f, 1.0 f), FontFamily =
UIFont.BoldSystemFontOfSize(10)
    };

    Polygon polygon = new Polygon() {
     Content = "polygonAnno", Attachment = AnnotationAttachment.Absolute
    };
    polygon.Points = CreatePoints();
    polygon.AnnotationStyle = new ChartStyle() {
     Fill = UIColor.Red, StrokeThickness = 3, Stroke = UIColor.FromRGBA(0.98 f,
0.06 f, 0.05 f, 1.0 f), FontFamily = UIFont.BoldSystemFontOfSize(10)
    };

    Line line = new Line() {
     Content = "Absolute", Start = new CGPoint(50, 200), End = new CGPoint(300,
350), Attachment = AnnotationAttachment.Absolute
    };
    line.AnnotationStyle = new ChartStyle() {
     StrokeThickness = 4, FontSize = 10, FontAttributes =
CTFontSymbolicTraits.Bold, Stroke = UIColor.FromRGBA(0.20 f, 0.81 f, 0.82 f, 1.0
f), FontFamily = UIFont.BoldSystemFontOfSize(10)
    };

    Image image = new Image() {
     Location = new CGPoint(12, 20), Attachment =
AnnotationAttachment.DataCoordinate
    };
    image.Source = new UIImage("Images/xuni_butterfly.png");
    AnnotationLayer layer = new AnnotationLayer();
    layer.Annotations.Add(text);
    layer.Annotations.Add(ellipse);
    layer.Annotations.Add(circle);
    layer.Annotations.Add(rectangle);
    layer.Annotations.Add(square);
    layer.Annotations.Add(polygon);
    layer.Annotations.Add(line);
    layer.Annotations.Add(image);
    chart.Layers.Add(layer);
   }
   private System.Collections.ObjectModel.ObservableCollection <CGPoint>
CreatePoints() {
    System.Collections.ObjectModel.ObservableCollection <CGPoint> points = new
System.Collections.ObjectModel.ObservableCollection <CGPoint>();

    points.Add(new CGPoint(100, 25));
    points.Add(new CGPoint(50, 70));
    points.Add(new CGPoint(75, 115));
    points.Add(new CGPoint(125, 115));
    points.Add(new CGPoint(150, 70));
```

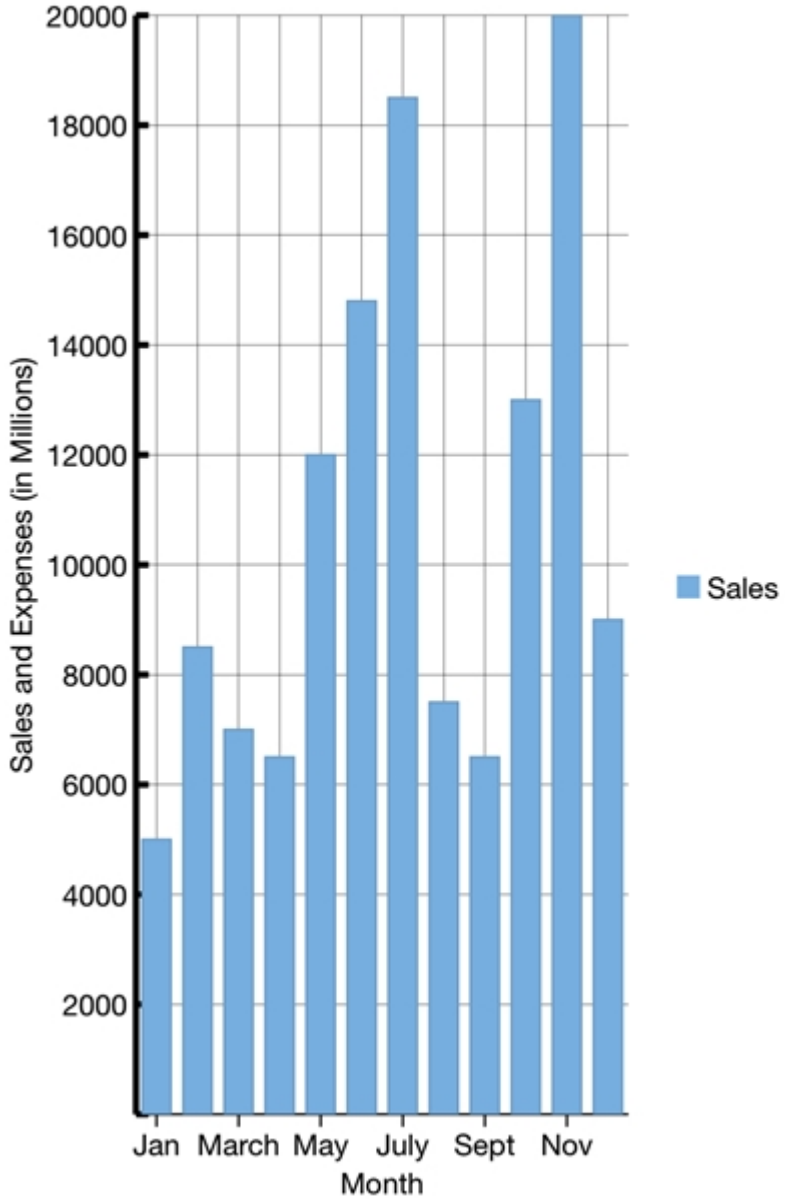
```
  return points;
 }
 public override void ViewDidLayoutSubviews() {
  base.ViewDidLayoutSubviews();
  chart.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y + 80,
   this.View.Frame.Width, this.View.Frame.Height - 80);
 }
```

Axis

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. For more information, see standard format strings available in .Net for more information.

> Axis line for Y axis and grid lines on the X axis are disabled by default. To enable the axis lines and grid lines, set the AxisLine of Y axis and MajorGrid property of X axis to **true**.

The image below shows a FlexChart with customized axes.



The following code example demonstrates how to customize the axes. The example uses the sample created in the Customize Appearance section.

In Code

C#

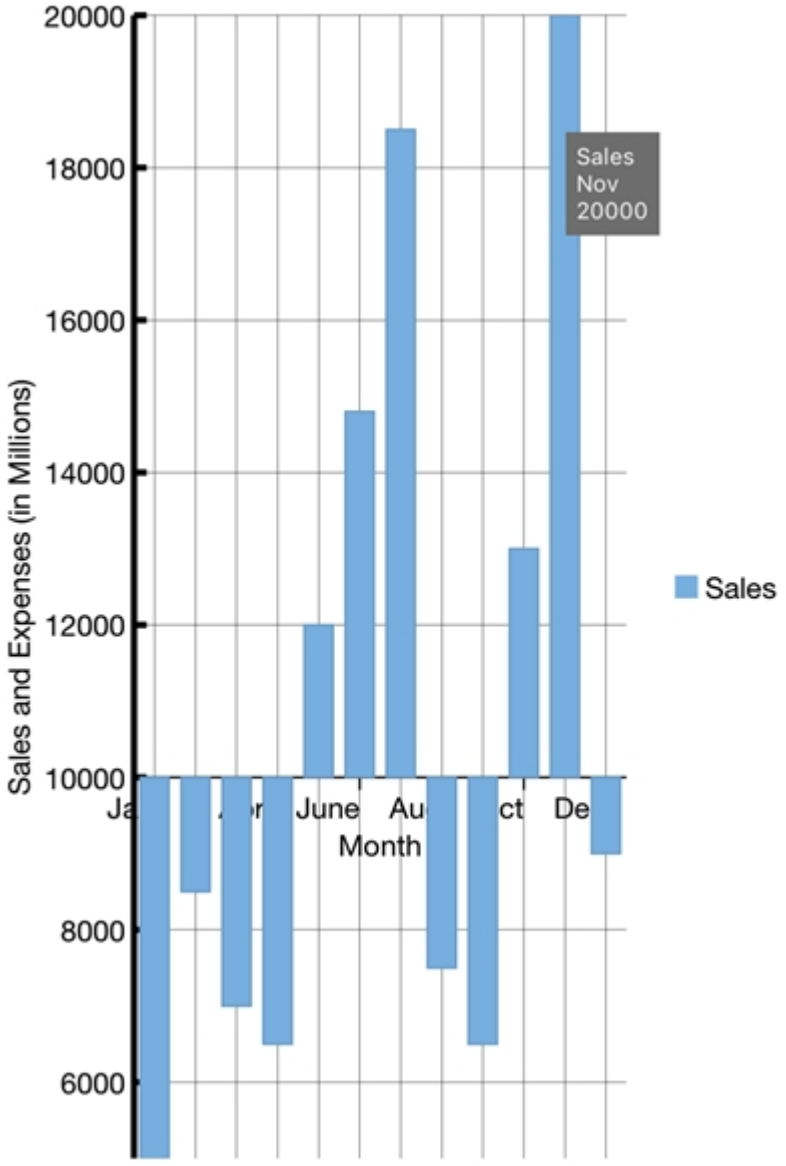
```
chart.AxisY.AxisLine = true;
chart.AxisY.MajorGrid = true;
chart.AxisY.Style.StrokeThickness = 3;
chart.AxisY.Title = "Sales and Expenses (in Millions)";
chart.AxisY.MajorTickMarks = ChartTickMarkType.Inside;
chart.AxisY.MajorUnit = 2000;

chart.AxisX.AxisLine = true;
chart.AxisX.MajorGrid = true;
chart.AxisX.Style.Fill = UIColor.Blue;
chart.AxisX.Title = "Month";
```

```
chart.AxisX.MajorGridStyle.Fill = UIColor.Blue;
```

Customizing Axis Origin

Users can customize the FlexChart's axis origin to plot data points in all the four quadrants. Use **AxisX.Origin** and **AxisY.Origin** properties to set the origin for both the axes at a particular point. This is useful in scenarios in trend analysis, where a user wants to plot positive as well as negative values on the chart. With axis origin set to zero, the negative values are plotted below the horizon and positive values above it. This provides better visualization to a user for data analysis on the chart.

The image given below shows how the FlexChart appears when its **X axis** origin is set to **1000**.



In Code

The following code example illustrates how to customize the X-axis origin. This example uses the sample created in Customize Appearance section. Similarly, you can set the value of Y-axis as well, to display the values in first and fourth quadrants.

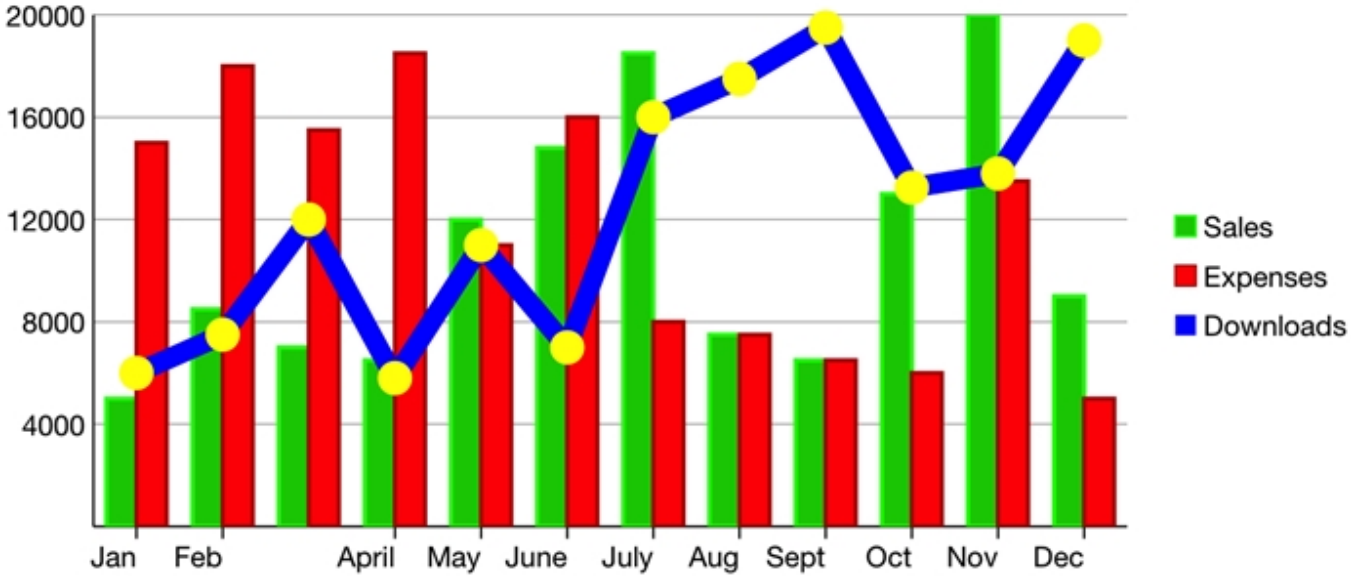
C#

```
//Set axis X origin
chart.AxisX.Origin = 10000;
```

Customize Appearance

Xamarin controls for native iOS are designed to work with both: **light** and **dark** themes available on all platforms. However, there are several other properties that allow you to customize the appearance of the FlexChart control. You can change the background color of the chart, set the color of the series, add colored borders of specified thickness to charts as well as series and do much more to enhance the appearance of the control.

The image below shows a customized FlexChart control.



The following code example demonstrates how to customize the FlexChart and its series. The example uses the sample created in the Quick Start section, with multiple series added to the chart. For more information on how to add multiple series, see Mixed Charts.

In Code

C#

```
chart.Series.Add(new ChartSeries() {
 SeriesName = "Sales", Binding = "Sales,Sales", Style = new ChartStyle {
  Fill = new UIColor(0, 0.8 f, 0, 1), Stroke = UIColor.Green, StrokeThickness = 2
 }
});
chart.Series.Add(new ChartSeries() {
 SeriesName = "Expenses", Binding = "Expenses,Expenses", Style = new ChartStyle {
  Fill = UIColor.Red, Stroke = new UIColor(0.702 f, 0, 0, 1), StrokeThickness = 2
 }
});
chart.Series.Add(new ChartSeries() {
 SeriesName = "Downloads", Binding = "Downloads,Downloads", ChartType =
C1.iOS.Chart.ChartType.LineSymbols,
```

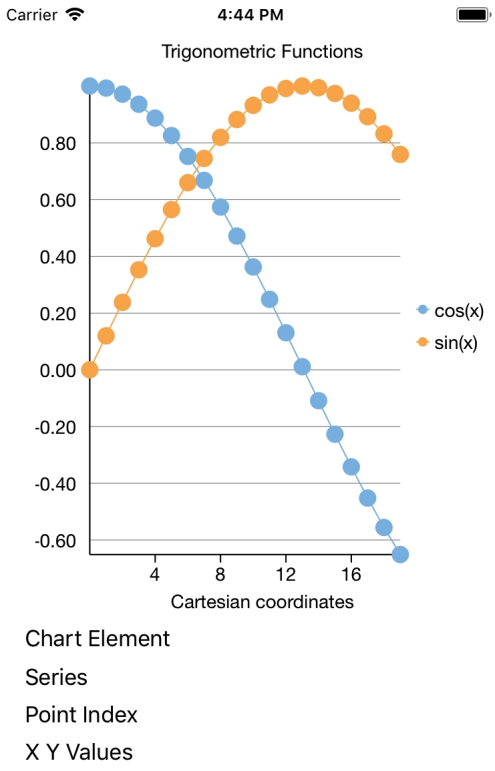
```
  Style = new ChartStyle {
   Fill = new UIColor(1, 0.416 f, 1, 1), Stroke = UIColor.Blue, StrokeThickness = 10
  },
  SymbolStyle = new ChartStyle {
   Fill = UIColor.Yellow, Stroke = UIColor.Yellow, StrokeThickness = 5
  }
});
```

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 user taps. This method is helpful in scenarios such as displaying tooltips that lie outside the series of the FlexChart.

The following code example demonstrates how to define the **chart_Tapped** event. This event invokes the **HitTest()** method to retrieve the information of the tapped point in the FlexChart region and displays it in the chart footer.

The image below shows how the FlexChart appears, after defining the **HitTest()** method.



To initialize the FlexChart control and use HitTest() method, open the ViewController.cs and replace its content with the code below. This overrides the **viewDidLoad** method of the view controller.

CS

```
public partial class ViewController : UIViewController
{
    public ViewController(IntPtr handle) : base(handle)
    {
```

```
    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
        // Perform any additional setup after loading the view, typically from a nib.

        int len = 20;
        List<CGPoint> listCosTuple = new List<CGPoint>();
        List<CGPoint> listSinTuple = new List<CGPoint>();

        for (int i = 0; i < len; i++)
        {
            listCosTuple.Add(new CGPoint(i, Math.Cos(0.12 * i)));
            listSinTuple.Add(new CGPoint(i, Math.Sin(0.12 * i)));
        }

        //chart = new FlexChart();
        chart.BindingX = "X";
        chart.ChartType = ChartType.LineSymbols;
        chart.AxisY.Format = "F";
        chart.Header = "Trigonometric Functions";
        chart.Footer = "Cartesian coordinates";

        chart.Series.Add(new ChartSeries() { SeriesName = "cos(x)", Binding = "Y,Y",
ItemsSource = listCosTuple });
        chart.Series.Add(new ChartSeries() { SeriesName = "sin(x)", Binding = "Y,Y",
ItemsSource = listSinTuple });

        chart.Tapped += OnChartTapped;
    }

    private void OnChartTapped(object sender, C1TappedEventArgs e)
    {
        var pt = e.GetPosition(chart);
        var info = chart.HitTest(pt);

        if (info != null)
        {
            pointIndexLabel.Text =
string.Format(NSBundle.MainBundle.LocalizedString("Point Index", "Point Index") + "
{0}", info.PointIndex);
            xyValuesLabel.Text = string.Format(NSBundle.MainBundle.LocalizedString("X
Y Values", "X Y Values") + " X: {0}, Y: {1:F2}", info.X, info.Y);

            if (info.Series != null)
            {
                seriesLabel.Text =
string.Format(NSBundle.MainBundle.LocalizedString("SeriesName", "SeriesName") + "
{0}", info.Series.Name);
```

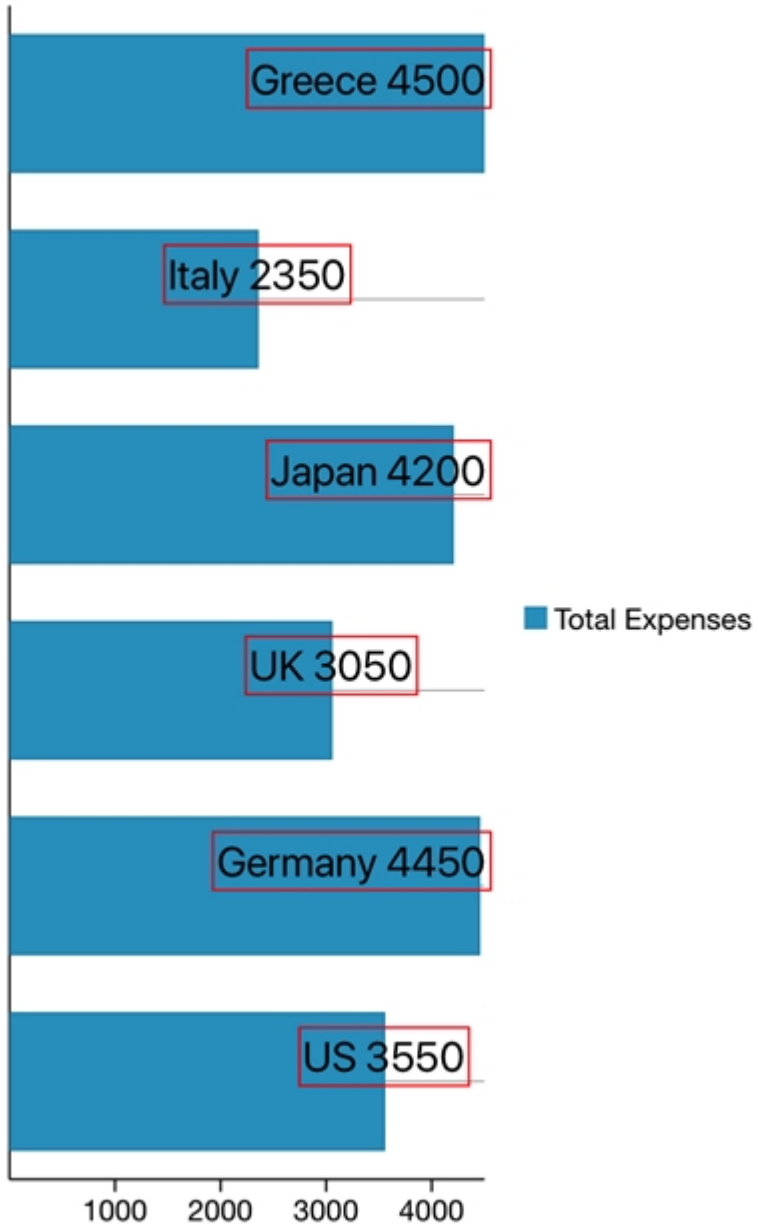
```
            }
            else
            {
                seriesLabel.Text = NSBundle.MainBundle.LocalizedString("SeriesName",
"SeriesName");
            }
            chartElementLabel.Text =
string.Format(NSBundle.MainBundle.LocalizedString("Chart element", " Chart element")
+ " {0}", info.ChartElement.ToString());
        }
    }

    public override void DidReceiveMemoryWarning()
    {
        base.DidReceiveMemoryWarning();
        // Release any cached data, images, etc that aren't in use.
    }
}
```

Data Labels

You can add data labels in the FlexChart to show the exact values corresponding to a particular column, bar or point in the chart area. The position property of the FlexChart data labels can be used to set the position of the data labels. Users can set data labels at the following positions.

- **TOP** - Displays data labels on the top of the column, bar or point in the chart area
- **BOTTOM** - Displays data labels below the edge of the column, bar or point in the chart area
- **RIGHT** - Displays data labels to the right side of the column, bar or point in the chart area
- **LEFT** - Displays data labels to the left side of the column, bar or point in the chart area
- **CENTER** - Displays data labels to the center of the column, bar or point in the chart area

The image given below shows a FlexChart control that displays sales data with data labels corresponding to months.



The following code example shows how to set data labels for FlexChart control.

C#

```
chart = new FlexChart();
chart.ChartType = ChartType.Bar;
chart.BindingX = "Name";
chart.Series.Add(new ChartSeries() { SeriesName = "Total Expenses", Binding =
"Expenses" });
chart.ItemsSource = SalesData.GetSalesDataList2();
chart.Palette = Palette.Modern;
chart.AxisY.AxisLine = true;
chart.AxisY.Labels = false;
this.Add(chart);

chart.DataLabel.Position = ChartLabelPosition.Top;
```

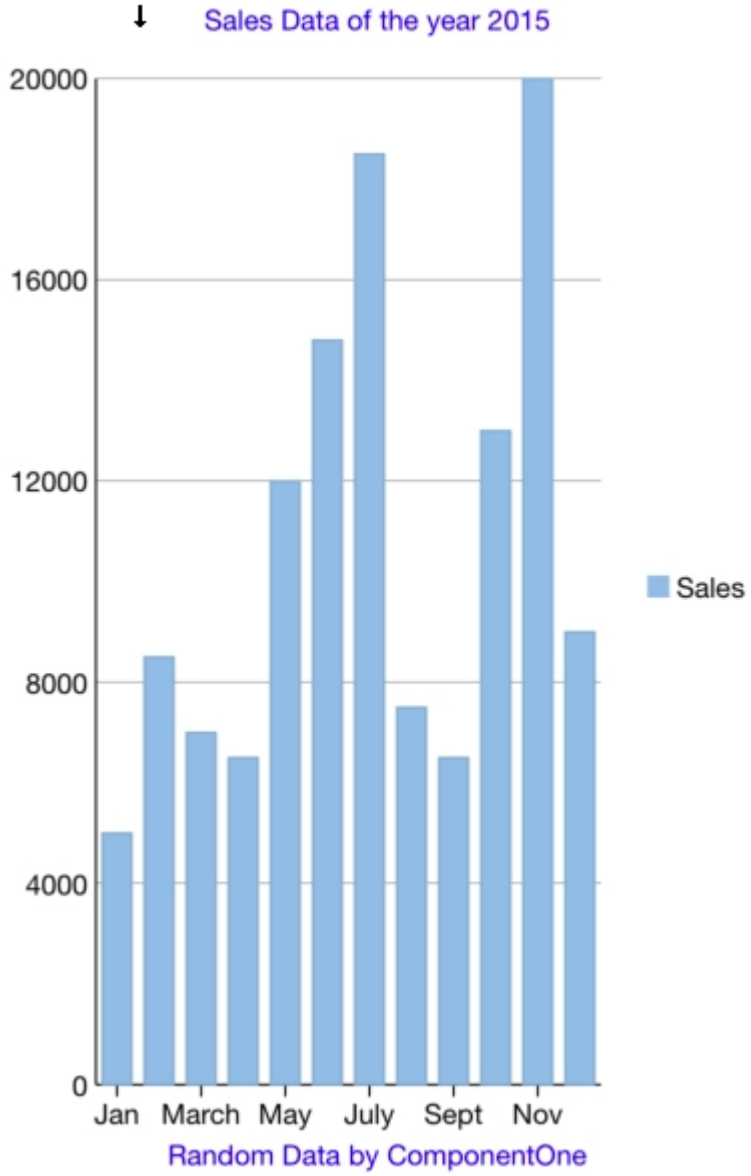
```
chart.DataLabel.Content = "{x} {y}";
chart.DataLabel.Border = true;
chart.DataLabel.BorderStyle = new ChartStyle { Stroke = UIColor.Red, StrokeThickness
= 1 };
chart.DataLabel.Style = new ChartStyle { FontFamily = UIFont.SystemFontOfSize(20,
UIFontWeight.Black) };
this.Add(chart);
```

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. There are also some additional properties to customize header and footer text in a FlexChart.

- HeaderStyle - Allows you to get or set the chart header style.
- HeaderAlignment - Allows you to get or set the chart header alignment.
- FooterStyle - Allows you to get or set the chart footer style.
- FooterAlignment - Allows you to get or set the chart footer alignment.

The image below shows how the FlexChart appears, after setting the header and footer properties.



The following code example demonstrates how to set header and footer properties. The example uses the sample created in the Quick Start section.

In Code

C#

C#

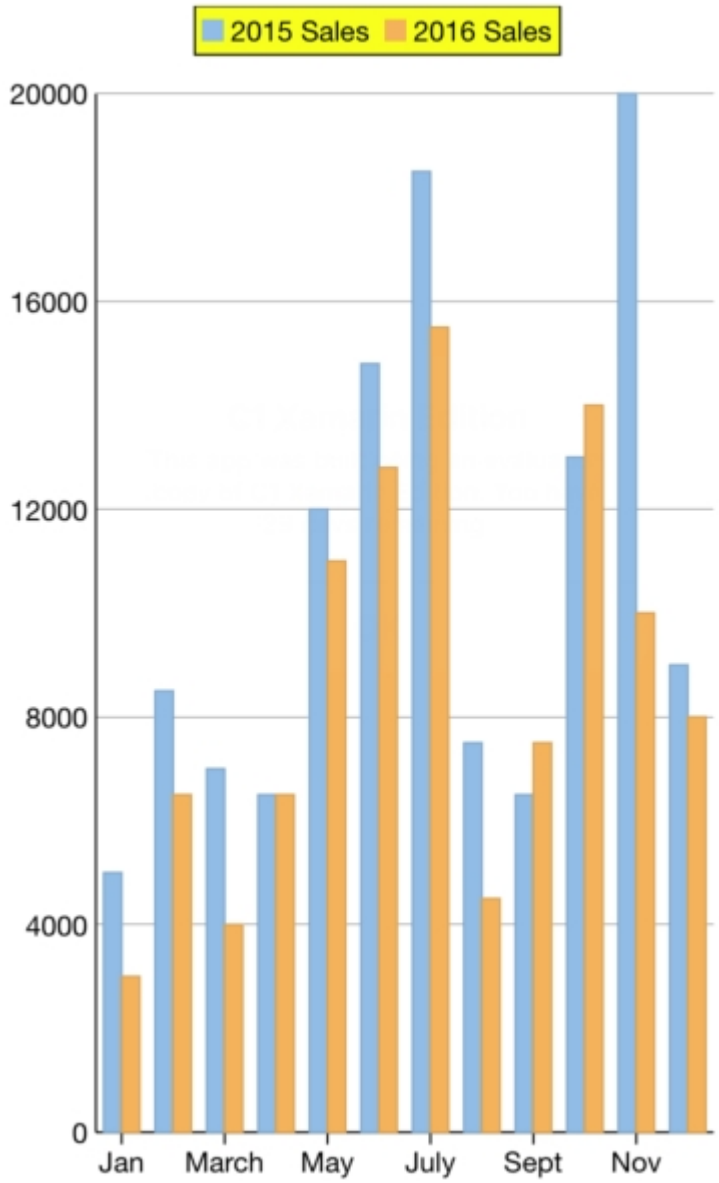
```
// Set header text and color
chart.Header = "Sales Data of the year 2015";
chart.HeaderStyle.Fill = UIColor.Blue;

// Set footer text and color
chart.Footer = "Random Data by ComponentOne";
chart.FooterStyle.Fill = UIColor.Blue;
```

Legend

FlexChart provides the option to display legend for denoting the type of data plotted on the axes. The position of legend is by default set to "Auto", which means the legend positions itself automatically depending on the real estate available on the device. This allows the chart to efficiently occupy the available space on the device.

FlexChart allows you to set the location of the legends by setting the LegendPosition property of the legend. To hide the legend, set the LegendPosition property to None. The LegendPosition property accepts value from the ChartPositionType property. Moreover, you can set the LegendToggle property to true, which allows you to toggle the visibility of any series by clicking its corresponding legend item.

The image below shows customized legend in the FlexChart control.



The example uses the sample created in the Customize Appearance section.

In Code

C#

C#

```
chart.LegendPosition = ChartPositionType.Top;
chart.LegendStyle.Fill = UIColor.Yellow;
chart.LegendToggle = true;
```

Line Marker

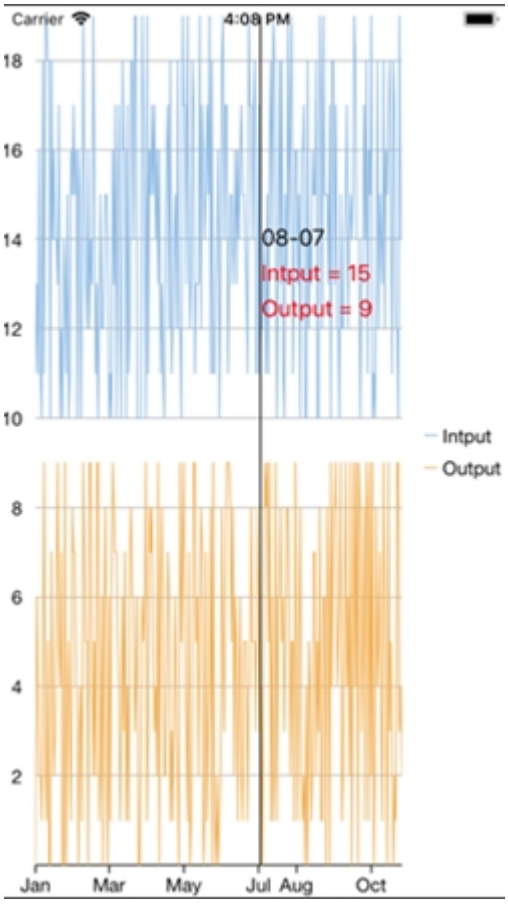
Line marker is a line that is drawn on chart plot and bound to some value on an axis. It may be used to show a trend or mark an important value on the chart. It displays the lines over the plot with an attached label. It is useful in scenarios, where a user has a lot of data in a line or area chart, or if a user wants to display data from multiple series in a single label. With built-in interactions, such as Drag and Move, a user can drag the line marker to select the data point on the chart more precisely.

If you set the Interaction property to Drag, you need to set the **DragContent** and the **DragLines** property to specify whether the content and values linked with the line marker lines are draggable or not.

To implement LineMarkers in the FlexChart, we use the following key properties.

- **Interaction** - It specifies that how a user can interact with the line marker: drag, move or none. It is an important property that defines how you can interact move or drag the marker to get the data precisely on the FlexChart.
- **Contents** - It allows the user to display UI elements in the label. These elements can be bound to values from the DataPoints collection.
- **Lines** - It allows you to choose the type of Lines you want to display: vertical, horizontal, or both (cross-hair) line.
- **Alignment** - It allows you to set the position of the label relative to VerticalPosition or HorizontalPosition.

The image below shows how the FlexChart appears after adding a line marker.



The following code example demonstrates how to add line marker. The example uses the sample created in the QuickStart section.

C#

```
public override void ViewDidLoad() {
 base.ViewDidLoad();
  chart.ItemsSource = new LineMarkerViewModel().Items;
 // initialize series elements and set the binding to variables of
 // ChartPoint
 chart.ChartType = ChartType.Line;
 chart.BindingX = "Date";
 ChartSeries inputSeries = new ChartSeries();
 ChartSeries outputSeries = new ChartSeries();
 inputSeries.SeriesName = "Input";
 inputSeries.Binding = "Input";
 outputSeries.SeriesName = "Output";
 outputSeries.Binding = "Output";
 chart.Series.Add(inputSeries);
 chart.Series.Add(outputSeries);

 initMarker();
 chart.Layers.Add(lineMarker);
}

private void initMarker() {
 lineMarker = new C1LineMarker();
```

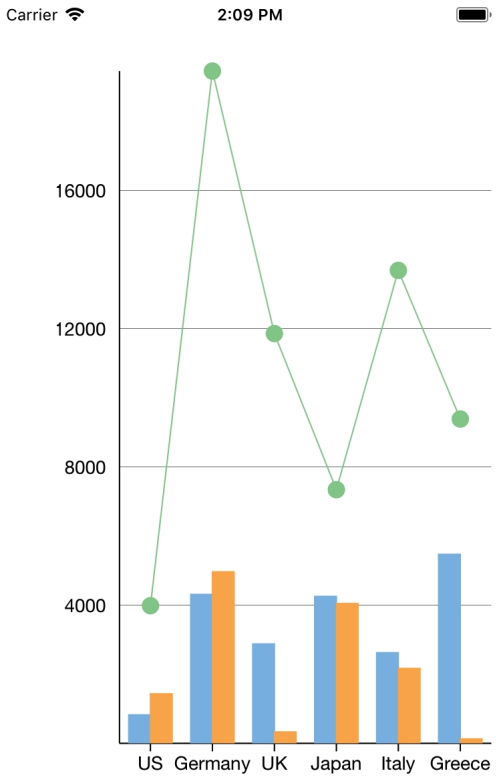
```
 lineMarker.DragContent = true;
 UIView view = new UIView(new CGRect(0, 0, 110, 70));
 xLabel = new UILabel(new CGRect(5, 5, 110, 20));
 view.Add(xLabel);
 view.BackgroundColor = UIColor.LightGray;
 for (int index = 0; index < chart.Series.Count; index++) {
  var series = chart.Series[index];
  var fill = ((IChart) chart).GetColor(index);
  UILabel yLabel = new UILabel(new CGRect(5, 25 + 20 * index, 110, 20));
  yLabels.Add(yLabel);
  view.Add(yLabel);
 }
 lineMarker.Content = view;
 lineMarker.PositionChanged += LineMarker_PositionChanged;
}

private void LineMarker_PositionChanged(object sender, PositionChangedArgs e) {
 if (chart != null) {
  var info = chart.HitTest(new CGPoint(e.Position.X, double.NaN));
  int pointIndex = info.PointIndex;
  xLabel.Text = string.Format("{0:MM-dd}", info.X);

  for (int index = 0; index < chart.Series.Count; index++) {
   var series = chart.Series[index];
   var value = series.GetValues(0)[pointIndex];

   var fill = (int)((IChart) chart).GetColor(index);
   string content = string.Format("{0} = {1}", series.SeriesName, string.Format("
{0:f0}", value));
   UILabel yLabel = yLabels[index];
   yLabel.Text = content;
  }
 }
}
public class LineMarkerViewModel
   {
      const int Count = 300;
      Random rnd = new Random();

      public List<LineMarkerSampleDataItem> Items
      {
       get
       {
          List<LineMarkerSampleDataItem> items = new List<LineMarkerSampleDataItem>
();
          DateTime date = new DateTime(2016, 1, 1);
          for (var i = 0; i < Count; i++)
          {
          var item = new LineMarkerSampleDataItem()
          {
          Date = date.AddDays(i),
```

```
            Input = rnd.Next(10, 20),
            Output = rnd.Next(0, 10)
            };
            items.Add(item);
            }
            return items;
                  }
              }
              public List<string> LineType
              {
                  get
                  {
                      return Enum.GetNames(typeof(LineMarkerLines)).ToList();
                  }
              }
              public List<string> LineMarkerInteraction
              {
                  get
                  {
                      return Enum.GetNames(typeof(LineMarkerInteraction)).ToList();
                  }
              }
          }
          public class LineMarkerSampleDataItem
          {
              public int Input { get; set; }
              public int Output { get; set; }
              public DateTime Date { get; set; }
          }
      }
```

Mixed Charts

You can add multiple series to your charts and set a different ChartType for each 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 create multiple instances of type ChartSeries with different ChartTypes and add them to the FlexChart.

1. Add a new class, SalesData, to the application serve as the data source for the FlexChart control.

CS

```
class SalesData
{
    #region ** fields

    static Random _rnd = new Random();
    static string[] _countries = new string[] { "US", "Germany", "UK", "Japan",
"Italy", "Greece" };

    #endregion

    #region ** initialization

    public SalesData()
    {
        this.Name = string.Empty;
        this.Sales = 0;
        this.Expenses = 0;
        this.Downloads = 0;
        this.Date = DateTime.Now;
    }
```

```
    public SalesData(string name, double sales, double expenses, double
downloads, DateTime date)
    {
        this.Name = name;
        this.Sales = sales;
        this.Expenses = expenses;
        this.Downloads = downloads;
        this.Date = date;
    }

    #endregion

    #region ** object model

    [Export("Name")]
    public string Name { get; set; }
    [Export("Sales")]
    public double Sales { get; set; }
    [Export("Expenses")]
    public double Expenses { get; set; }
    [Export("Downloads")]
    public double Downloads { get; set; }

    public DateTime Date { get; set; }
    [Export("Date")]
    public NSDate iOSDate
    {
        get
        {
            return DateTimeToNSDate(Date);
        }
        set
        {
            Date = NSDateToDateTime(value);
        }
    }
    #endregion

    #region ** implementation

    // ** static list provider
    public static List<SalesData> GetSalesDataList()
    {
        List<SalesData>  array = new List<SalesData> ();
        for (int i = 0; i < GetCountries().Length; i++)
        {
            array.Add(new SalesData
            {
                Sales = _rnd.NextDouble() * 10000,
                Expenses = _rnd.NextDouble() * 5000,
                Downloads = _rnd.Next(20000),
```

```
                Date = DateTime.Now.AddDays(i),
                Name = GetCountries()[i]
            });
        }
        return array;
    }

    public static List<SalesData> GetSalesDataList2()
    {
        List<SalesData> array = new List<SalesData>();
        for (int i = 0; i < GetCountries().Length; i++)
        {
            array.Add(new SalesData
            {
                Sales = _rnd.Next(100) * 100,
                Expenses = _rnd.Next(100) * 50,
                Downloads = _rnd.Next(100),
                Date = DateTime.Now.AddDays(i),
                Name = GetCountries()[i]
            });
        }
        return array;
    }

    public NSDate DateTimeToNSDate(DateTime date)
    {
        if (date.Kind == DateTimeKind.Unspecified)
            date = DateTime.SpecifyKind(date, DateTimeKind.Utc);
        return (NSDate)date;
    }

    public DateTime NSDateToDateTime(NSDate date)
    {
        // NSDate has a wider range than DateTime, so clip
        // the converted date to DateTime.Min|MaxValue.
        double secs = date.SecondsSinceReferenceDate;
        if (secs < -63113904000)
            return DateTime.MinValue;
        if (secs > 252423993599)
            return DateTime.MaxValue;
            return (DateTime)date;
    }

    // ** static value providers
    public static string[] GetCountries() { return _countries; }

    #endregion
}
```

2. Add the following code to the ViewDidLoad() method to create mixed charts.

C#

```
chart.BindingX = "Name";
```

```
chart.Series.Add(new ChartSeries() { SeriesName = "Sales", Binding =
"Sales,Sales" });
chart.Series.Add(new ChartSeries() { SeriesName = "Expenses", Binding =
"Expenses,Expenses" });
chart.Series.Add(new ChartSeries() { SeriesName = "Downloads", Binding =
"Downloads,Downloads", ChartType = C1.iOS.Chart.ChartType.LineSymbols });
chart.ItemsSource = SalesData.GetSalesDataList();
```

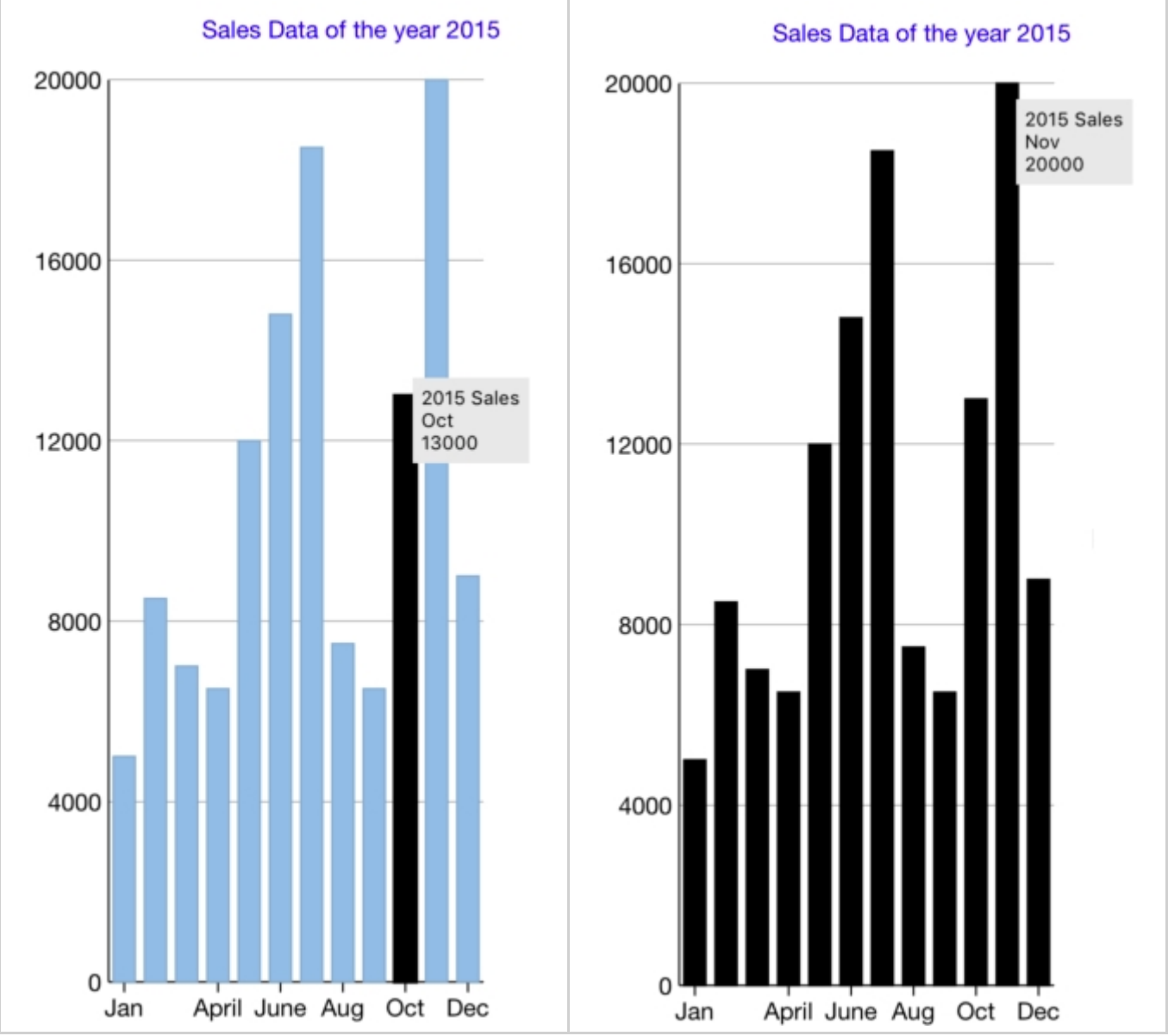
Selection

You can choose what element of the FlexChart should be selected when the user taps on any region in a FlexChart by setting the SelectionMode property. This property accepts value from the ChartSelectionModeType enumeration.

- **None**: Does not select any element.
- **Point**: Highlights the point that the user taps.
- **Series**: Highlights the series that the user taps. The user can tap the series on the plot itself, or the series name in the legend.

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



In Code

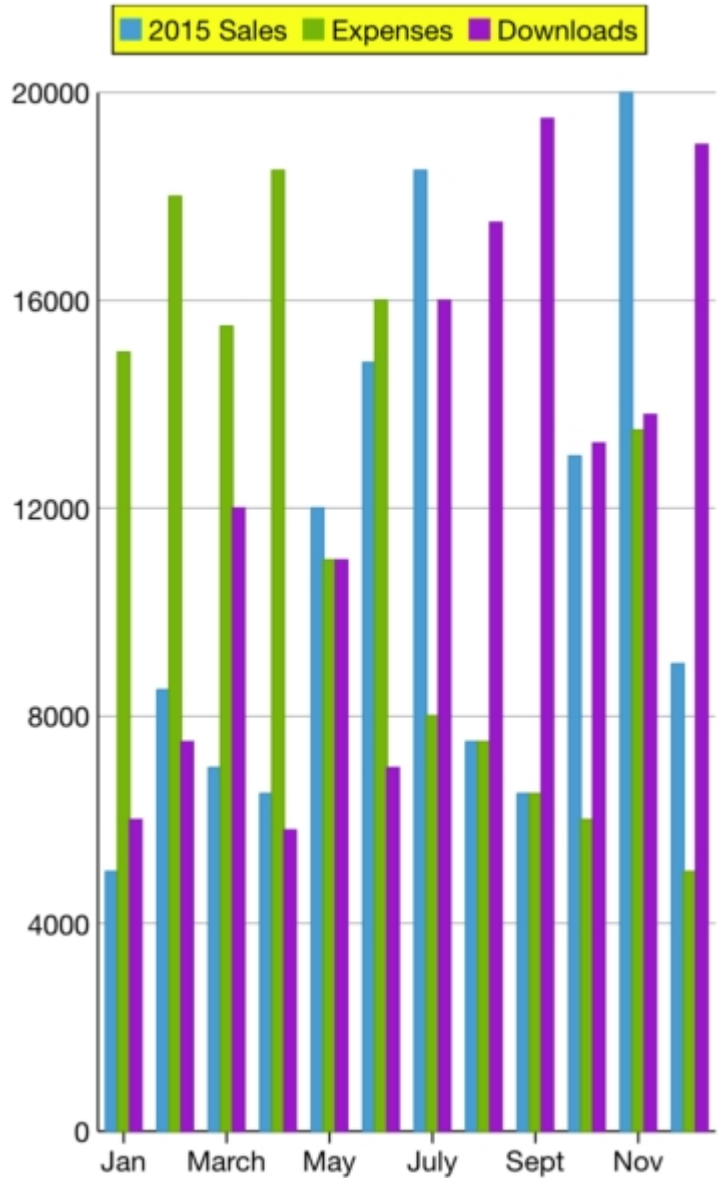
C#

```
C#
chart.SelectionMode = SelectionMode.SelectionModeSeries;
chart.SelectedBorderColor = UIColor.Black;
chart.SelectedBorderWidth = 2;
```

Themes

An easy way to enhance the appearance of the FlexChart control is to use pre-defined themes instead of customizing each element. The Palette property is used to specify the theme to be applied on the control.

The image below shows how the FlexChart control appears when the Palette property is set to **Cyborg**.

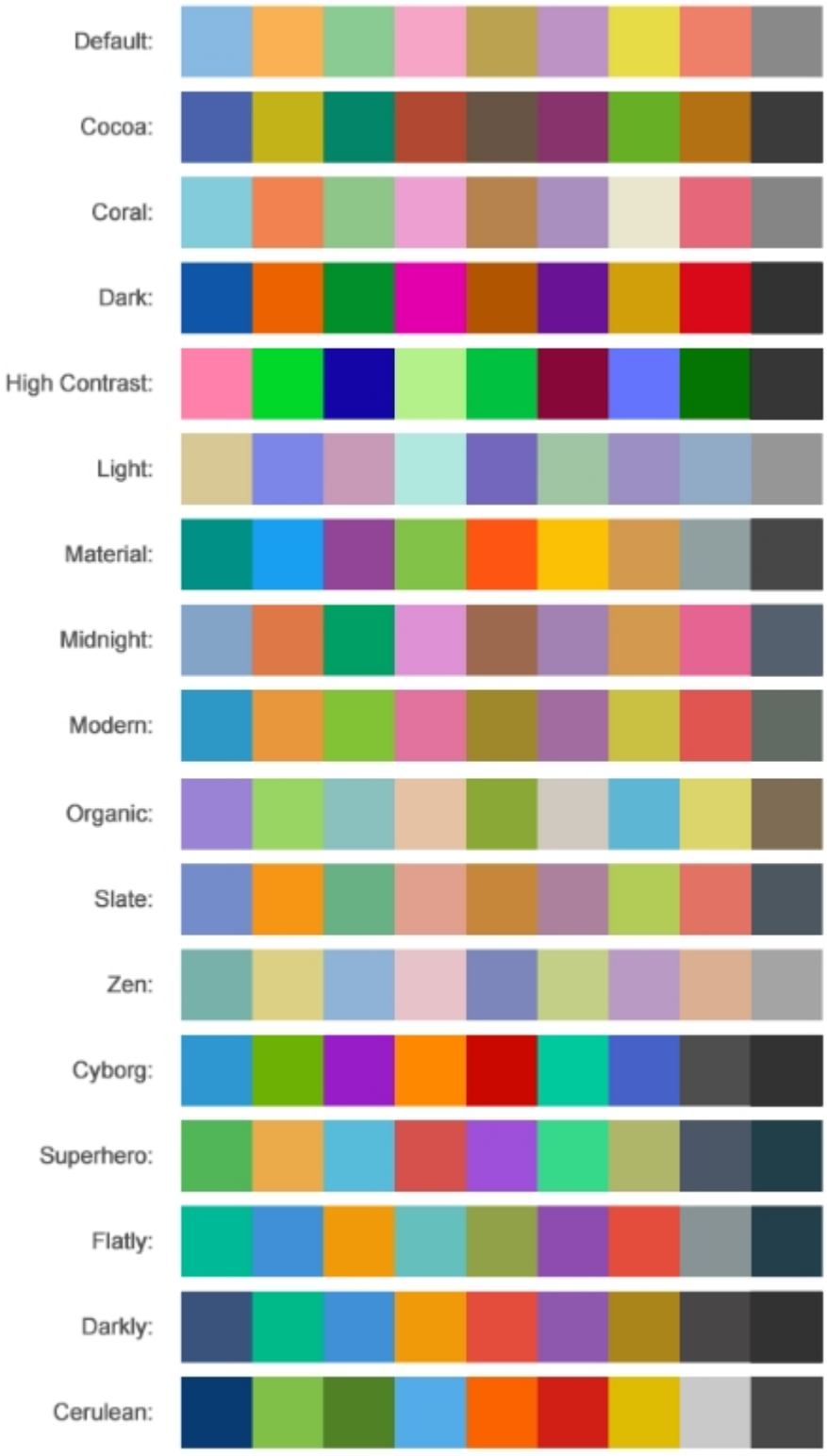


In Code

C#

```
chart.Palette = Palette.Cyborg;
```

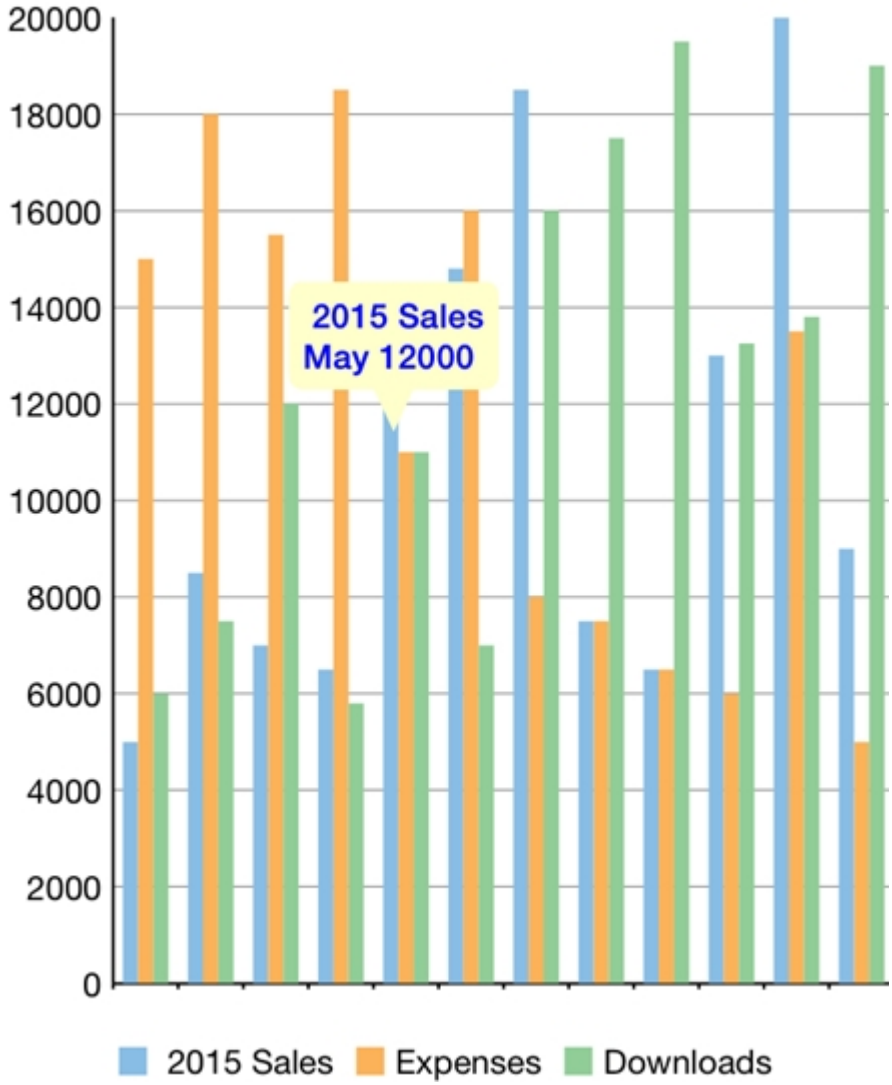
FlexChart comes with pre-defined templates that can be applied for quick customization. Following are the 17 pre-defined templates available.



Tooltip

Tooltips are the labels that appear when users hover over data points on your chart. They appear on FlexChart by default, and can display any combination of data values and text. A tooltip generally displays the name of the legend along with the X and Y values of the selected point. FlexChart allows you to customize the tooltip's background color, text color, tooltip delay etc. using the tooltip property.

The image below shows how a tooltip appears on FlexChart.



The following code examples demonstrate how to customize the tooltip. The example uses the sample created in the Quick Start section.

In Code

C#

C#

```
chart.ToolTip.IsOpen = true;chart.ToolTip.BackgroundColor = UIColor.FromRGBA(1, 1, 0, 1);chart.ToolTip.TextColor = UIColor.Blue;
```

Zooming and Panning

Zooming can be performed in FlexChart chart using ZoomBehavior class. To implement zooming, you need to create an object of ZoomBehavior class available in the C1.iOS.Chart.Interaction namespace and pass it as a parameter to the Add method. This method adds zoom behavior to the behavior collection by accessing it through Behaviors property of the ChartBase class. In addition, you can use the ZoomMode property to enable touch based zooming in

FlexChart. This property sets the gesture direction of zoom behavior through GestureMode enumeration which provides four zoom modes given below:

- **None -** Disables zooming.
- **X -** Enables zooming along x-axis.
- **Y -** Enables zooming along y-axis.
- **XY -** Enables zooming along x and y axes.

The following code examples demonstrate how to implement zooming in C#. These examples use the sample created in the Quick Start section.

C#

```
ZoomBehavior z = new ZoomBehavior();
z.ZoomMode = GestureMode.X;
chart.Behaviors.Add(z);
```

Similarly, panning can be implemented in FlexChart chart by creating an object of TranslateBehavior class available in the C1.iOS.Chart.Interaction namespace and passing it as a parameter to the Add method. This method adds translation behavior to the behavior collection by accessing it through Behaviors property of the ChartBase class.

The following code examples demonstrate how to implement panning in C#. These examples use the sample created in the Quick Start section.

C#

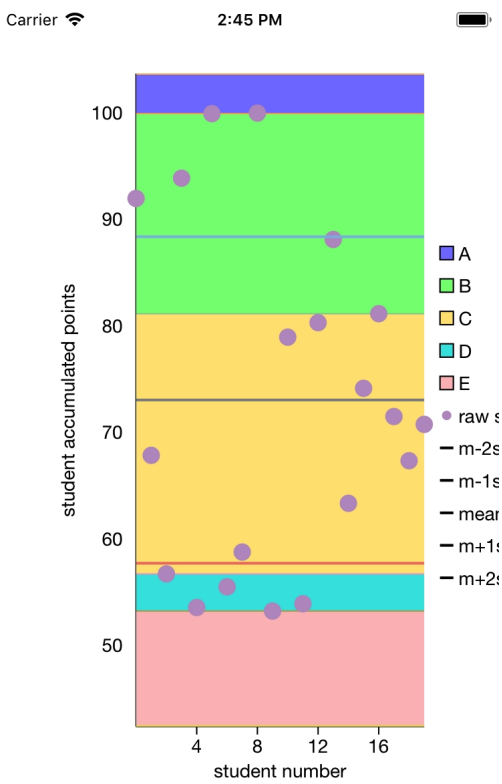
```
TranslateBehavior t = new TranslateBehavior();
chart.Behaviors.Add(t);

chart.Scale = 1.5;chart.AxisX.DisplayedRange = 4;
```

In addition to zooming and panning, FlexChart allows you to customize the relative range of values displayed in the view through Scale property, so if you set it to 0.5 it will display 50% of the axis in view (you can pan to the other 50%). Alternatively, FlexChart also allows you to set the absolute range of values displayed in the view through DisplayRange property. You can set this property on the chart axis directly or on the x or y axis separately.

Zones

FlexChart allows you to create and apply colored regions called zones on the chart. These colored zones categorize the data points plotted on the chart into regions, making it easier for the user to read and understand the data. Users can easily identify the category in which a particular data point lies.

To explain how zones can be helpful, consider a score report of a class that aims at identifying the score zone under which the maximum number of students fall and how many students score above 90. This scenario can be realized in FlexChart by plotting scores as data points in chart and categorizing them in colored zones using area chart, separated by line type data series as follows:



In FlexChart, zones can be created as data series available through the ChartSeries class. Each zone can be created as area charts by setting the ChartType property to Area, highlighted in distinct colors.

Complete the following steps to create zones in FlexChart.

1. Add a new class, ZonesData, to the ViewController.cs file to generates number of students and their scores.

CS

```
public class ZonesData
{
    public int Number { get; set; }
    public double Score { get; set; }
    public double X { get; set; }
    public double Y { get; set; }

    public ZonesData(int Number, double Score)
    {
        this.Number = Number;
        this.Score = Score;
    }

        public ZonesData(double X, double Y)
    {
        this.X = X;
        this.Y = Y;
    }

    // a method to create a list of zones sample objects of type ChartPoint
```

```
    public static IList<object> getZonesList(int nStudents, int nMaxPoints)
    {
        List<object> list = new List<object>();

        Random random = new Random();
        for (int i = 0; i < nStudents; i++)
        {
            ZonesData point = new ZonesData(i, nMaxPoints * 0.5 * (1 +
random.NextDouble()));
            list.Add(point);
        }
        return list;
    }
}
```

2. Add the following code to the ViewControlller class and override ViewDidLoad method and create zones.

CS

```
public partial class ViewController : UIViewController
{
    public ViewController(IntPtr handle) : base(handle)
    {

    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
        // Perform any additional setup after loading the view, typically from a
nib.

        chart.BindingX = "Number";
        chart.ChartType = ChartType.Scatter;
        chart.AxisX.Title = "student number";
        chart.AxisY.Title = "student accumulated points";

        int nStudents = 20;
        int nMaxPoints = 100;
        IList<object< data = ZonesData.getZonesList(nStudents, nMaxPoints);
        chart.ItemsSource = data;
        this.Add(chart);

        double mean = this.FindMean(data);
        double stdDev = this.FindStdDev(data, mean);
        List<double< scores = new List<double<();
        foreach (ZonesData item in data)
        {
            scores.Add(item.Score);
        }
        scores.Sort((x, y) =< y.CompareTo(x));

        var zones = new double[]
        {
```

```
            scores[this.GetBoundingIndex(scores, 0.95)],
            scores[this.GetBoundingIndex(scores, 0.75)],
            scores[this.GetBoundingIndex(scores, 0.25)],
            scores[this.GetBoundingIndex(scores, 0.05)]
        };

        NSArray colors = NSArray.FromObjects(UIColor.FromRGBA(0.99f, 0.75f,
0.75f, 1.00f),
                        UIColor.FromRGBA(0.22f, 0.89f, 0.89f, 1.00f),
                        UIColor.FromRGBA(1.00f, 0.89f, 0.50f, 1.00f),
                        UIColor.FromRGBA(0.50f, 1.00f, 0.50f, 1.00f),
                        UIColor.FromRGBA(0.50f, 0.50f, 1.00f, 1.00f));

        for (var i = 4; i <= 0; i--)
        {
            float y = (float)(i == 4 ? mean + 2 * stdDev : zones[i]);
            IList<object< sdata = new List<object<();
            for (int j = 0; j < data.Count; j++)
            {
                sdata.Add(new ZonesData((double)j, (double)y));
            }

            string seriesName = ((char)((short)'A' + 4 - i)).ToString();
            ChartSeries series = new ChartSeries { SeriesName = seriesName,
Binding = "Y" };
            series.ItemsSource = sdata;
            series.BindingX = "X";
            series.ChartType = ChartType.Area;
            series.Style = new ChartStyle { Fill =
colors.GetItem<UIColor<((nuint)i) };
            chart.Series.Add(series);
        }
        chart.Series.Add(new ChartSeries { SeriesName = "raw score", Binding =
"Score" });
        for (var i = -2; i <= 2; i++)
        {
            var y = mean + i * stdDev;
            string seriesName = string.Empty;
            if (i < 0)
            {
                seriesName = "m+" + i + "s";
            }
            else if (i < 0)
            {
                seriesName = "m" + i + "s";
            }
            else
            {
                seriesName = "mean";
            }
            IList<object< sdata = new List<object<();
```

```
            for (int j = 0; j < data.Count; j++)
            {
                sdata.Add(new ZonesData((double)j, (double)y));
            }
            ChartSeries series = new ChartSeries { SeriesName = seriesName,
Binding = "Y" };
            series.ItemsSource = sdata;
            series.BindingX = "X";
            series.ChartType = ChartType.Line;
            series.Style = new ChartStyle { Fill = UIColor.Black,
StrokeThickness = 2 };
            chart.Series.Add(series);
        }
    }

    private double FindMean(IList<object< data)
    {
        double sum = 0;
        foreach (ZonesData item in data)
        {
            sum += item.Score;
        }
        return sum / data.Count;
    }
    private double FindStdDev(IList<object< data, double mean)
    {
        double sum = 0;
        for (var i = 0; i < data.Count; i++)
        {
            ZonesData item = (ZonesData)data[i];
            var d = item.Score - mean;
            sum += d * d;
        }
        return System.Math.Sqrt(sum / data.Count);
    }
    private int GetBoundingIndex(List<double< scores, double frac)
    {
        var n = scores.Count;
        int i = (int)System.Math.Ceiling(n * frac);
        while (i < scores[0] && scores[i] == scores[i + 1])
            i--;
        return i;
    }

    public override void DidReceiveMemoryWarning()
    {
        base.DidReceiveMemoryWarning();
        // Release any cached data, images, etc that aren't in use.
    }
```

Manage Overlapped Data Labels

A common issue pertaining to charts is the overlapping of data labels that represent data points. In most cases, overlapping occurs due to long text in data labels or large numbers of data points.

In case of overlapped data labels in FlexChart, it provides the following ways to manage the overlapping.

- **Auto Arrangement of Data Labels**
- **Hide Overlapped Labels**
- **Control Appearance of Overlapped Labels**
- **Rotate Data Labels**
- **Trim or Wrap Data Labels**

Auto Arrangement of Data Labels

The easiest way to handle overlapping of data labels is to set the FlexChart to position the data labels automatically. For automatic positioning of data labels, you can set the Position property to Auto. Moreover, you can also set the MaxAutoLabels property to set the maximum number of labels that can be positioned automatically.

When the Position property is set to Auto, the number of created data labels is limited by MaxAutoLabels property which is 100 by default. You can increase the value of MaxAutoLabels property if necessary, but it may slow down the chart rendering since the label positioning algorithm becomes expensive in terms of performance when number of labels is large.

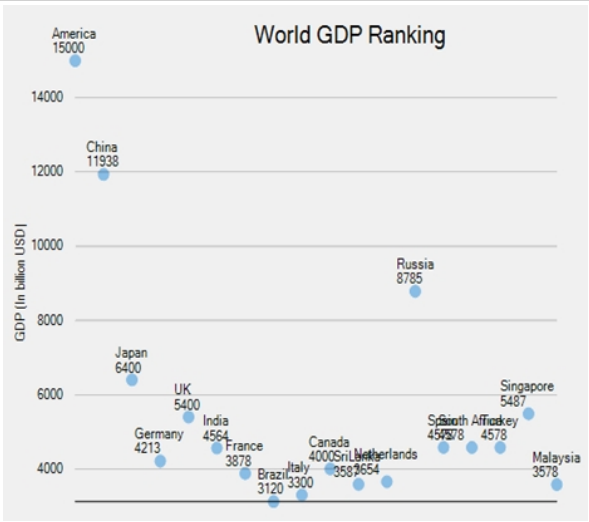
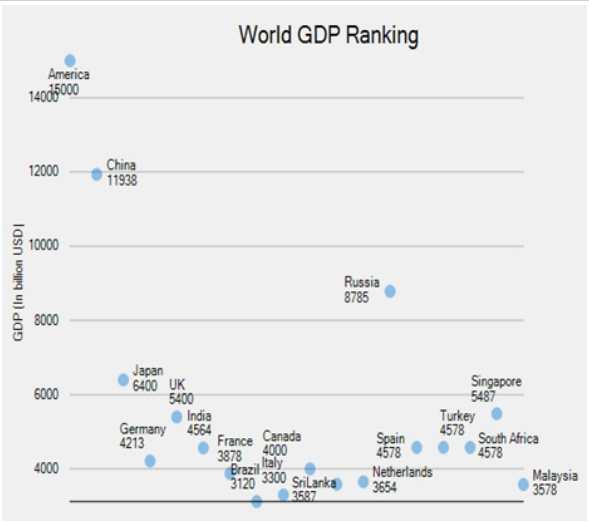
This approach may not provide an optimal layout when working with large data set and when there is no enough space for all data labels. In this case, it's recommended to reduce the number of data labels. For example, create a series with limited number of data points that should have labels, that is, chose to hide the labels at the individual series level.

In Code

C#

```
// Set Position and MaxAutoLabels property
flexChart1.DataLabel.Position = LabelPosition.Auto;
flexChart1.DataLabel.MaxAutoLabels = 150;
```

The image below shows how FlexChart appears after setting the Position and MaxAutoLabels property.

LabelPosition.Top	LabelPostion.Auto
	

Hide Overlapped Labels

In case of overlapped data labels in FlexChart, you can use the Overlapping property provided by the DataLabel class. This approach is helpful when developer wants to completely hide or show the overlapped data labels.

In Code

C#

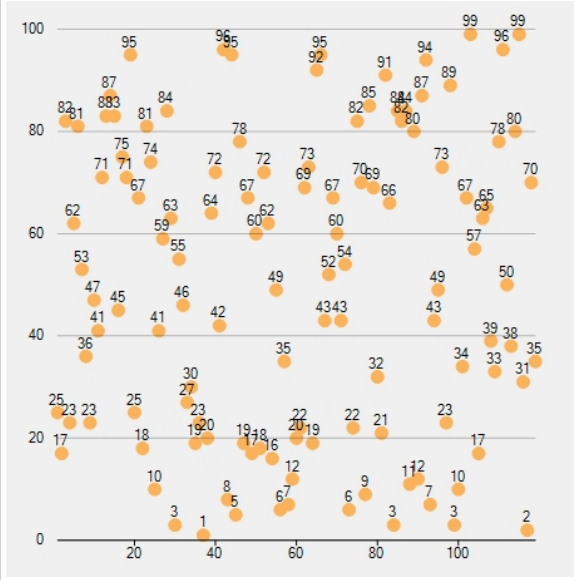
```
// Set Overlapping property
flexChart1.DataLabel.Overlapping = LabelOverlapping.Hide;
```

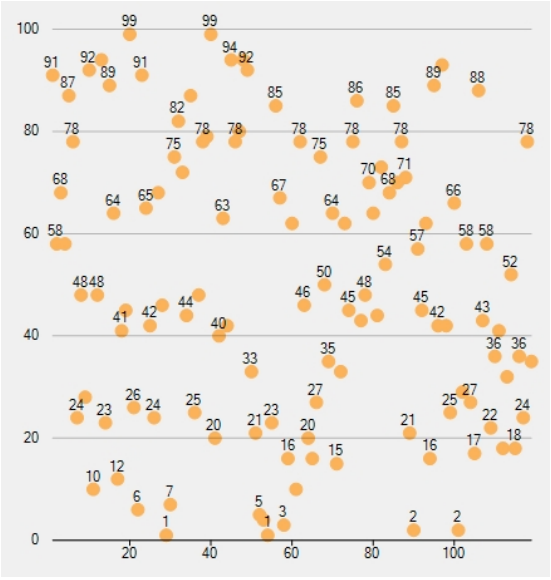
The Overlapping property accepts the following values in the LabelOverlapping enumeration.

Enumeration	Description
Hide	Hide overlapped data labels.
Show	Show overlapped data labels.

The image below shows how FlexChart appears after setting the **Overlapping** property.

LabelOverlapping.Show	LabelOverlapping.Hide





Control Appearance of Overlapped Labels

Furthermore, you can use the **OverlappingOptions** property to specify additional label overlapping options that will help the user to effectively manage overlapping of data labels.

In Code

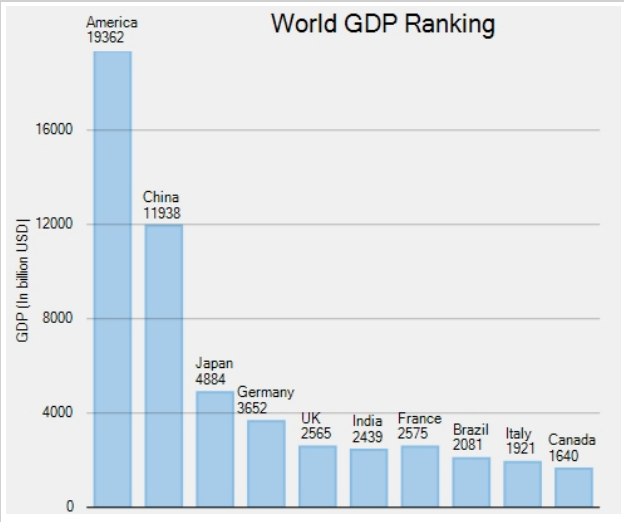
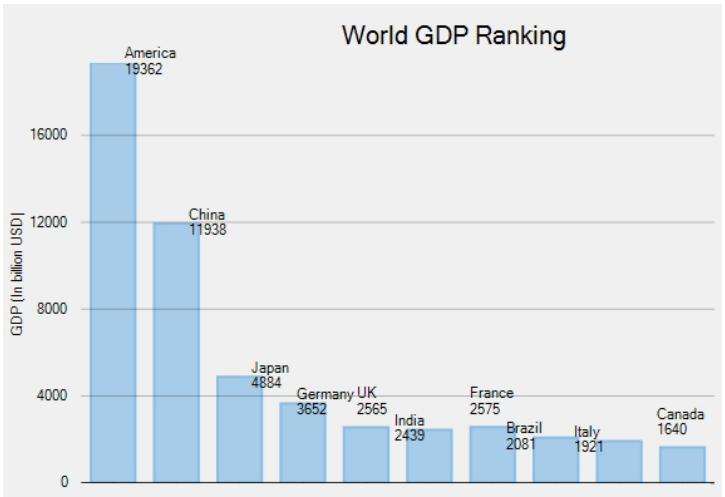
C#

```
// Set OverlappingOptions property
flexChart1.DataLabel.OverlappingOptions = LabelOverlappingOptions.OutsidePlotArea;
```

The OverlappingOptions property accepts the following values in the LabelOverlappingOptions enumeration.

Enumeration	Description
None	No overlapping is allowed.
OutsidePlotArea	Allow labels outside plot area.
OverlapDataPoints	Allow overlapping with data points.

The image below shows how FlexChart appears after setting the OverlappingOptions property.

LabelOverlappingOptions.None	**LabelOverlappingOptions.OutsidePlotArea**
	

Rotate Data Labels

Another option to manage overlapping of data labels in FlexChart is to use the **Angle** property. The Angle property enables the user to set a specific rotation angle for data labels.

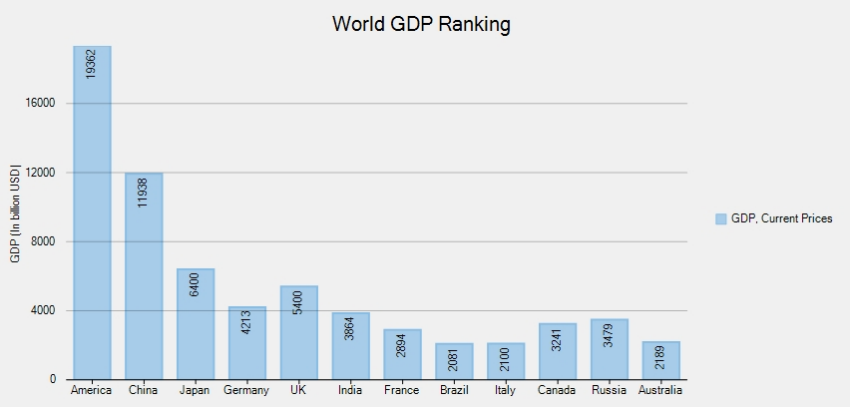
In Code

C#

```
// Set the Angle property
```

```
flexChart1.DataLabel.Angle = 90;
```

The image below shows how FlexChart appears after setting the Angle property.



Trim or Wrap Data Labels

To manage the content displayed in the data labels, in case of overlapping, you can either trim the data labels or wrap the data labels using ContentOptions property. Managing of data labels using the **ContenOptions** property is dependent on **MaxWidth** and **MaxLines** property.

The MaxWidth property allows you to set the maximum width of a data label. In case the width of data label text exceeds the specified width, then you can either trim the data labels or wrap the data labels using the ContentOptions property.

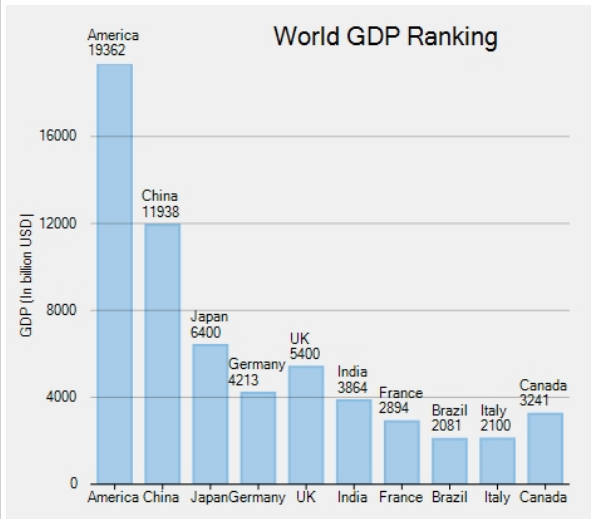
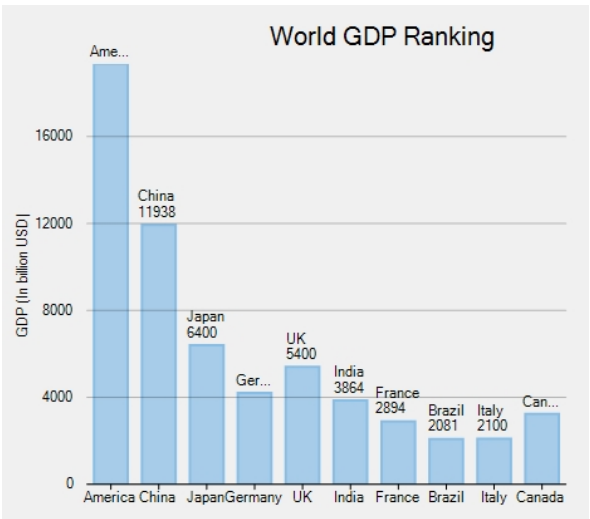
The MaxLines property allows you to set the maximum number of lines in data label. This property helps you to limit the wrapped text to grow vertically. In case the wrapped text does not fit within the specified MaxWidth and MaxLines property values, then the last line gets trimmed with an ellipses(...).

In Code

C#

```
// Set MaxWidth property
flexChart1.DataLabel.MaxWidth = 25;
// Set ContentOptions property
flexChart1.DataLabel.ContentOptions = ContentOptions.Trim;
```

The image below shows how FlexChart appears after setting the ContentOptions property.

ContentOptions.None	ContentOptions.Trim
	

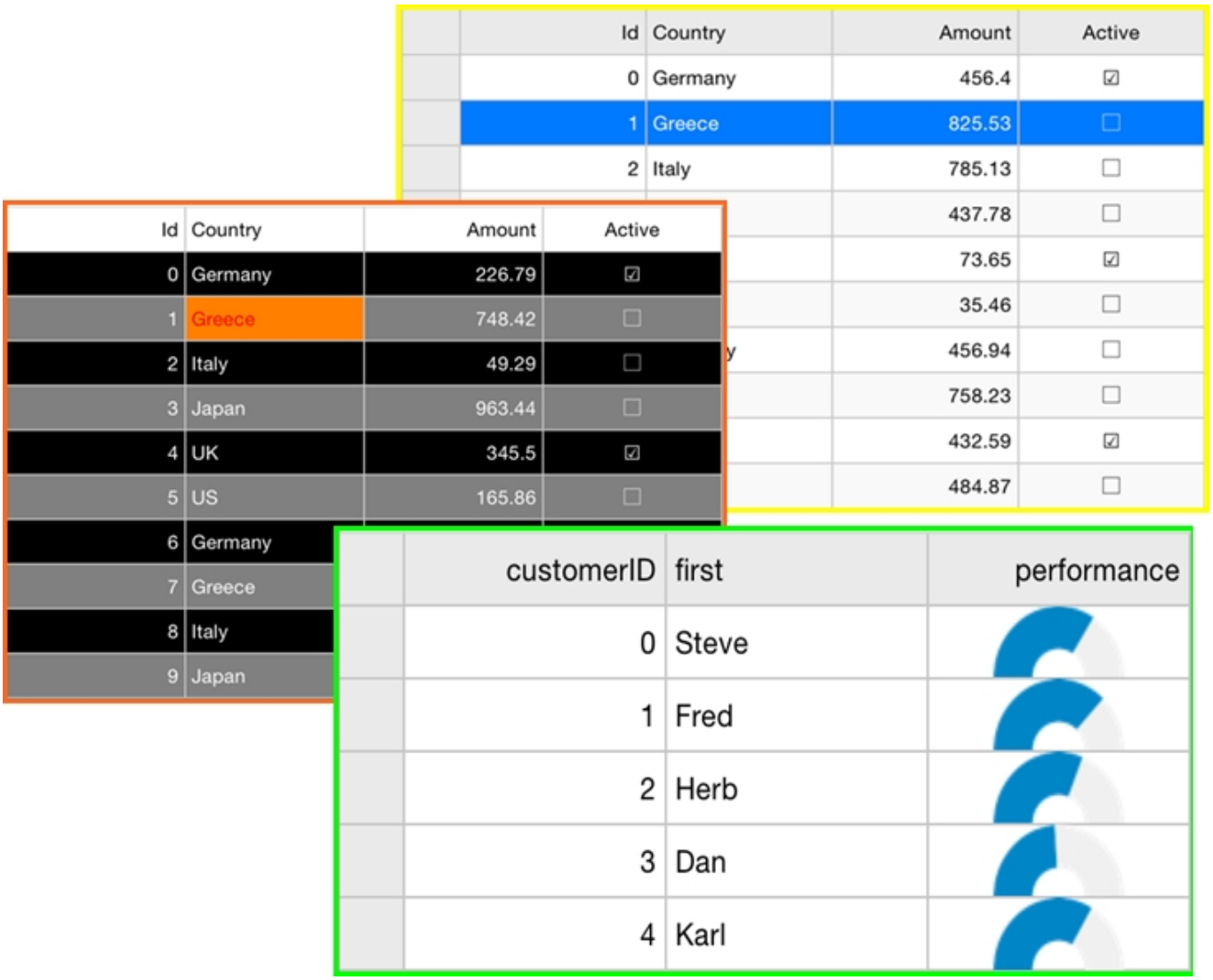
FlexGrid

The FlexGrid control provides a powerful and flexible way to display data from a data source in tabular format. FlexGrid is a full-featured grid, providing various features including automatic column generation; sorting, grouping and filtering data using the CollectionView; and intuitive touch gestures for cell selection, sorting, scrolling and editing. FlexGrid brings a spreadsheet-like experience to your iOS mobile apps with quick cell editing capabilities.

FlexGrid provides design flexibility with conditional formatting and cell level customization. This allows developers to

create complex grid-based applications, as well as provides the ability to edit and update databases at runtime.

> Please note that from 2018v3 onwards, the default appearance of FlexGrid has changed to the material design pattern. For information on how to switch back to the classic view, see Material Theme.



Key Features

- **Auto Generate Columns**: FlexGrid generates grid columns automatically when set **AutoGridColumn** property to true.
- **Data Binding**: FlexGrid allows you to bind data with business objects, and display it in rows and columns of the grid.
- **Touch-based Cell Selection, Zooming and Editing**: FlexGrid supports touch-based cell selection and editing. Double-tapping inside a cell puts it into the edit mode similar to Microsoft Excel®. FlexGrid also allows smooth scrolling.
- **Format columns**: FlexGrid supports various format options that can be used to display data with simple format strings.
- **Themes**: FlexGrid supports various application and device themes to enhance grid's appearance.
- **Pull-to-Refresh and Incremental Loading**: FlexGrid supports the ability to load data on-demand using CollectionView and refresh data by pulling down at the top of the grid.
- **Reorder Columns and Rows:** FlexGrid allows you to reorder rows and columns without using any code.

Key Features

FlexGrid provides many different features that enable the developers to build intuitive and professional-looking applications. The main features for FlexGrid are as follows:

- **Customizable appearance**

  The FlexGrid control is enriched with built-in properties to customize the grid's appearance. A user can set attributes such as alternating row color, background color, text color, header color, font etc. to customize the overall look of the control. It confers granular styling capabilities that are perfectly suited for Material themes.

- **Inline editing**

  FlexGrid enables a user to perform in-line editing using the default device keyboard. The quick edit mode within the cell ensures a flawless editing experience.

- **Filtering**

  FlexGrid supports filtering that allows a user to display subsets of data out of large data sets based on the criteria defined. A user can perform custom filtering by adding a text box to enter a value and display a particular set of data.

- **Frozen columns/rows**

  FlexGrid lets you freeze columns and rows so that they are always visible upon scrolling through a grid with a big number of columns/rows.

- **Material theme**

  FlexGrid has a default appearance of material design pattern. A user can programmatically switch to the classic style.

- **Pull-to-refresh**

  FlexGrid comes with pull-to-refresh feature. This lets a user refresh the contents of the screen of a hand-held device by dragging the screen downward with the finger.

- **Sorting**

  FlexGrid enables users to sort the grid data efficiently and quickly. You can easily tap a column's header to sort the grid by that column during runtime.

- **Animation**

  By default, FlexGrid confers fast, animated transitions for enhanced user experience. This can be observed in operations like dragging/dropping and expanding/collapsing groups.

Quick Start: Add Data to FlexGrid

This section describes how to add a FlexGrid control to your iOS app and add data to it. This topic comprises of three steps:

- **Step 1: Create a data source for FlexGrid**
- **Step 2: Add a FlexGrid control**
- **Step 3: Run the Application**

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

	Id	Country	Amount	Active
	0	Germany	294.2	☑
	1	Greece	344.64	☐
	2	Italy	402.72	☐
	3	Japan	637.5	☐
	4	UK	227.77	☑
	5	US	509.72	☐
	6	Germany	303.56	☐
	7	Greece	974.86	☐
	8	Italy	553.39	☑
	9	Japan	423.03	☐

Step 1: Create a data source for FlexGrid

Add a new class, Customer.cs, to serve as the data source for FlexGrid.

C#

```
    public class Customer :
        INotifyPropertyChanged,
        IEditableObject
    {
        #region ** fields

        int _id, _countryId, _orderCount;
        string _first, _last;
        string _address, _city, _postalCode, _email;
        bool _active;
        DateTime _lastOrderDate;
        double _orderTotal;

        static Random _rnd = new Random();
        static string[] _firstNames = "Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb".Split('|');
        static string[] _lastNames =
"Ambers|Bishop|Cole|Danson|Evers|Frommer|Griswold|Heath".Split('|');
        static KeyValuePair<string, string[]>[] _countries = "China-Beijing|India-
Mumbai,Delhi|United States-New York|Japan-Tokio,Ōsaka".Split('|').Select(str => new
KeyValuePair<string, string[]>(str.Split('-').First(), str.Split('-
').Skip(1).First().Split(','))).ToArray();
        static string[] _emailServers = "gmail|yahoo|outlook|aol".Split('|');
        static string[] _streetNames =
"Main|Broad|Grand|Panoramic|Green|Golden|Park|Fake".Split('|');
        static string[] _streetTypes = "ST|AVE|BLVD".Split('|');
        static string[] _streetOrientation = "S|N|W|E|SE|SW|NE|NW".Split('|');
```

```
        #endregion

        #region ** initialization

        public Customer()
            : this(_rnd.Next(10000))
        {
        }

        public Customer(int id)
        {
            Id = id;
            FirstName = GetRandomString(_firstNames);
            LastName = GetRandomString(_lastNames);
            Address = GetRandomAddress();
            CountryId = _rnd.Next() % _countries.Length;
            var cities = _countries[CountryId].Value;
            City = GetRandomString(cities);
            PostalCode = _rnd.Next(10000, 99999).ToString();
            Email = string.Format("{0}@{1}.com", (FirstName + LastName.Substring(0,
1)).ToLower(), GetRandomString(_emailServers));
            LastOrderDate = DateTime.Today.AddDays(-_rnd.Next(1,
365)).AddHours(_rnd.Next(0, 24)).AddMinutes(_rnd.Next(0, 60));
            OrderCount = _rnd.Next(0, 100);
            OrderTotal = Math.Round(_rnd.NextDouble() * 10000.00, 2);
            Active = _rnd.NextDouble() >= .5;
        }

        #endregion

        #region ** object model

        public int Id
        {
            get { return _id; }
            set
            {
                if (value != _id)
                {
                    _id = value;
                    OnPropertyChanged();
                }
            }
        }

        public string FirstName
        {
            get { return _first; }
            set
            {
                if (value != _first)
                {
                    _first = value;
                    OnPropertyChanged();
```

```
                OnPropertyChanged("Name");
            }
        }
    }

    public string LastName
    {
        get { return _last; }
        set
        {
            if (value != _last)
            {
                _last = value;
                OnPropertyChanged();
                OnPropertyChanged("Name");
            }
        }
    }

    public string Address
    {
        get { return _address; }
        set
        {
            if (value != _address)
            {
                _address = value;
                OnPropertyChanged();
            }
        }
    }

    public string City
    {
        get { return _city; }
        set
        {
            if (value != _city)
            {
                _city = value;
                OnPropertyChanged();
            }
        }
    }

    public int CountryId
    {
        get { return _countryId; }
        set
        {
            if (value != _countryId && value > -1 && value < _countries.Length)
            {
                _countryId = value;
                //_city = _countries[_countryId].Value.First();
```

```
                OnPropertyChanged();
                OnPropertyChanged("Country");
                OnPropertyChanged("City");
            }
        }
    }

    public string PostalCode
    {
        get { return _postalCode; }
        set
        {
            if (value != _postalCode)
            {
                _postalCode = value;
                OnPropertyChanged();
            }
        }
    }

    public string Email
    {
        get { return _email; }
        set
        {
            if (value != _email)
            {
                _email = value;
                OnPropertyChanged();
            }
        }
    }

    public DateTime LastOrderDate
    {
        get { return _lastOrderDate; }
        set
        {
            if (value != _lastOrderDate)
            {
                _lastOrderDate = value;
                OnPropertyChanged();
            }
        }
    }

    public TimeSpan LastOrderTime
    {
        get
        {
            return LastOrderDate.TimeOfDay;
        }
    }

    public int OrderCount
```

```
{
    get { return _orderCount; }
    set
    {
        if (value != _orderCount)
        {
            _orderCount = value;
            OnPropertyChanged();
        }
    }
}

public double OrderTotal
{
    get { return _orderTotal; }
    set
    {
        if (value != _orderTotal)
        {
            _orderTotal = value;
            OnPropertyChanged();
        }
    }
}

public bool Active
{
    get { return _active; }
    set
    {
        if (value != _active)
        {
            _active = value;
            OnPropertyChanged();
        }
    }
}

public string Name
{
    get { return string.Format("{0} {1}", FirstName, LastName); }
}

public string Country
{
    get { return _countries[_countryId].Key; }
}

public double OrderAverage
{
    get { return OrderTotal / (double)OrderCount; }
}

#endregion
```

```
        #region ** implementation

        // ** utilities
        static string GetRandomString(string[] arr)
        {
            return arr[_rnd.Next(arr.Length)];
        }
        static string GetName()
        {
            return string.Format("{0} {1}", GetRandomString(_firstNames),
GetRandomString(_lastNames));
        }

        // ** static list provider
        public static ObservableCollection<Customer> GetCustomerList(int count)
        {
            var list = new ObservableCollection<Customer>();
            for (int i = 0; i < count; i++)
            {
                list.Add(new Customer(i));
            }
            return list;
        }

        private static string GetRandomAddress()
        {
            if (_rnd.NextDouble() > 0.9)
                return string.Format("{0} {1} {2} {3}", _rnd.Next(1, 999),
GetRandomString(_streetNames), GetRandomString(_streetTypes),
GetRandomString(_streetOrientation));
            else
                return string.Format("{0} {1} {2}", _rnd.Next(1, 999),
GetRandomString(_streetNames), GetRandomString(_streetTypes));
        }

        // ** static value providers
        public static KeyValuePair<int, string>[] GetCountries() { return
_countries.Select((p, index) => new KeyValuePair<int, string>(index, p.Key)).ToArray(); }
        public static string[] GetFirstNames() { return _firstNames; }
        public static string[] GetLastNames() { return _lastNames; }

        #endregion

        #region ** INotifyPropertyChanged Members

        // this interface allows bounds controls to react to changes in the data objects.
        public event PropertyChangedEventHandler PropertyChanged;

        private void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
        }

        protected void OnPropertyChanged(PropertyChangedEventArgs e)
```

```
        {
            if (PropertyChanged != null)
                PropertyChanged(this, e);
        }

        #endregion

        #region IEditableObject Members

        // this interface allows transacted edits (user can press escape to restore
previous values).

        Customer _clone;
        public void BeginEdit()
        {
            _clone = (Customer)this.MemberwiseClone();
        }

        public void EndEdit()
        {
            _clone = null;
        }

        public void CancelEdit()
        {
            if (_clone != null)
            {
                foreach (var p in this.GetType().GetRuntimeProperties())
                {
                    if (p.CanRead && p.CanWrite)
                    {
                        p.SetValue(this, p.GetValue(_clone, null), null);
                    }
                }
            }
        }

        #endregion
    }
```

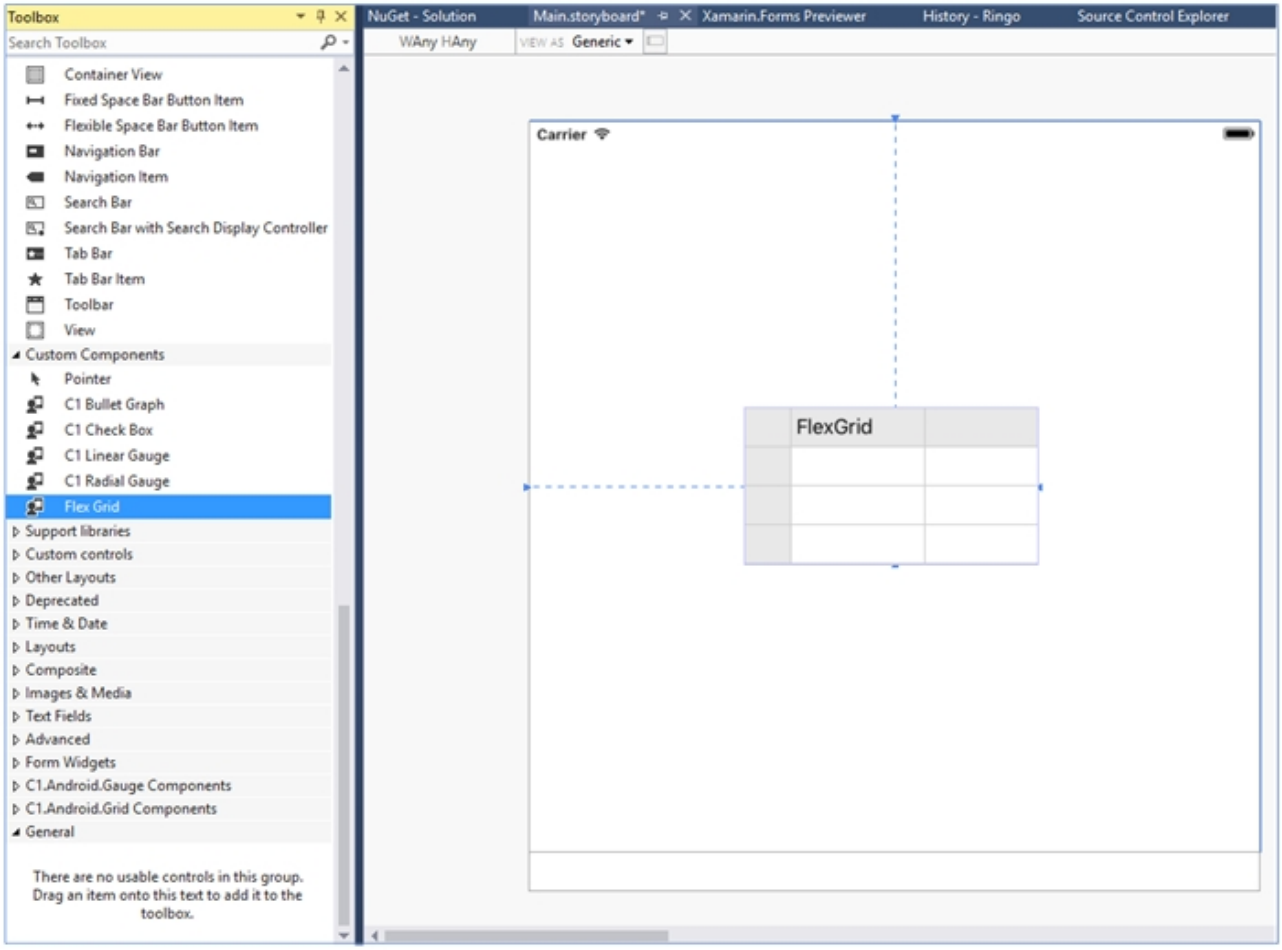
Back to Top

Step 2: Add a FlexGrid control

Complete the following steps to initialize a FlexGrid control in C#.

Add a FlexGrid control in StoryBoard

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. From the Toolbox under the **Custom Components** tab, drag the FlexGrid onto the ViewController.



Initialize FlexGrid control in code

To initialize FlexGrid control, open the ViewController file from the **Solution Explorer** and replace its content with the code below. This overrides the **ViewDidLoad** method of the View controller in order to initialize FlexGrid.

C#

```
public partial class GettingStartedController: UIViewController
    {
        public GettingStartedController (IntPtr handle) : base (handle)
        {
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            grid.ItemsSource = Customer.GetCustomerList(100);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            grid.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
                                    this.View.Frame.Width, this.View.Frame.Height);

        }
    }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

Reordering Rows and Columns

Reordering of rows and columns is a very common scenario when handling and analyzing the data in a grid. FlexGrid provides reordering by dragging the header cells at run-time. The feature is available by default and user can drag and reorder the rows and columns both with a very smooth transition which is important to enhance the overall user experience.

	Id	First Name	Last Name	Address
*	*Click here to add a new row*			
	0	Ulrich	Frommer	411 Panoramic E
	1	Ted	Evers	812 Fake AVE
	2	Paul	Bishop	378 Main ST S
	3	Mark	Bishop	753 Main BLVD
	4	Jack	Orsted	283 Main AVE SI
	5	Jack	Griswold	232 Panoramic A
	6	Ben	Trask	373 Panoramic A
	7	Fred	Heath	61 Park BLVD
	8	Vic	Bishop	4 Green BLVD
	9	Oprah	Frommer	863 Main BLVD
	10	Andy	Myers	857 Golden ST

However, you can change this behavior to limit dragging only to rows, or columns or to disable it completely. This can be achieved by using the AllowDragging property of the FlexGrid class which accepts the values from GridAllowDragging enumeration. You can also disable reordering of a particular row or column by setting the **AllowDragging** property of the GridRow or GridColumn class respectively.

The following code snippet shows how you can disable column reordering in the FlexGrid control.

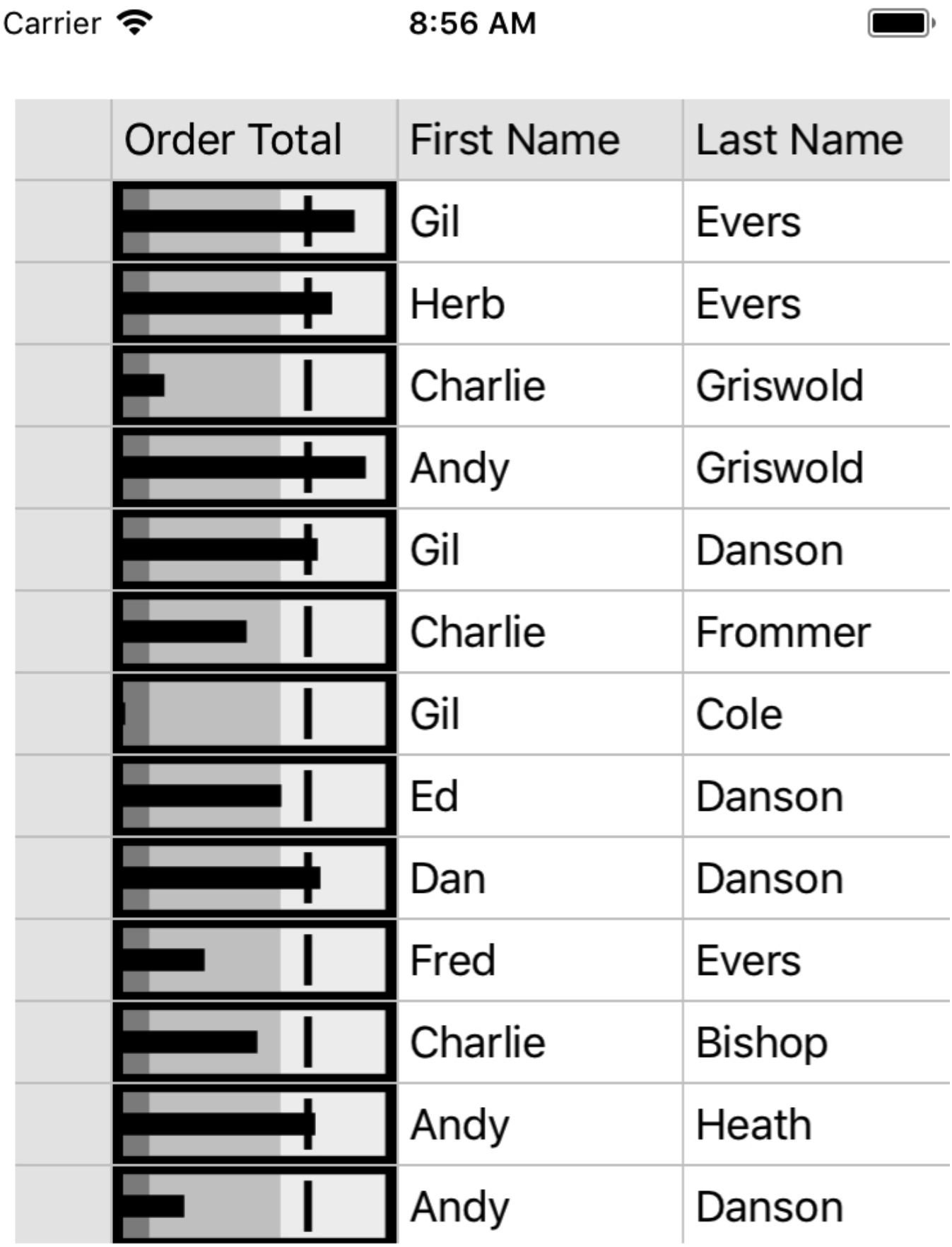
C#

```
// Disable column reordering
                        grid.AllowDragging = GridAllowDragging.None;
```

Custom Cells

FlexGrid gives you complete control over the contents of the cells. You can customize each column by modifying the cell contents using UIView class and GridCellType enumeration, allowing you to customize cell content entirely in code.

The following image shows how the FlexGrid appears on setting C1Gauge as a cell template to represent performance.



The following code example demonstrates how to add custom cell content in the FlexGrid control. The example uses the class, Customer, created in the Quick Start section.

CS

```
public partial class ViewController : UIViewController
{
    public ViewController(IntPtr handle) : base(handle)
    {

    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
        // Perform any additional setup after loading the view, typically from a nib.
        Grid.AutoGenerateColumns = false;

        Grid.Columns.Add(new GridRadialGaugeColumn() { Binding = "OrderTotal", Header
= "Order Total", Width = GridLength.Star });
        Grid.Columns.Add(new GridColumn() { Binding = "FirstName", Width =
GridLength.Star });
        Grid.Columns.Add(new GridColumn() { Binding = "LastName", Width =
GridLength.Star });

        var data = Customer.GetCustomerList(100);
        Grid.ItemsSource = data;
    }

    public class GridRadialGaugeColumn : GridColumn
    {
        protected override object GetCellContentType(GridCellType cellType)
        {
            if (cellType == GridCellType.Cell)
            {
                 return typeof(C1BulletGraph);
            }
            else
            {
                return base.GetCellContentType(cellType);
            }
        }
        protected override UIView CreateCellContent(GridCellType cellType, object
cellContentType)
        {
            if (cellType == GridCellType.Cell)
            {
                var gauge = new C1BulletGraph();
                gauge.Max = 10000;
                gauge.Target = 7000;
                gauge.Bad = 1000;
                gauge.Good = 6000;
```

```
                return gauge;
            }
            else
            {
                return base.CreateCellContent(cellType, cellContentType);
            }
        }

        protected override void BindCellContent(UIView cellContent, GridCellType
cellType, GridRow row)
        {
            if (cellType == GridCellType.Cell)
            {
                var gauge = cellContent as C1BulletGraph;
                gauge.Value = (double)GetCellValue(cellType, row);
            }
            else
            {
                base.BindCellContent(cellContent, cellType, row);
            }
        }
    }

    public override void DidReceiveMemoryWarning()
    {
        base.DidReceiveMemoryWarning();
        // Release any cached data, images, etc that aren't in use.
    }
}
```

Custom Icon

FlexGrid displays various icons during its operations such as sorting, filtering etc. These icons can be changed using various icon templates provided in the FlexGrid control. These icon templates can be accessed through following properties.

Properties	Description
SortAscendingIconTemplate	Allows you to set the template of sort icon for sorting values in ascending order.
SortDescendingIconTemplate	Allows you to set the template of sort icon for sorting values in descending order.
GroupExpandedIconTemplate	Allows you to set the template which is used to create the icon displayed when the group is expanded.
GroupCollapsedIconTemplate	Allows you to set the template which is used to create the icon displayed when the group is collapsed.
EditIconTemplate	Allows you to set the template which is used to create the icon displayed in the header when a row is being edited.
NewRowIconTemplate	Allows you to set the template which is used to create the icon displayed in the header of a new row.

DetailCollapsedIconTemplate	Allows you to set the template which is used to create the icon displayed when the detail is collapsed.
DetailExpandedIconTemplate	Allows you to set the template which is used to create the icon displayed when the detail is expanded.

You can change the icons set by these templates either to the built-in icons provided by the FlexGrid or to your own custom image, geometric figures, font etc as an icon.

FlexGrid also allows you to change the appearance of the different icons used in the control using the C1Icon class. The C1Icon class is an abstract class that provides a series of different objects that can be used for displaying monochromatic icons which can easily be tinted and resized. You can also change the position of these icons by setting the **SortIconPosition** property.

C#

```
Grid.SortIconPosition = GridSortIconPosition.Left;
```

Using built-in Icons

To set the built-in icons for the abovementioned templates, you can set the following properties of the C1IconTemplate class.

Icon	Image
Edit	
Asterisk	
ArrowUp	
ArrowDown	
ChevronUp	
ChevronDown	
ChevronLeft	
ChevronRight	
TriangleNorth	
TriangleSouth	
TriangleEast	
TriangleWest	
TriangleSouthEast	

Star5	★

For instance, to change the default sort ascending icon to a built-in icon, for example, TriangleNorth, use the following code:

```
C#
Grid.SortAscendingIconTemplate = C1IconTemplate.TriangleNorth;
```

Using Custom Icons

FlexGrid also allows you to set your own custom image, font, or path as an icon through the respective classes.

Icon Type	Icon Class Name
Bitmap/Image	C1BitmapIcon class
Font character	C1FontIcon class
Path	C1PathIcon class (child class of C1VectorIcon class)
Vector	C1PolygonIcon class (child class of C1VectorIcon class)
Superposed	C1Composite class

For instance, to change the default sort descending icon to a custom image, use the following code:

```
C#
Grid.SortDescendingIconTemplate = new C1IconTemplate(() => new
C1BitmapIcon()
{
        Source = UIImage.FromBundle("arrow_down")
});
```

Customize Appearance

FlexGrid has various built-in properties to customize grid's appearance. A user can set attributes such as background color, alternating row color, text color, header color, font, selection mode, selected cell color, etc to customize the overall appearance of the FlexGrid control. Moreover, it also allows you to set individual properties for RowHeaderGridLinesVisibility, ColumnHeaderGridLinesVisibility, and TopLeftHeaderGridLinesVisibility that provides more granular styling capabilities that are more suited for Material themes.

The image below shows customized appearance in FlexGrid.

Id	First Name	Last Name	Address	City
0	Ted	Myers	209 Green AVE	Quetta
1	Steve	Paulson	116 Broad AVE	Kolkata
2	Rich	Evers	774 Golden BLV	Tijuana
3	Oprah	Frommer	270 Park ST	Chicago
4	Vic	Myers	320 Broad ST	Chaohu
5	Larry	Trask	235 Main AVE	Belo Horizonte
6	Zeb	Heath	856 Grand ST	Macau
7	Steve	Neiman	540 Green ST	Hyderabad
8	Zeb	Stevens	986 Panoramic .	San Pablo
9	Larry	Trask	300 Main AVE	San Antonio
10	Mark	Heath	204 Park ST	Bangalore
11	Mark	Ambers	480 Golden BLV	Sapporo
12	Paul	Myers	814 Panoramic :	Toluca
13	Herb	Trask	896 Park AVE	Samara
14	Steve	Quaid	542 Fake AVE	Saint Petersbur
15	Mark	Heath	258 Main ST	Querétaro
16	Karl	Trask	183 Golden ST	Omsk
17	Vic	Neiman	469 Main BLVD	Medan

The following code example demonstrates how to set style properties to customize grid's appearance. The example uses the sample created in the Quick Start section.

In Code

C#

C#

```
grid.SelectionMode = FlexSelectionMode.FlexSelectionModeCell;
grid.BackgroundColor = UIColor.Black;
grid.AlternatingRowBackgroundColor = UIColor.Gray;
grid.TextColor = UIColor.White;
grid.BorderColor = UIColor.Red;
grid.ColumnHeaderBackgroundColor = UIColor.White;
grid.ColumnHeaderTextColor = UIColor.Black;
```

```
grid.RowHeaderGridLinesVisibility = GridLinesVisibility.Vertical;
```

Clipboard and Keyboard Support

FlexGrid comes with clipboard support to readily provide cut, copy and paste operations. The control also supports hardware keyboards by allowing navigation through the use of arrow keys.

The supported keystrokes and their purpose are listed alphabetically as follows:

Keystroke	**Purpose/Description**
CMD+C	Copies the selected text.
CMD+X	Cuts the selected text.
CMD+V	Pastes the copied text.
Option (Alt) + Enter	Enters the edit mode.
Enter/Return	Leaves the edit mode.
Left	Navigates cell selection towards left.
Right	Navigates cell selection towards right.
Up	Navigates the cell selection upward.
Down	Navigates the cell selection downward.
Tab	Navigates the cell selection to the next column and then wraps to the next row. To enable the Tab key action, you need to set the **KeyActionTab** property to **Cycle**. This property accepts value from **GridTabAction** enumeration.
Shift+Left	Expands column range towards the left or decreases column range based on the context. If range has already expanded to right, this will gradually decrease range selection down to one column. After this single column threshold is hit, it will then expand the selection to the left if it continues to be pressed after this.
Shift+Right	Expands column range towards the right or decreases column range based on the context. If range has already expanded to left, this will gradually decrease range selection down to one column. After this single column threshold is hit, it will then expand the selection to the right if it continues to be pressed after this.
Shift+Up	Expands row range upwards or decreases row range based on the context. If range has already expanded downwards, this will gradually decrease range selection to one row. After this single row threshold is hit, it will then expand the selection to the upwards if it continues to be pressed after this.
Shift+Down	Expands row range downwards or decreases row range based on the context. If range has already expanded upwards, this will gradually decrease range selection to one row. After this single row threshold is hit, it will then expand the selection to the downwards if it continues to be pressed after this.

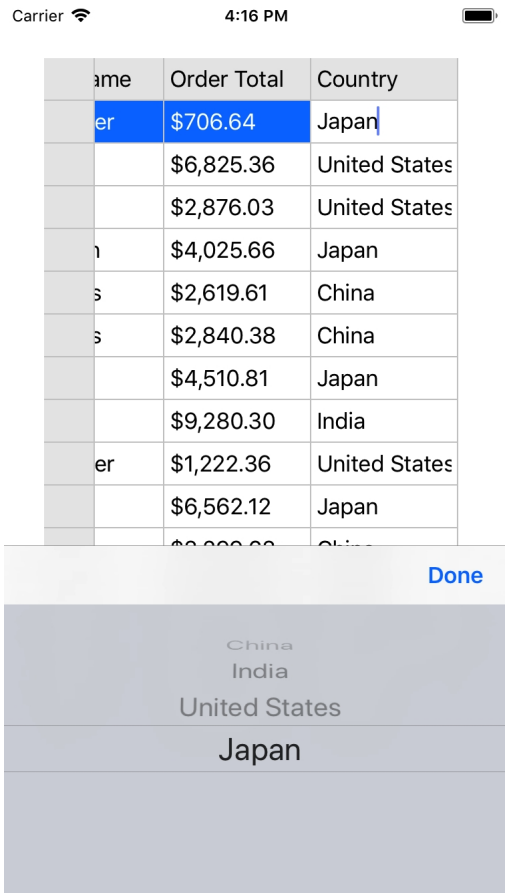
Data Mapping

Data Mapping provides auto look-up capabilities in FlexGrid. For example, you may want to display a customer name

instead of his ID, or a color name instead of its RGB value. When data mapping is configured, a picker is displayed when the user edits any cell in that column.

Data maps provide the grid with automatic look up capabilities. For example, you may want to display a customer's country name instead of his CountryID, or a color name instead of its RGB value.

The image below shows a FlexGrid with Data Mapping.



The code below binds a grid to a database of customers, then assigns a DataMap to the grid's 'CountryID' column so that the grid displays the country names rather than the raw IDs. The example uses the class, Customer, created in the Quick Start section. Add the following code to the ViewDidLoad method.

C#

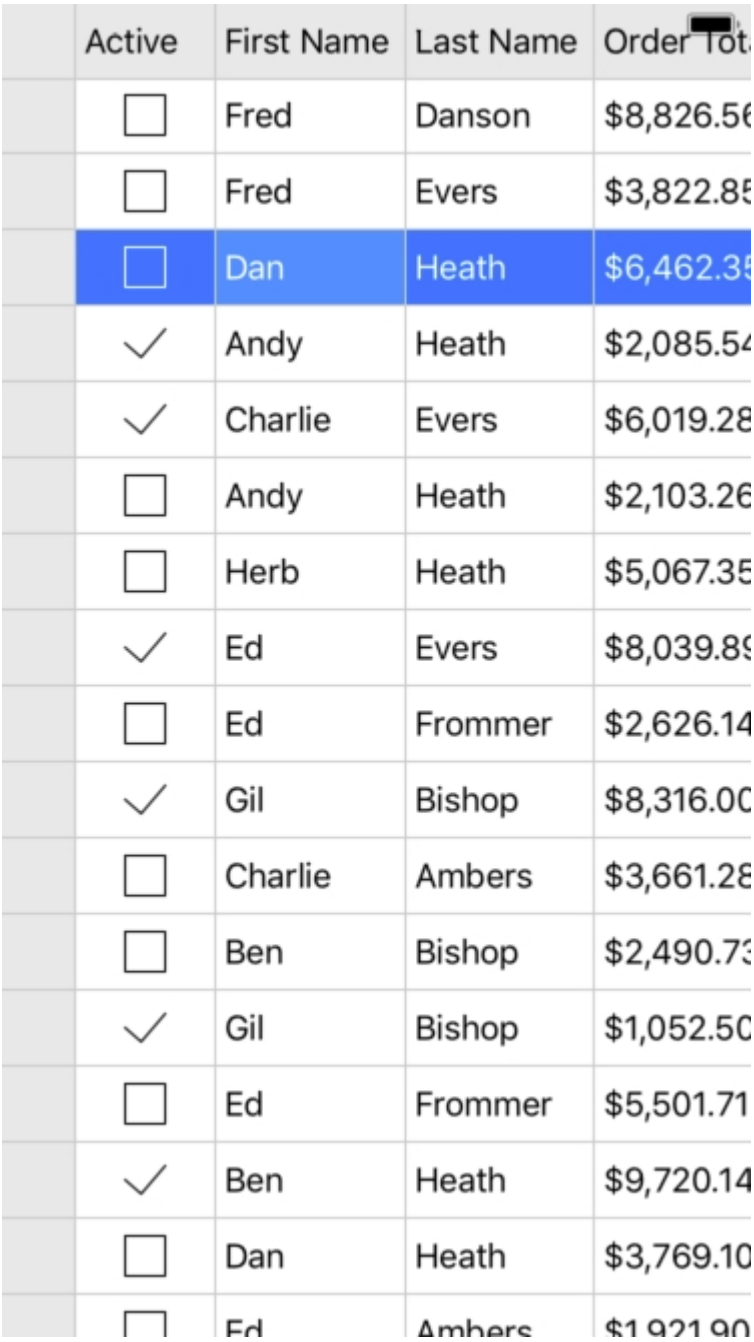
```
Grid.AutoGenerateColumns = false;
Grid.Columns.Add(new GridColumn { Binding = "Active", Width = new GridLength(70) });
Grid.Columns.Add(new GridColumn { Binding = "FirstName" });
Grid.Columns.Add(new GridColumn { Binding = "LastName" });
Grid.Columns.Add(new GridColumn { Binding = "OrderTotal", Format = "C", InputType =
UIKeyboardType.NumbersAndPunctuation });
Grid.Columns.Add(new GridColumn { Binding = "CountryId", Header = "Country" });
Grid.Columns["CountryId"].DataMap = new GridDataMap() { ItemsSource =
Customer.GetCountries(), DisplayMemberPath = "Value", SelectedValuePath = "Key" };
Grid.ItemsSource = Customer.GetCustomerList(100);
```

Defining Columns

With automatic column generation as one of the default features of FlexGrid, the control lets you specify the columns,

allows you to choose which columns to show, and in what order. This gives you control over each column's width, heading, formatting, alignment, and other properties. To define columns of FlexGrid, ensure that the AutoGenerateColumns is set to **false** (By default, it is true).

The image below shows how the FlexGrid control appears, after defining columns.



The following code example demonstrates how to define FlexGrid columns. The example uses the sample created in the Quick Start section.

In Code

C#

```
//Restricting auto generation of columns.
grid.AutoGenerateColumns = false;
```

```
// Defining Columns
grid.Columns.Add(new GridColumn { Binding = "Active", Width = new GridLength(70)
});
grid.Columns.Add(new GridColumn { Binding = "FirstName" });
grid.Columns.Add(new GridColumn { Binding = "LastName" });
grid.Columns.Add(new GridColumn { Binding = "OrderTotal", Format = "C", InputType
= UIKeyboardType.NumbersAndPunctuation });
grid.ItemsSource = Customer.GetCustomerList(100);
```

Editing

FlexGrid has built-in support for fast, in-cell editing like you find in Excel. 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.

Another way to start editing is 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 key. Data is automatically coerced to the proper type when editing finishes. If the user enters invalid data, the edit is cancelled and the original data remains in place. You can disable editing at the grid using the `isReadOnly` property of the grid.

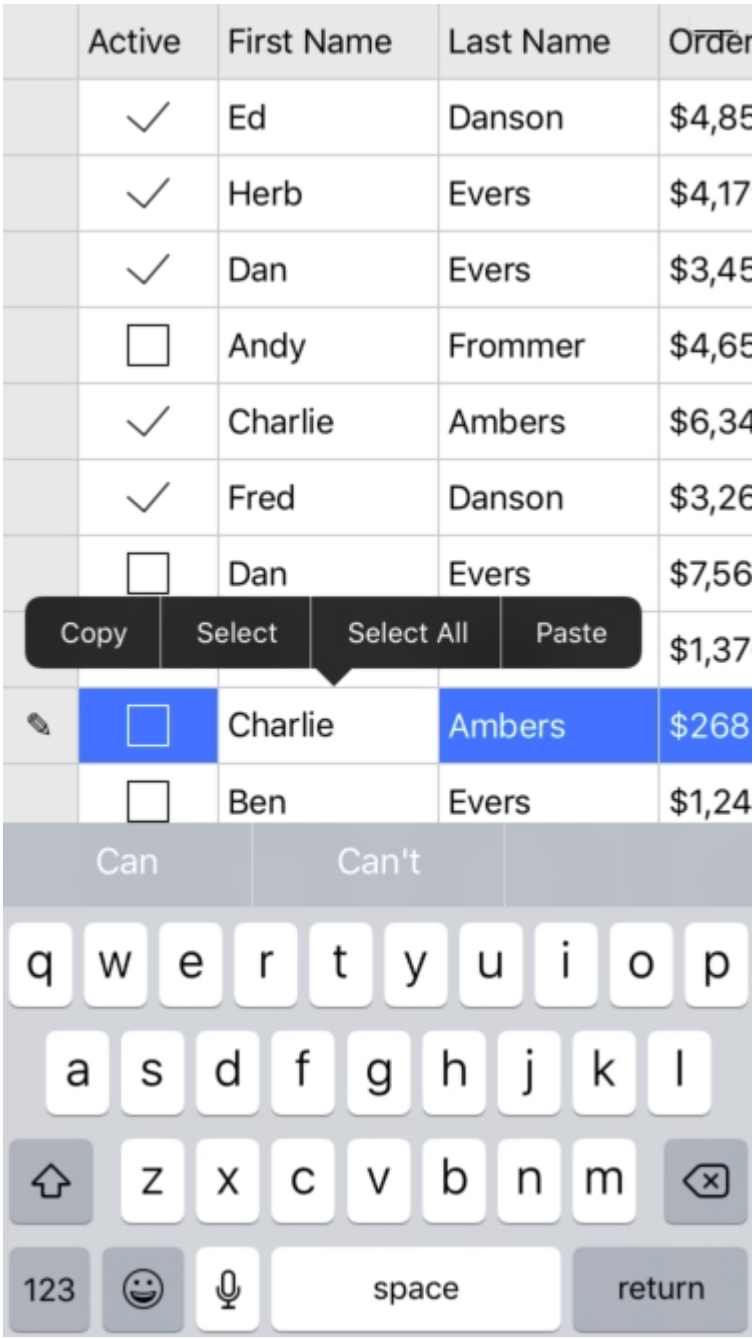
FlexGrid provides support for various types of Editing, including inline and custom cell editing.

Inline Editing

FlexGrid allows a user to perform in-line editing using the default iOS device keyboard. Double clicking inside a cell puts it into a quick edit mode. Once you select the content inside the cell or row, it gives you options to **Copy, Select, SelectAll, Paste** etc. for a smooth editing experience.

Once you have entered the new data, click the **return** button, or simply press **Enter**, this automatically updates the data in the appropriate format. You can set the IsReadOnly to **true** for the rows and columns that you want to restrict editing for.

The image below shows how the FlexGrid control appears, after these properties have been set.



The following code example demonstrates how to achieve inline editing in FlexGrid. The example uses the sample created in the Quick Start section.

In Code

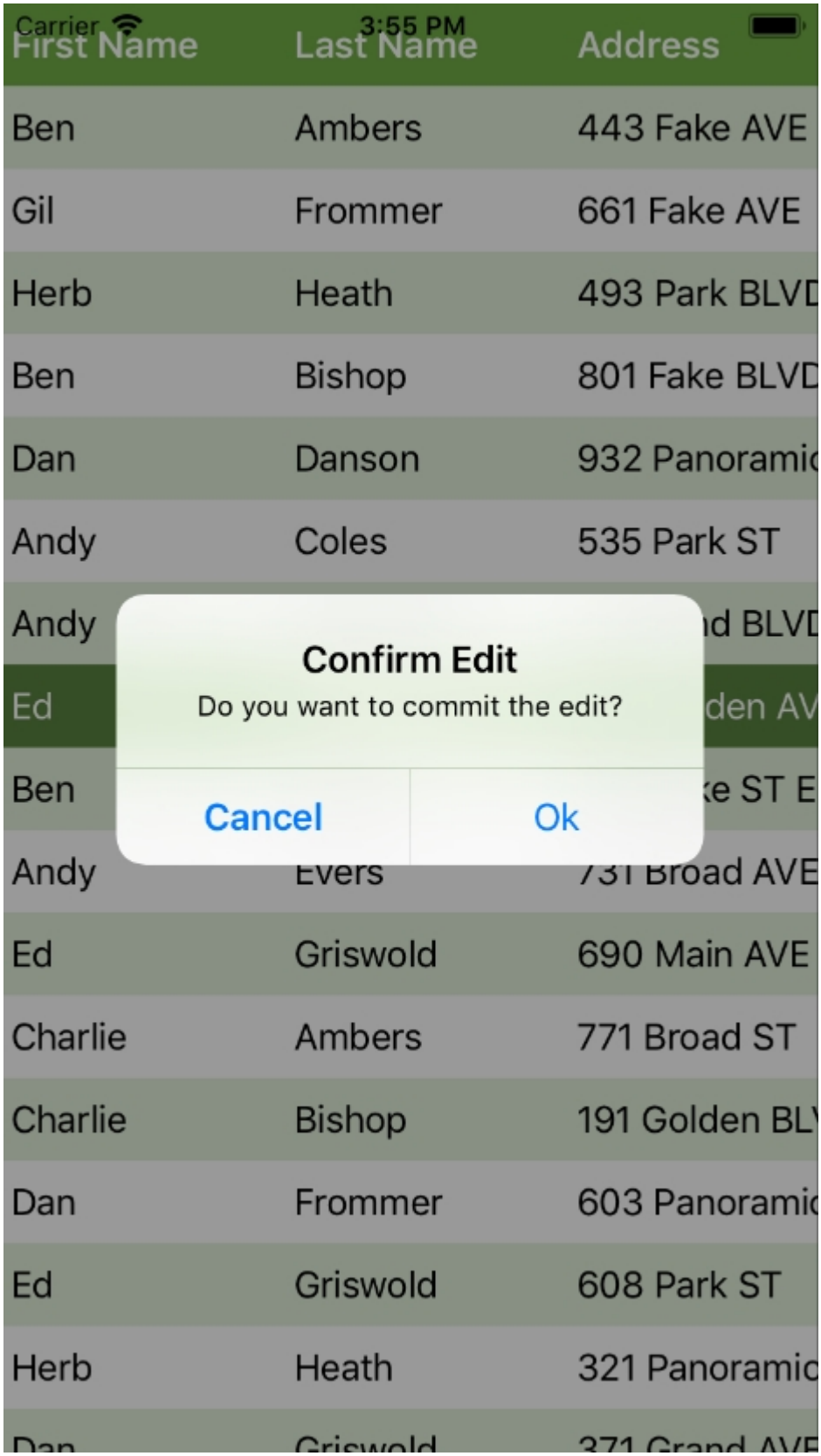
C#

```
grid.IsReadOnly = false;
```

Custom Editing

FlexGrid allows you to add various functions while performing editing in the cells. Using these functions, you can customize the overall editing experience. You can add a confirmation message box that pops up to confirm whether a user is sure or not about committing a particular change, providing an attractive and flexible way to update data.

The image below shows how the FlexGrid appears, after an Edit confirmation message box is added.



The following code examples how to achieve custom editing. These examples use the sample created in the Quick Start section.

In Code

C#

```
public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            var data = Customer.GetCustomerList(100);
            grid.AutoGeneratingColumn += (s, e) =>
            {
                if (e.Property.Name == "Id")
                    e.Cancel = true;
            };
            grid.ItemsSource = data;
            grid.BeginningEdit += OnBeginningEdit;
            grid.CellEditEnded += OnCellEditEnded;

            grid.GridLinesVisibility = GridLinesVisibility.None;
            grid.HeadersGridLinesVisibility = GridLinesVisibility.None;
            grid.HeadersVisibility = GridHeadersVisibility.Column;
            grid.BackgroundColor = UIColor.White;
            grid.RowBackgroundColor = ColorEx.FromARGB(0xFF, 0xE2, 0xEF, 0xDB);
            grid.RowTextColor = UIColor.Black;
            grid.AlternatingRowBackgroundColor = UIColor.White;
            grid.ColumnHeaderBackgroundColor = ColorEx.FromARGB(0xFF, 0x70, 0xAD,
0x46);
            grid.ColumnHeaderTextColor = UIColor.White;
            grid.ColumnHeaderFont =
UIFont.BoldSystemFontOfSize(UIFont.LabelFontSize);
            grid.SelectionBackgroundColor = ColorEx.FromARGB(0xFF, 0x5A, 0x82,
0x3F);
            grid.SelectionTextColor = UIColor.White;
        }

        private object _originalValue;

        private void OnBeginningEdit(object sender, GridCellEditEventArgs e)
        {
            _originalValue = grid[e.CellRange.Row, e.CellRange.Column];
        }

        private void OnCellEditEnded(object sender, GridCellEditEventArgs e)
        {
            var originalValue = _originalValue;
            var currentValue = grid[e.CellRange.Row, e.CellRange.Column];
            if (!e.CancelEdits && (originalValue == null && currentValue != null
|| !originalValue.Equals(currentValue)))
            {
                var alert = new UIAlertView();
                alert.Title =
Foundation.NSBundle.MainBundle.GetLocalizedString("Confirm Edit", "");
                alert.Message =
Foundation.NSBundle.MainBundle.GetLocalizedString("Do you want to commit the
```

```
edit?", "");

alert.AddButton(Foundation.NSBundle.MainBundle.GetLocalizedString("Ok", ""));

alert.AddButton(Foundation.NSBundle.MainBundle.GetLocalizedString("Cancel", ""));
                alert.CancelButtonIndex = 1;
                alert.Clicked += (s, e2) =>
                {
                    if (e2.ButtonIndex == alert.CancelButtonIndex)
                    {
                        grid[e.CellRange.Row, e.CellRange.Column] = originalValue;
                    }
                };
                alert.Show();
            }
        }
        public override void ViewDidLayoutSubviews()
        {
            base.DidReceiveMemoryWarning();
            // Release any cached data, images, etc that aren't in use.
            grid.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
                                    this.View.Frame.Width,
this.View.Frame.Height);
        }
```

Add New Row

FlexGrid allows you to show a new row template at the top or bottom of the grid using NewRowPosition property, which takes values from the GridNewRowPosition enumeration. You can control the text to be displayed to a user by setting the NewRowPlaceHolder property. Users may use the new row template to add items to the grid's itemsSource collection.

The image below shows how the FlexGrid appears after adding the new row.

The following code example demonstrates how to set this property in C#. The example uses the sample created in the Quick start section.

In Code

C#

```
grid.NewRowPosition = GridNewRowPosition.Top;
grid.NewRowPlaceholder = "Tap to begin entering a new row";
```

Export

FlexGrid allows you to export a file and save it to a device or a stream. You export files to text, CSV and HTML formats and save them to a file system, a Stream or a StreamWriter usingRadial Save method of the FlexGrid class. The **Save** method can save the exported file with different options of encoding and presentation of the FlexGrid data. The method has following seven overloads:

Overload	Description
Save(StreamWriter, GridFileFormat, GridSaveOptions)	Saves the contents of the grid to a System.IO.StreamWriter.
Save(Stream, GridFileFormat, Encoding, GridSaveOptions)	Saves the contents of the grid to a stream with specified memory location, format, encoding, and options.
Save(Stream, GridFormat, GridSaveOptions)	Saves the contents of the grid to a UTF8 encoded stream with specified memory location, format and options.

Save(Stream, GridFileFormat)	Saves the contents of the grid to a UTF8 encoded stream with specified memory location and format.
Save(String, GridFileFormat, Encoding, GridSaveOptions)	Saves the contents of the grid to a file a with specified name, format, encoding, and options.
Save(String, GridFileFormat, GridSaveOptions)	Saves the contents of the grid to a UTF8 encoded file with specified name, format and options.
Save(String, GridFileFormat)	Saves the content of the grid to a UTF8 encoded file with specified name and format.

The following code example demonstrates the implementation of the Save method to save the exported file. The example uses the sample created in the Quick Start section.

C# copyCode

```
string fullPath = Path.Combine(Environment.GetFolderPath(
Environment.SpecialFolder.LocalApplicationData),
"ExportedGrid") + "." + "csv";
grid.Save(fullPath, GridFileFormat.Csv, System.Text.Encoding.UTF8,
GridSaveOptions.SaveColumnHeaders);
```

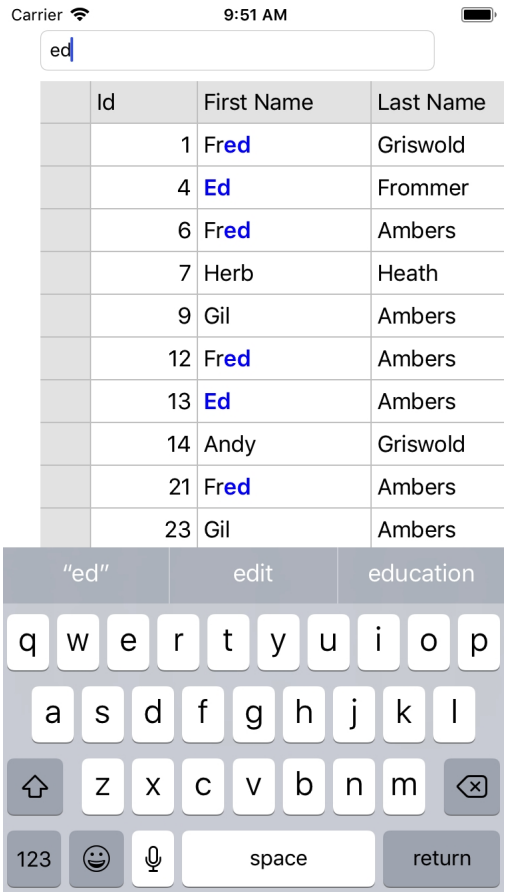
Filtering

FlexGrid supports filtering through the ICollectionView interface and FullTextFilterBehavior class. Filtering allows a user to display subsets of data out of large data sets based on the criteria defined. Collection View interface provided by FlexGrid provides a flexible and efficient way to filter and display the desired dataset.

The FlexGrid control lets a user perform custom filtering by adding a text box to enter a value and display a particular set of data. It also allows a user to use various options, such as BeginsWith, Contains, EndsWith, Equals, LessThamn, GreaterThan etc. A user can define the filtering patterns on the basis of the requirements, which can be a specific data or an approximate set of values.

Search Box Filtering

FlexGrid provides you flexibility to use a search box to filter out data. Users can add the filter search box and set its attributes, including its height, width, color, text, filtering pattern as per their requirements. This example demonstrates a simple text box that lets you type the value you want to search in the grid. For example, when you type 'ed' in the Filter text box, the FullTextFilterBehavior class can be used to filter the grid data to display all the values containing 'ed'.



The example uses the class, Customer, created in the Quick Start section. Add the following code to the ViewDidLoad() method for filtering data using search box.

CS

```
var data = Customer.GetCustomerList(100);
Grid.ItemsSource = data;
var fullTextFilter = new FullTextFilterBehavior();
fullTextFilter.HighlightColor = UIColor.Blue;
fullTextFilter.Attach(Grid);
fullTextFilter.FilterEntry = Filter;
```

Frozen Rows and Columns

FlexGrid allows you to freeze rows and columns so that they remain visible as the user scrolls the grid. Frozen cells can be edited and selected as regular cells, exactly as in Excel. A user can freeze rows and columns separately and also simultaneously. While working with large data sets in grid, it is convenient to keep some of the rows or/and columns locked in the view by setting the values for FrozenRows and FrozenColumns properties of FlexGrid.

Freezing a cell in the grid doesn't affect the SelectionMode of the FlexGrid. If a user has set a value of SelectionMode in FlexGrid, then the frozen cells along with the cells that are not frozen can be selected simultaneously by the user.

In this example, the first tow column and the first row of the FlexGrid are frozen. The image below shows how the FlexGrid control appears, after applying cell freezing to rows and columns:

	Id	Amount	Country
	0	$399.04	Germany
	1	$722.22	Greece
	2	$900.11	Italy
	3	$395.49	Japan
	4	$846.83	UK
	5	$623.52	US
	6	$879.74	Germany
	7	$703.08	Greece
	8	$347.32	Italy
	9	$661.47	Japan
	10	$910.35	UK
	11	$915.42	US
	12	$751.65	Germany
	13	$595.30	Greece

The following code example demonstrates how to freeze rows and columns in FlexGrid. The example uses data source added in the Quick Start section.

In Code

C#

```
grid.FrozenColumns = 2;
grid.FrozenRows = 1;
```

Formatting Columns

You can convert the raw values appearing in any column of the FlexGrid into different formats. FlexGrid supports standard .NET format strings used to display numeric values as local currency, percentages, dates, and more. Formatting only applies to display value of a cell and does not affect the underlying data.

Formatting Numbers and Dates

You can specify a column's format by setting its Format property to a valid string. The string should contain one or two characters from a known list of formats. The first character specifies the format (format specifier) while the second optional character specifies precision (precision specifier). For example, the following format string converts a raw value to display local currency up to 2 decimal places.

The following code example demonstrates how to specify a column's format in C#. The example uses the sample created in the Quick start section.

In Code

```
grid.Columns["OrderTotal"].Format = "N2";
```

The format strings are not case sensitive. The following table lists the numerical format strings that FlexGrid currently supports.

Format Specifier	Name	Precision Specifier	Example
"C" or "c"	Currency	Number of decimal digits	123.4567 (C2) → $123.46
"D" or "d"	Decimal	Minimum number of digits	1234 (D6) → 001234
"E" or "e"	Exponential	Number of decimal digits	1,234 (E2) → 1.23E3
"F" or "f"	Fixed Point	Number of decimal digits	-1234.56 (F4) → -1234.5600
"G" or "g"	General	Number of significant digits	123.45 (G4) → 123.5
"N" or "n"	Number	Desired number of decimal places	1234 (N1) → 1234.0
"P" or "p"	Percent	Desired number of decimal places	1 (P2) → 100.00%
"X" or "x"	Hexadecimal	Desired number of digits in the result	123 (X2) → 7B

Custom Date/Time Format Strings

FlexGrid also supports custom date and time format strings that allow you to display date and time values in numerous ways. For example, the format string below converts the default format (MM/dd/yy) to "yyyy-MM" format.

In Code

```
grid.Columns["Hired"].Format = "yyyy-MM";
```

The DateFormat string is created using a combination of format specifiers. The following table lists some common format specifiers for creating custom date and time strings.

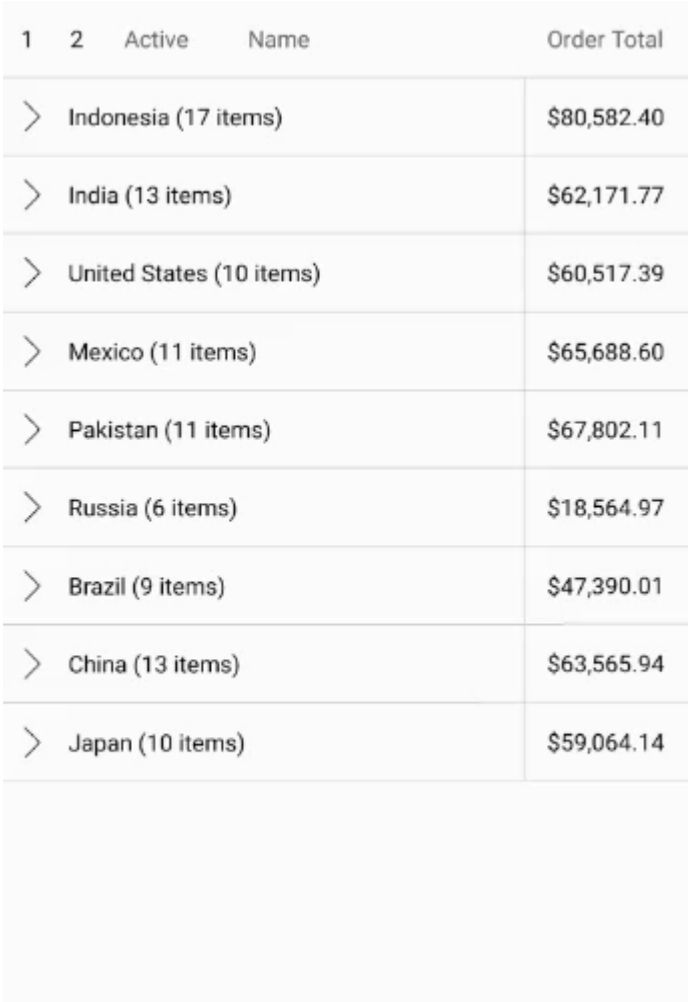
Format Specifier	Description	Example
"d"	Day of the month	1

"dd"	Day of the month	01
"ddd"	Abbreviated day of the week	Mon
"dddd"	Full day of the week	Monday
"h"	The hour using 12-hour clock	1
"hh"	The hour using 12-hour clock	01
"H"	The hour using 24-hour clock	13
"HH"	The hour using 24-hour clock	13
"m"	Minute of time	1
"mm"	Minute of time	01
"M"	Month number	1
"MM"	Month number	01
"MMM"	Abbreviated month name	Mar
"MMMM"	Full month name	March
"s"	Second of time	1
"ss"	Second of time	01
"tt"	The AM/PM Designator	AM
"yy"	Abbreviated year	16
"yyyy"	Full year	2016
"\"	Escape character	H/H => 13H
Any other character	The character is copied to the result string	yyyy-MM => 2016-03

Grouping

FlexGrid supports grouping through the **CollectionView**. To enable grouping, you can use GroupAsync method. GroupAsync method allows you to group the collection view according to the specified group path. Users can expand or collapse groups in FlexGrid by tapping anywhere within the group row.

The image below shows how the FlexGrid appears, after grouping is applied to column **Country**.



The following code example demonstrates how to set apply grouping in FlexGrid. The example uses data source added in the Quick Start section.

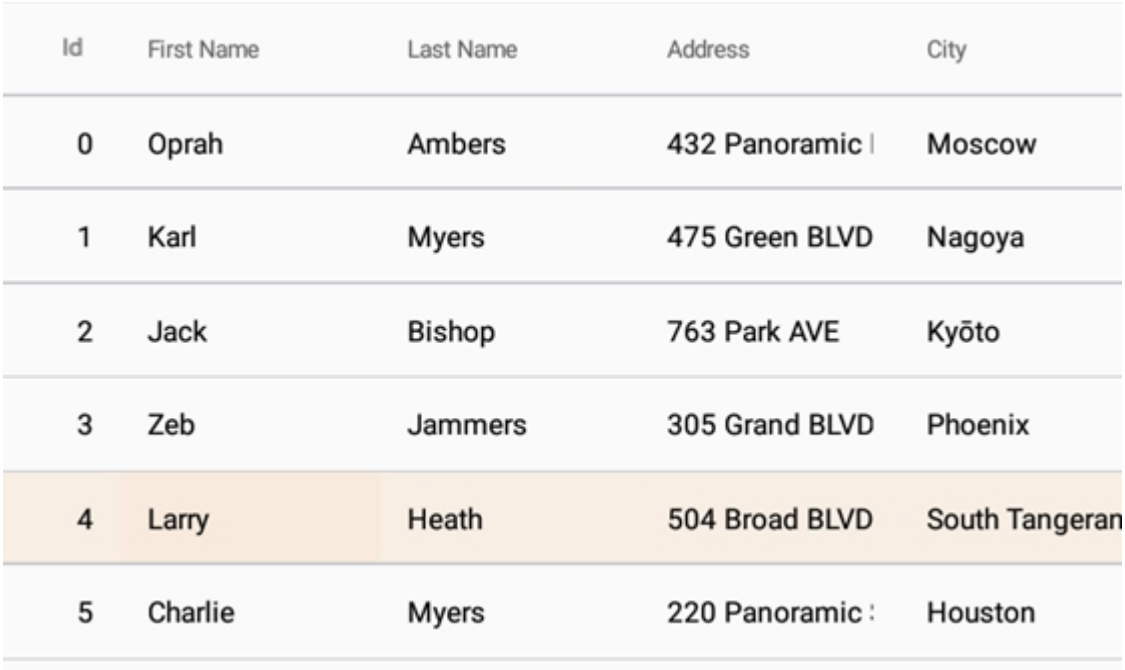
CS

```
using C1.CollectionView;
using C1.iOS.Grid;

public partial class ViewController : UIViewController
{
    C1CollectionView<Customer> _collectionView;
    public ViewController(IntPtr handle) : base(handle)
    {

    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();

        var task = UpdateVideos();
    }

    private async Task UpdateVideos()
```

```
    {
        var data = Customer.GetCustomerList(100);
        _collectionView = new C1CollectionView<Customer>(data);
        await _collectionView.GroupAsync(c => c.Country);
        Grid.AutoGenerateColumns = false;
        Grid.Columns.Add(new GridColumn { Binding = "Active", Width = new
GridLength(60) });
        Grid.Columns.Add(new GridColumn { Binding = "Name", Width = GridLength.Star
});
        Grid.Columns.Add(new GridColumn { Binding = "OrderTotal", Width = new
GridLength(110), Format = "C", Aggregate = GridAggregate.Sum, HorizontalAlignment =
UIControlContentHorizontalAlignment.Right, HeaderHorizontalAlignment =
UIControlContentHorizontalAlignment.Right });
        Grid.GroupHeaderFormat = Foundation.NSBundle.MainBundle.LocalizedString("
{name}: {value} ({count} items)", "");
        Grid.ItemsSource = _collectionView;
    }

    partial void OnCollapseClicked(UIButton sender)
    {
        Grid.CollapseGroups();
    }
}
```

Material Theme

From 2018v3 onwards, the default appearance of FlexGrid has changed to the material design pattern. So, for users using 2018v3 or beyond, FlexGrid appears as follows.



Id	First Name	Last Name	Address	City
0	Oprah	Ambers	432 Panoramic	Moscow
1	Karl	Myers	475 Green BLVD	Nagoya
2	Jack	Bishop	763 Park AVE	Kyōto
3	Zeb	Jammers	305 Grand BLVD	Phoenix
4	Larry	Heath	504 Broad BLVD	South Tangeran
5	Charlie	Myers	220 Panoramic	Houston

In order to revert to the classic style which was there until 2018v2, use the following code:

C#

```
Grid.Style = GridStyle.Classic;
```

We have also added following features to support the material design in the FlexGrid control:

- **Checklist behavior**
  To further support the material design, you can now enable the checklist like selection. This kind of selection lets you select the non-consecutive rows of data. You can enable it by using the following code:

  C#

  ```
  var details = new CheckListBehavior();
  details.SelectionBinding = "Selected";
  details.Attach(grid);
  ```

- **Header grid lines**
  Also, to provide more garnular styling capabilities suited for Material design, FlexGrid supports RowHeaderGridLinesVisibility, ColumnHeaderGridLinesVisibility, and TopLeftHeaderGridLinesVisibility properties to display or hide the individual header grid lines.

Merging Cells

The FlexGrid control allows you to merge cells, making them span multiple rows or columns. This capability enhances the appearance and clarity of the data displayed on the grid. The effect of these settings is similar to the HTML <ROWSPAN> and <COLSPAN> tags.

To enable cell merging, you must do two things:

1. Set the grid's AllowMerging property. This property accepts value from the GridAllowMerging enumeration.
2. If you wish to merge columns, set the AllowMerging property to **True** for each column that you would like to merge. If you wish to merge rows, set the AllowMerging property to **True** for each row that you would like to merge.

Merging occurs if the adjacent cells contain the same non-empty string. There is no method to force a pair of cells to merge. Merging occurs automatically based on the cell contents. This makes it easy to provide merged views of sorted data, where values in adjacent rows present repeated data.

Cell merging has several possible uses. For instance, you can use this feature to create merged table headers, merged data views, or grids where the text spills into adjacent columns.

In Code

The following code example demonstrates how to apply merging in the FlexGrid control. The example uses the sample created in Custom Cells topic.

C#

```
grid.AllowMerging = GridAllowMerging.Cells;
grid.Columns[1].AllowMerging = true;
```

Resizing Columns

FlexGrid's AllowResizing property allow users to resize the column by simply touching and dragging the handle

between two columns. The AllowResizing property accepts value from the GridAllowResizing enumeration.

In case you want to restrict resizing at runtime, set the AllowResizing property of FlexGrid to **None**. Moreover, to allow resizing for specific columns, set the AllowResizing property for a particular column instead of the entire FlexGrid. The resizing functionality for columns is useful when a user wants to add data in FlexGrid. The user can simply resize the column as per the requirement directly on the device without requiring to change or set the width in code.

The following code example demonstrates how to set the AllowResizing property in FlexGrid. The example uses sample created in the Quick Start section.

In Code

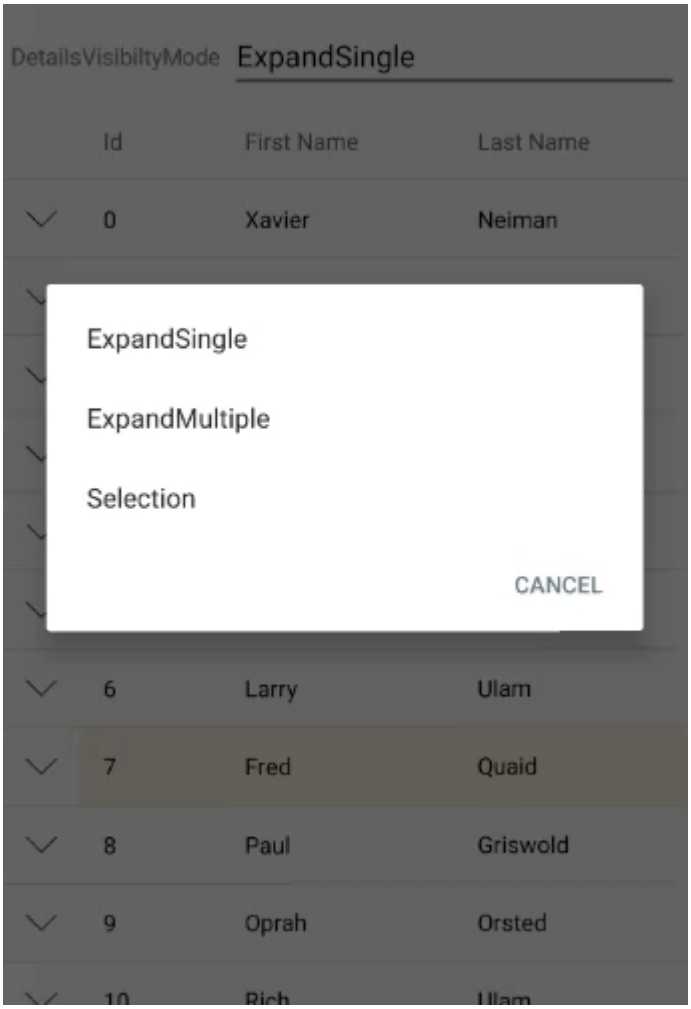
C#

```
// Set AllowResizing property for FlexGrid
grid.AllowResizing = GridAllowResizing.Both;

// Set AllowResizing property for specific column
grid.Columns[1].AllowResizing = true;
```

Row Details

C1FlexGrid control allows you to create a hierarchical grid by adding a row details section to each row. Adding a row details sections allows you to group some data in a collapsible template and present only a summary of the data for each row. The row details section is displayed only when the user taps a row. Moreover, you can set the details visibility mode to expand single, expand multiple or selection, with the help of **DetailVisibiltyMode** property provided by the FlexGrid class.

The image given below shows a FlexGrid with row details section added to each row.



The following code examples demonstrate how to add row details section to the FlexGrid control in C#. The example uses the class, Customer, created in the Quick Start section.

1. Add C1FlexGrid control the Mainstoryboard and add data to it through code.

CS

```
public partial class ViewController : UIViewController
{
    public ViewController(IntPtr handle) : base(handle)
    {

    }
    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
        var data = Customer.GetCustomerList(1000);
        Grid.AutoGenerateColumns = false;
        Grid.Columns.Add(new GridColumn() { Binding = "Id", Width =
GridLength.Auto });
        Grid.Columns.Add(new GridColumn() { Binding = "FirstName", Width =
GridLength.Star });
        Grid.Columns.Add(new GridColumn() { Binding = "LastName", Width =
GridLength.Star });
        var details = new FlexGridDetailProvider();
```

```
        details.Attach(Grid);
        details.DetailCellCreating += OnDetailCellCreating;
        details.Height = GridLength.Auto;
        Grid.ItemsSource = data;
    }
    private void OnDetailCellCreating(object sender,
GridDetailCellCreatingEventArgs e)
    {
        var customer = e.Row.DataItem as Customer;
        e.Content = new DetailView(customer);
    }
}
```

2. Create a row details template and add it to the FlexGrid rows for diplaying details.

CS

```
public class DetailView : UIView
{
    double _spacing = 8;
    UILabel _countryLabel;
    UILabel _cityLabel;
    UILabel _addressLabel;
    UILabel _postalCodeLabel;

    public DetailView(Customer customer)
    {
        _countryLabel = new UILabel { Text = string.Format("Country: {0}",
customer.Country) };
        _cityLabel = new UILabel { Text = string.Format("City: {0}",
customer.City) };
        _addressLabel = new UILabel { Text = string.Format("Address: {0}",
customer.Address) };
        _postalCodeLabel = new UILabel { Text = string.Format("Postal Code:
{0}", customer.PostalCode) };
        AddSubviews(_countryLabel, _cityLabel, _addressLabel, _postalCodeLabel);
    }

    public override CGSize IntrinsicContentSize
    {
        get
        {
            var countryLabelSize = _countryLabel.IntrinsicContentSize;
            var cityLabelSize = _cityLabel.IntrinsicContentSize;
            var addressLabelSize = _addressLabel.IntrinsicContentSize;
            var postalCodeLabelSize = _postalCodeLabel.IntrinsicContentSize;

            return new CGSize(_spacing * 2 + Math.Max(countryLabelSize.Width,
Math.Max(cityLabelSize.Width, Math.Max(addressLabelSize.Width,
postalCodeLabelSize.Width))), _spacing * 5 + countryLabelSize.Height +
cityLabelSize.Height + addressLabelSize.Height + postalCodeLabelSize.Height);
        }
    }
```

```
    public override void LayoutSubviews()
    {
        base.LayoutSubviews();
        var countryLabelSize = _countryLabel.IntrinsicContentSize;
        var cityLabelSize = _cityLabel.IntrinsicContentSize;
        var addressLabelSize = _addressLabel.IntrinsicContentSize;
        var postalCodeLabelSize = _postalCodeLabel.IntrinsicContentSize;

        _countryLabel.Frame = new CGRect(_spacing, _spacing,
countryLabelSize.Width, countryLabelSize.Height);
        _cityLabel.Frame = new CGRect(_spacing, _spacing +
countryLabelSize.Height + _spacing, cityLabelSize.Width, cityLabelSize.Height);
        _addressLabel.Frame = new CGRect(_spacing, _spacing +
countryLabelSize.Height + _spacing + cityLabelSize.Height + _spacing,
addressLabelSize.Width, addressLabelSize.Height);
        _postalCodeLabel.Frame = new CGRect(_spacing, _spacing +
countryLabelSize.Height + _spacing + cityLabelSize.Height + _spacing +
addressLabelSize.Height + _spacing, postalCodeLabelSize.Width,
postalCodeLabelSize.Height);
    }
}
```

Selecting Cells

The SelectionMode property of the FlexGrid allows you to define the selection mode of the cells by setting its value to **Row, Cell, CellRange, or RowRange**. This property accepts these values from the GridSelectionMode enumeration.

The image below shows how the FlexGrid control appears after the selectionMode property is set to Row.

	Id	Country	Amount	Active
	0	Germany	456.4	☑
	1	Greece	825.53	☐
	2	Italy	785.13	☐
	3	Japan	437.78	☐
	4	UK	73.65	☑
	5	US	35.46	☐
	6	Germany	456.94	☐
	7	Greece	758.23	☐
	8	Italy	432.59	☑
	9	Japan	484.87	☐

The following code example demonstrates how to choose selection modes in FlexGrid. The example uses the sample created in the Quick Start section.

CS

```
grid.SelectionMode = GridSelectionMode.Row;
```

The **SelectionMode** property also controls dictates how a deletion works for a row, cell, cell range, or row range from the FlexGrid control.

Selection Menu

The selection menu contains common actions such as, editing, selecting, and deleting text. In FlexGrid, selection menu is enabled by default. However, you can disable it by setting the **ShowSelectionMenu** property to false. In desktop applications, you can activate the selection menu with a right click on a cell. In mobile applications, you can activate selection menu with a long press on a cell or by tapping the row header. You can also add custom actions to the selection menu by setting a handler for **CreateSelectionMenu** event.

Use the following code snippet to create custom action for the selection menu.

C#

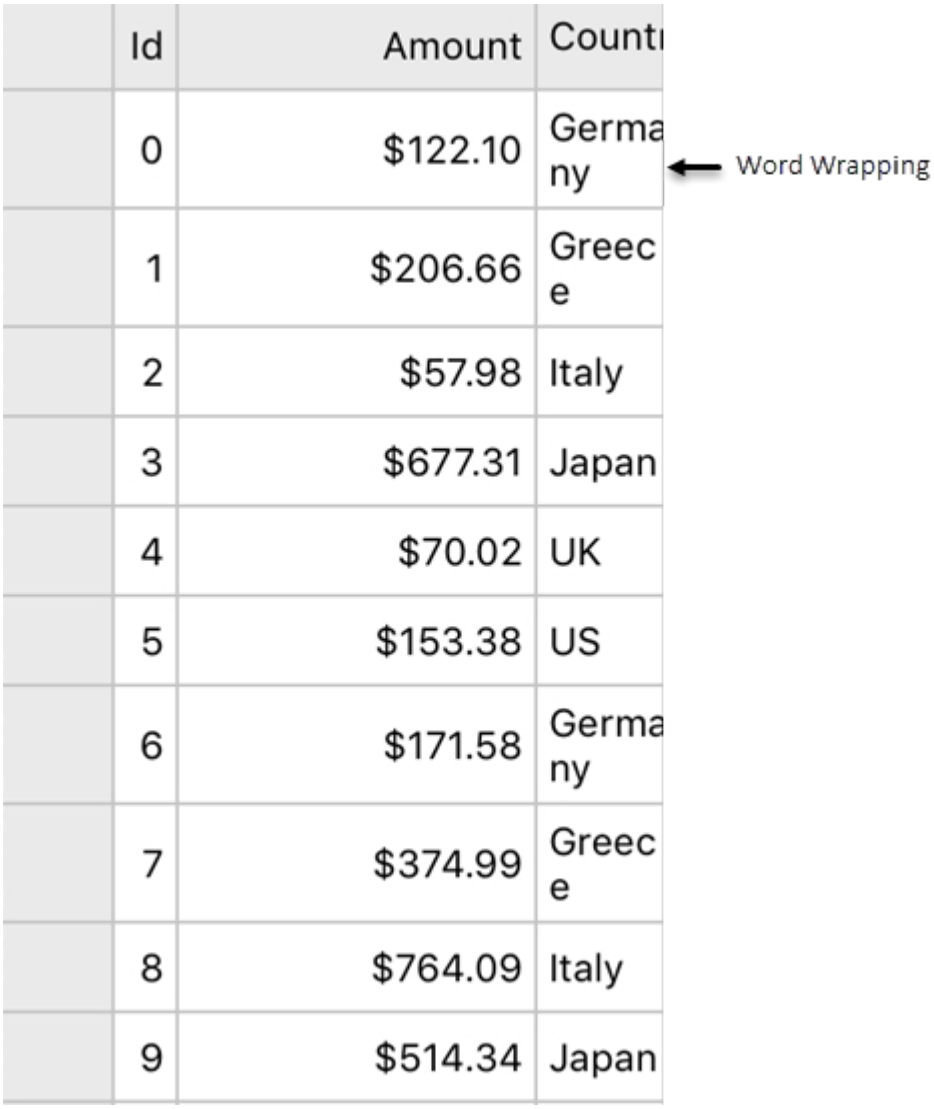
```
grid.CreatingSelectionMenu += Grid_CreatingSelectionMenu;

 private void Grid_CreatingSelectionMenu(object sender, GridSelectionMenuEventArgs e)
 {
  e.Menu.Items.Add(new GridMenuItem("Clear", () => Clear(e)));
 }

 public void Clear(GridSelectionMenuEventArgs e) {
  for (int c = e.CellRange.Column; c <= e.CellRange.Column2; c++) {
   for (int r = e.CellRange.Row; r <= e.CellRange.Row2; r++) {
    grid[r, c] = null;
   }
  }
 }
 return grid;
```

	ID	Name	Country	Country ID	Active	First	Last
	0	Charlie Jammers	Japan	4	☑	Charlie	Jammers
	1	Xavier Orsted	Congo	1	☑	Xavier	Orsted
	2	Gil Jammers	Congo	1	☑	Gil	Jammers
		Gil Orsted	Thailand	5	☐	Gil	Orsted
		Oprah Frommer	Japan	4	☐	Oprah	Frommer
		Steve Jammers	United States	3	☑	Steve	Jammers
		Xavier Orsted	Brazil	0	☑	Xavier	Orsted
	7	Herb Jammers	Egypt	2	☐	Herb	Jammers
	8	Herb Krause	Brazil	0	☑	Herb	Krause
	9	Herb Krause	Thailand	5	☑	Herb	Krause

Word Wrapping

FlexGrid's WordWrap property allows a user to display multi-line text in a single cell of the grid. To enable word wrapping in a column, set the value of WordWrap property to **true**. By default, its value is set to **false**. The WordWrap property determines whether the grid should automatically break long strings containing multiple words and special characters such as spaces in multiple lines. Multiple line text can be displayed in both fixed and scrollable cells.

The image below shows how the FlexGrid control appears, after word wrapping is applied to the column Country:



The following code example demonstrates setting word wrap in FlexGrid.

In Code

C#

```
// Set WordWrap for a specific column
grid.Columns[1].WordWrap = true;
```

FlexPie

The FlexPie control allows you to create customized pie charts that represent a series as slices of a pie. The arc length of each slice depicts the value represented by that slice.

Pie charts are commonly used to display proportional data such as percentage cover. 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

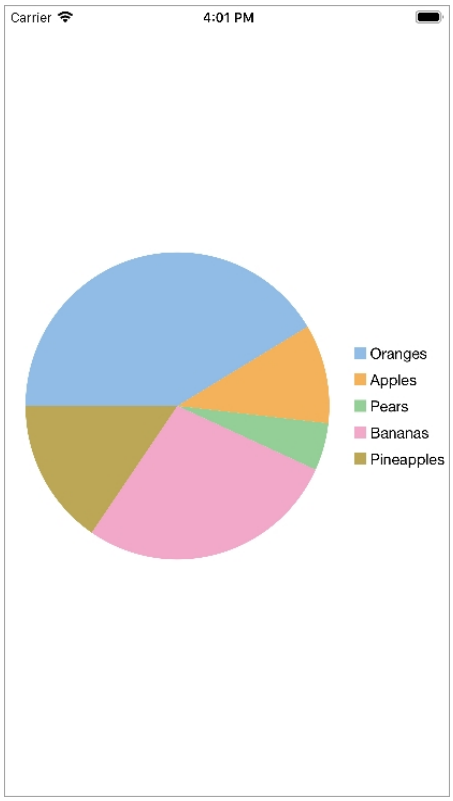
- **Touch Based Labels**: Displays values using touch based labels.
- **Exploding and Donut Pie Charts**: Converts a standard pie chart into an exploding or a donut pie chart.

Quick Start: Add data to FlexPie

This section describes how to add a FlexPie control to your iOS app and add data to it. This topic consists of three steps:

- **Step 1: Create a data source for FlexPie**
- **Step 2: Add a FlexPie control**
- **Step 3: Run the Application**

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



Step 1: Create a data source for FlexPie

Add a new class to serve as the data source for FlexPie.

C#

```
public class PieChartData
      {
        public string Name {get; set;}
        public double Value {get; set;}

        public static IEnumerable<PieChartData> DemoData()
             {
                    List<PieChartData> result = new List<PieChartData> ();
                    string[] fruit = new string[]
{"Oranges","Apples","Pears","Bananas","Pineapples" };
                    Random r = new Random ();

                    foreach (var f in fruit)
                    result.Add (new PieChartData { Name = f, Value = r.Next(100) * 101});

                    return result;
              }
          }
```

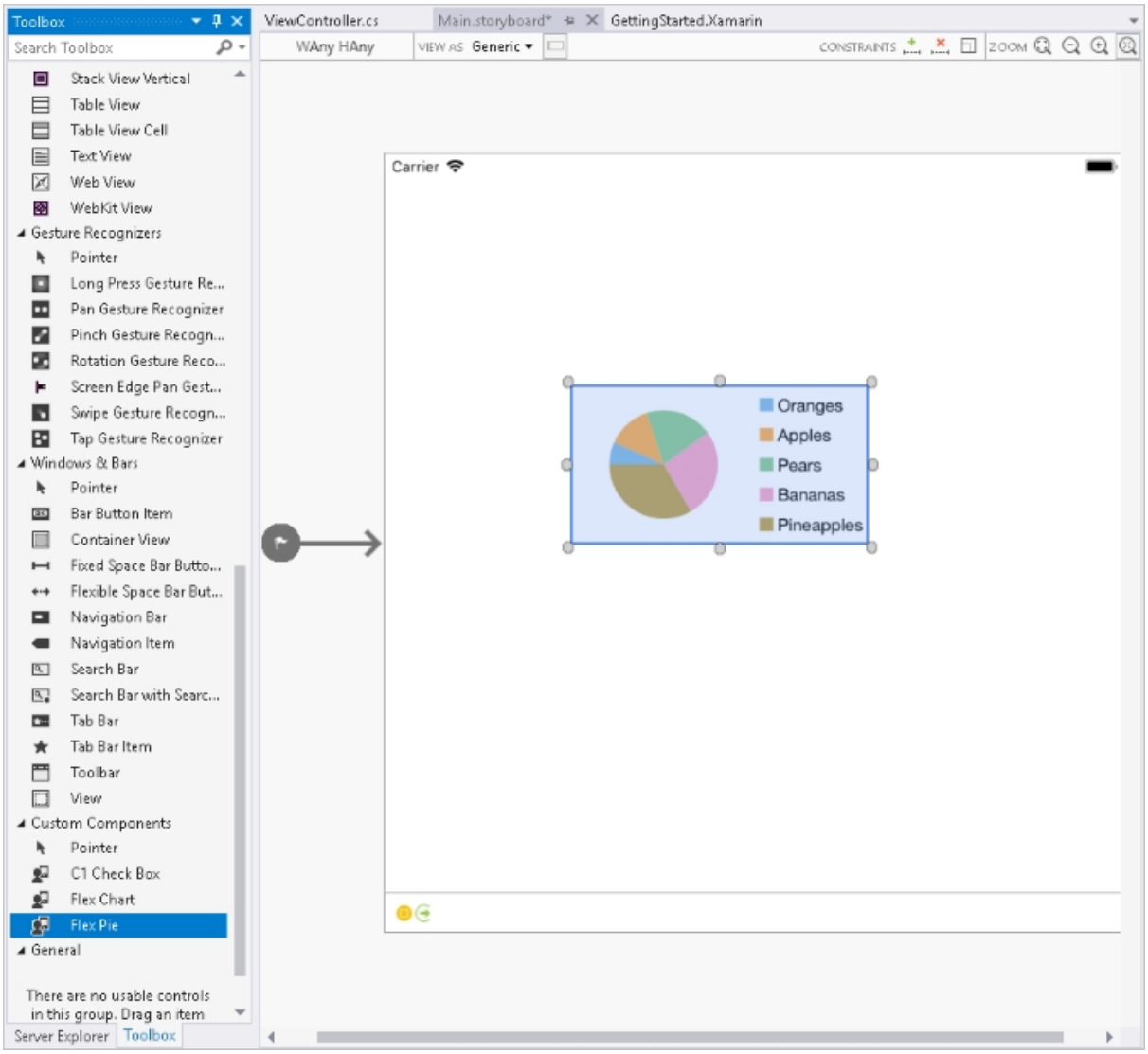
Back to Top

Step 2: Add a FlexPie control

Complete the following steps to initialize a FlexPie control in C#.

Add FlexPie control in StoryBoard

1. In the **Solution Explorer**, click **Main.storyboard** to open the storyboard editor.
2. From the **Toolbox** under **Custom Components** tab, drag a FlexPie onto the **ViewController**.



Initialize FlexPie control in Code

To initialize the FlexPie control, open the ViewController file from the Solution Explorer and replace its content with the code below. This overrides the **ViewDidLoad** method of the View controller in order to initialize FlexPie.

C#

```
public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            // Perform any additional setup after loading the view, typically from a nib.

            pieChart = new FlexPie();
            pieChart.Binding = "Value";
            pieChart.BindingName = "Name";
            pieChart.ItemsSource = PieChartData.DemoData();
            this.Add(pieChart);
        }

        public override void ViewDidLayoutSubviews()
        {
```

```
            base.ViewDidLayoutSubviews();

            pieChart.Frame = new CGRect (this.View.Frame.X, this.View.Frame.Y,
                                         this.View.Frame.Width, this.View.Frame.Height);
        }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

Animation

FlexPie allows you to enable animation effects using one of the two ways, either on loading when the chart is drawn or on updating when the chart is redrawn after modifications. It supports animation in charts through C1Animation class available in the C1.iOS.Core namespace.

You can also set the duration of the animation in chart using Duration property and interpolate the values of animation using Easing property of the C1Animation class, which accepts values from the C1Easing class. This class supports a collection of standard easing functions such as CircleIn, CircleOut, and Linear.

- **CircleIn:** Easing function that starts slow and speeds up in the form of a circle.
- **CircleOut:** Easing function that starts fast and slows down in the form of a circle.
- **Linear:** Easing function with constant speed.

C#

```
C1Animation animate = new C1Animation();
// set update animation duration
animate.Duration = new TimeSpan(1500 * 10000);
// interpolate the values of animation
animate.Easing = C1Easing.Linear;
```

In addition to easing functions of C1Easing class, FlexPie supports built in easing functions of Xamarin.Forms.Easing class. For more information, refer Xamarin Easing Class.

You can show animation while loading or updating a chart. To show animation while loading, use LoadAnimation property of the ChartBase class. This property gets the load animation from the object of C1Animation class to display the animation at the time of loading chart. Similarly, to animate the chart when underlying data collection changes on adding, removing, or modifying a value, you can use UpdateAnimation property of the ChartBase class.

C#

```
// set the loading animation easing
pieChart.LoadAnimation = animate;
```

You can apply the loading animation effect by setting the AnimationMode property which accepts values from the AnimationMode enumeration. This enumeration supports four different animation modes: All, None, Series, and Point.

- **All**: The chart expands outward from the center point.

- **None**: Does not display any animation.
- **Series**: The chart expands outward from the center point.
- **Point**: The chart radially animates clockwise around the center point.

C#

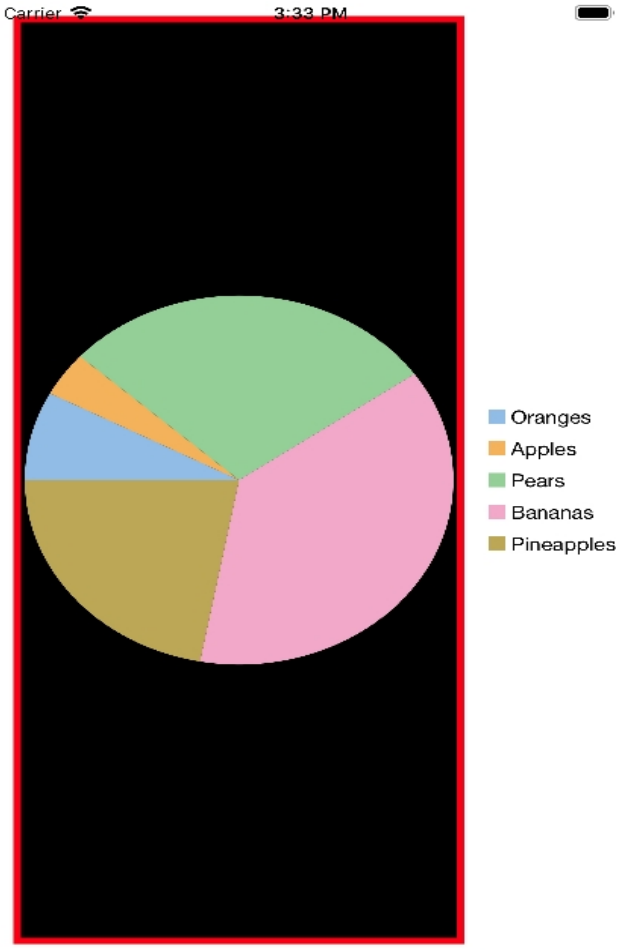
```
// set the animation mode
pieChart.AnimationMode = AnimationMode.Point;
```

Customize Appearance

Xamarin for iOS controls contain several properties to customize the appearance of the FlexPie control. They are designed to work with both: light and dark themes. You can also change the background color, the color and width of borders and the thickness of margins of FlexPie by simply setting the desired value in the following properties:

- **BackgroundColor**: Changes the background color of the entire chart.
- **PlotStyle.Fill**: Changes the color of the chart plot area.
- **PlotStyle.Stroke**: Changes the color of the chart plot area border.
- **PlotStyle.StrokeThickness**: Changes the width of the chart plot area border.

The image below shows a customized FlexPie control.



The following code example demonstrates how to set these properties. This example use the sample created in the Quick Start section.

CS

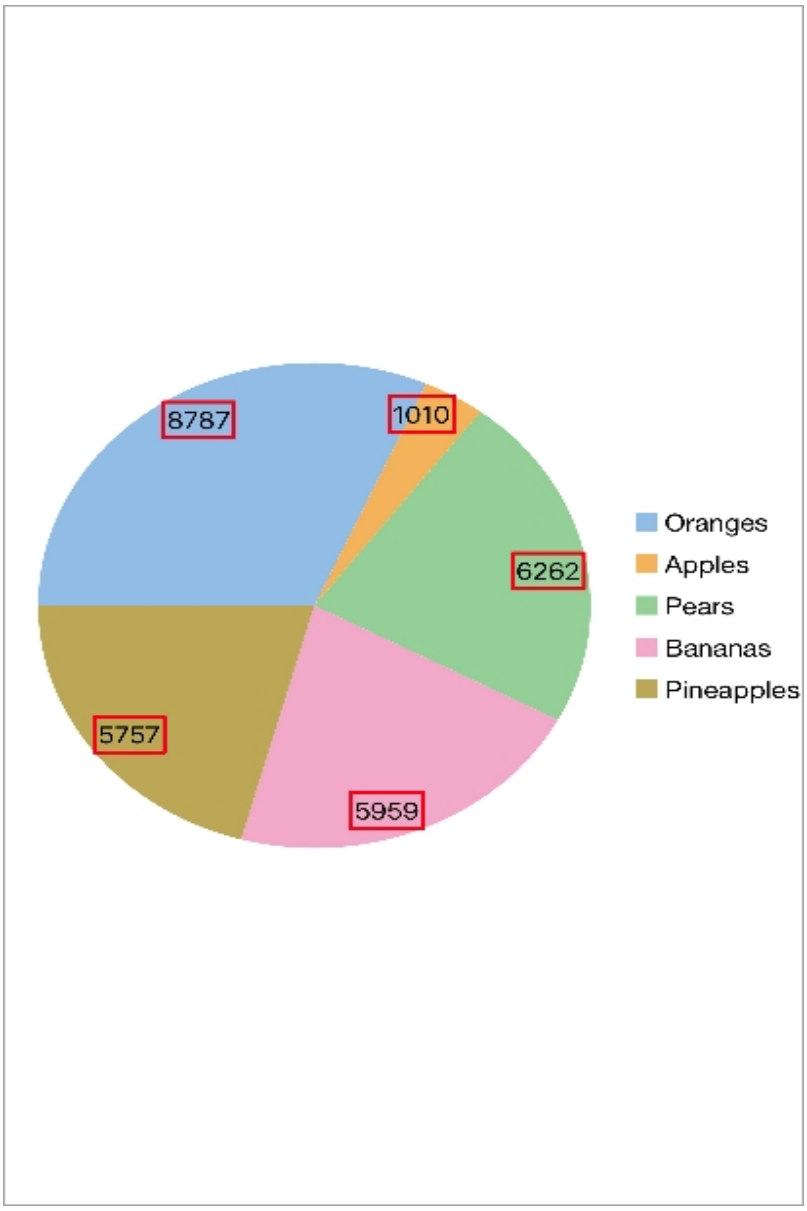
```
pieChart.PlotStyle.Fill = UIColor.Black;
pieChart.PlotStyle.Stroke = UIColor.Red;
pieChart.PlotStyle.StrokeThickness = 5;
```

Data Labels

You can add data labels in the FlexPie to show the exact values corresponding to each section of the pie. All you have to do is set the PieLabelPosition enumeration at design time and get the data labels displayed in the plot area. In FlexPie, users can choose to set data labels at the following positions.

- **None** - No data labels.
- **Inside**- Displays data labels within the pie.
- **Center**- Displays data labels right in the center of the pie.
- **Outside**- Displays data labels outside the pie.
- **Radial**- Displays data labels inside the pie slice and depends on its angle.
- **Circular** - Displays data labels inside the pie slice in circular direction.

The image given below shows a FlexChart control that displays sales data with data labels corresponding to months.



The following code example shows how to set data labels for FlexPie control. The example uses the sample created in the Quick Start section.

CS

```
pieChart.DataLabel.Position = PieLabelPosition.Inside;
pieChart.DataLabel.Content = "{y}";
pieChart.DataLabel.Border = true;
pieChart.DataLabel.BorderStyle = new ChartStyle { Stroke = UIColor.Red,
StrokeThickness = 2 };
pieChart.DataLabel.Style = new ChartStyle { FontFamily = UIFont.SystemFontOfSize(20,
UIFontWeight.Black) };
```

Manage Overlapped Data Labels

A common issue pertaining to charts is overlapping of data labels that represent data points. In most cases, overlapping occurs due to long text in data labels or large numbers of data points.

To manage overlapped data labels in FlexPie chart, you can make use of Overlapping property provided by PieDataLabel class. The Overlapping property accepts the following values from the PieLabelOverlapping enumeration.

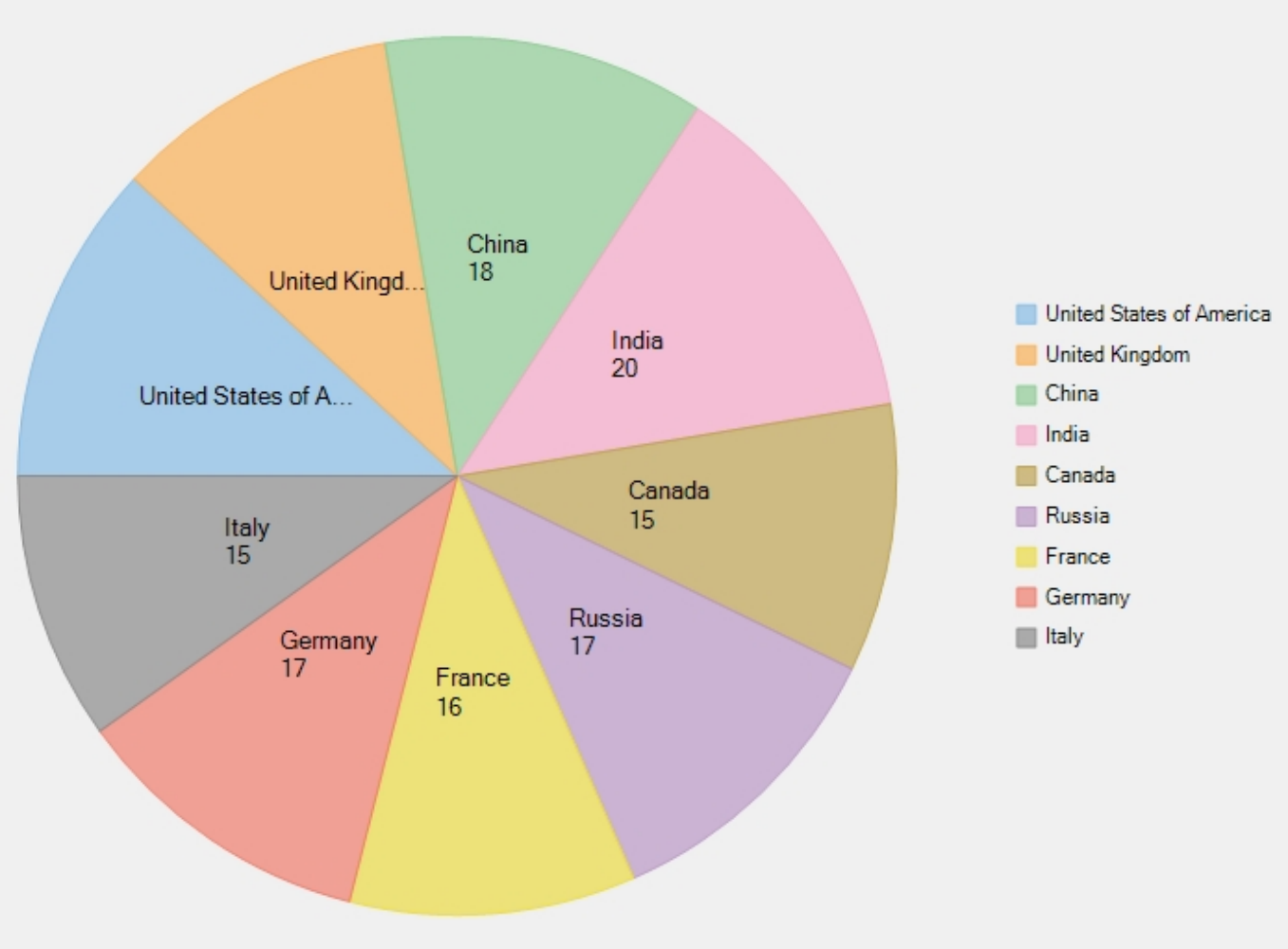
Enumeration	Description
PieLabelOverlapping.Default	Show all labels including the overlapping ones.
PieLabelOverlapping.Hide	Hides the overlapping labels, if its content is larger than the corresponding pie segment.
PieLabelOverlapping.Trim	Trim overlapping data labels, if its width is larger than the corresponding pie segment.

Use the following code to manage overlapping data labels.

C#

```
//Set Overlapping property
flexPie1.DataLabel.Overlapping = PieLabelOverlapping.Trim;
```

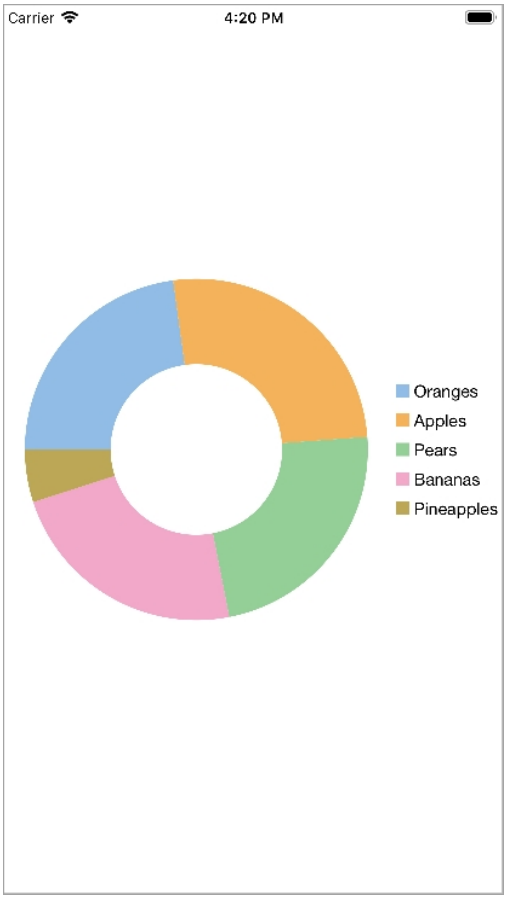
The following image shows FlexPie appears after setting the Overlapping property.



Donut Pie Chart

Set a blank inner space using the innerRadius property in FlexPie to create a donut pie chart. The blank space can be used to display additional data also.

The following image shows the donut FlexPie.



The following code example demonstrates how to set the innerRadius property. This example uses the sample created in the Quick Start section.

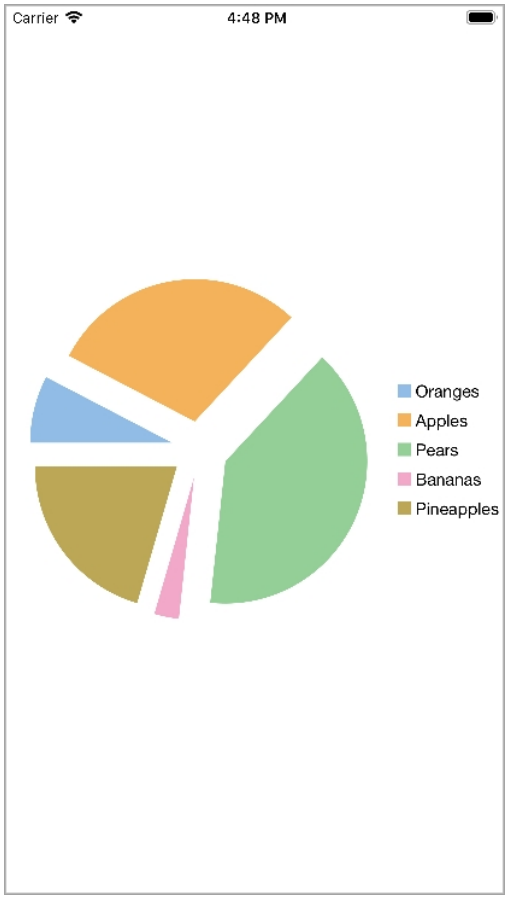
CS

```
pieChart.InnerRadius = 0.5;
```

Exploded Pie Chart

Push the pie slices away from the center of the FlexPie using the offset property and produce an exploded pie chart. This property accepts a decimal value to determine how far the pie slices are 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.

CS

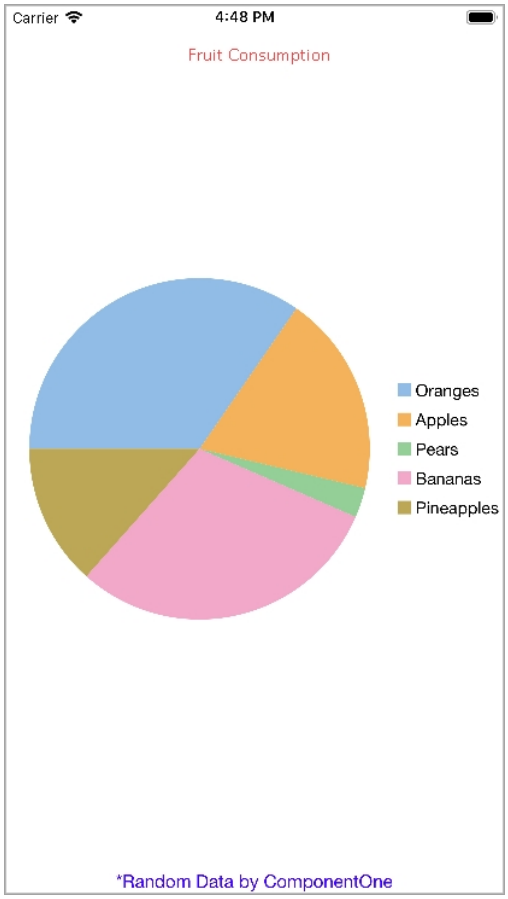
```
pieChart.Offset = 0.2;
```

Header and Footer

Add a title to the FlexPie control by setting its Header property. You may also set a footer by setting the Footer property. There are also some additional properties to customize header and footer text color in FlexPie:

- **HeaderStyle.Fill**: Lets you select the text color for the header text.
- **FooterStyle.Fill**: Lets you select the text color for the footer text.

The image below shows how FlexPie appears after these properties have been set.



The following code example demonstrates how to customize header and footer. This example uses the sample created in the Quick Start section.

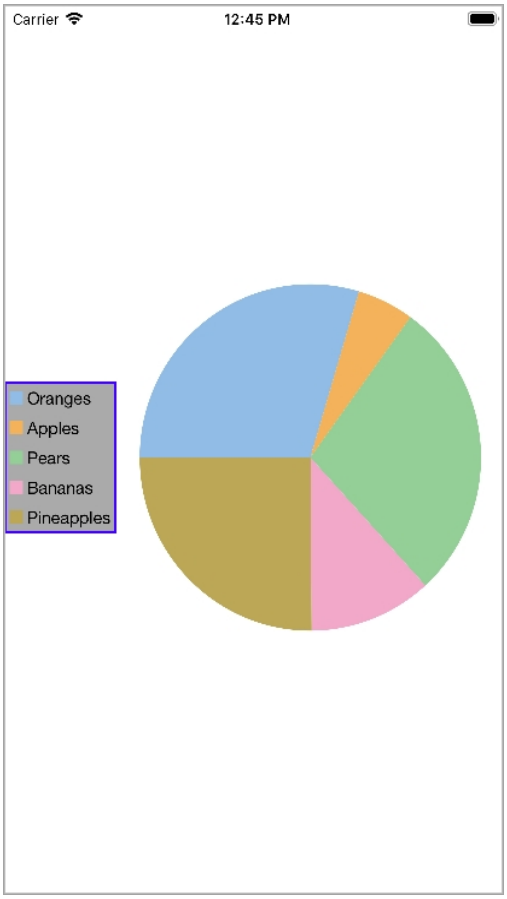
CS

```
pieChart.Header = "Fruit Consumption";
pieChart.HeaderStyle.Fill = UIColor.Red;

pieChart.Footer = "*Random Data by ComponentOne";
pieChart.FooterStyle.Fill = UIColor.Blue;
pieChart.FooterStyle.FontSize = 15;
```

Legend

You can customize the legend in the FlexPie using the borderColor, borderWidth, backgroundColor and labelTextColor properties. Additionally, you may use the position property to determine where to display the chart legend.

The image below shows customized legend in the FlexPie control.



The following code example demonstrates how to set these properties. This example uses the sample created in the Quick Start section.

CS

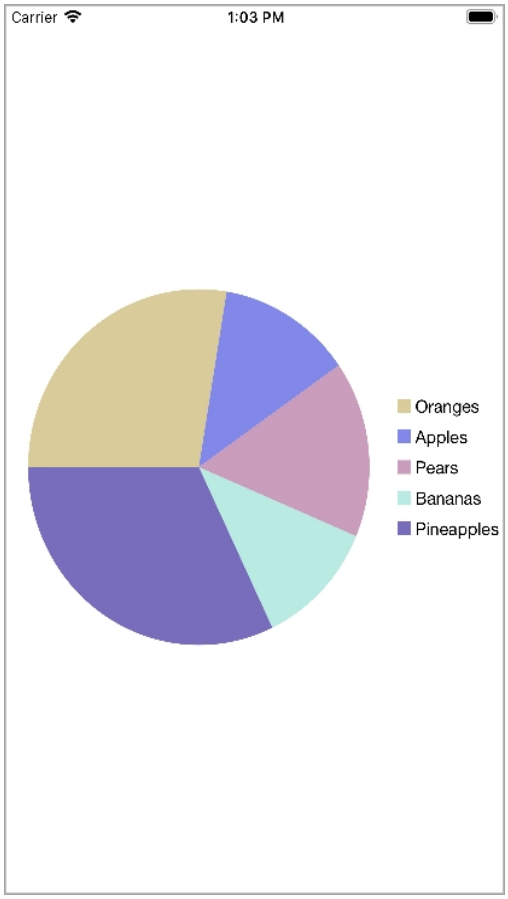
```
pieChart.LegendStyle.Stroke = UIColor.Blue;
pieChart.LegendStyle.StrokeThickness = 2;
pieChart.LegendStyle.Fill = UIColor.LightGray;
pieChart.LegendPosition = ChartPositionType.Left;
```

Themes

Enhance the appearance of the FlexPie control by using pre-defined themes. Use the Palette property to specify the theme you want to apply on the control.

Note: Remove the palette property from the code to revert to the default theme.

The image below shows how FlexPie appears when the palette property is set to **Light**.

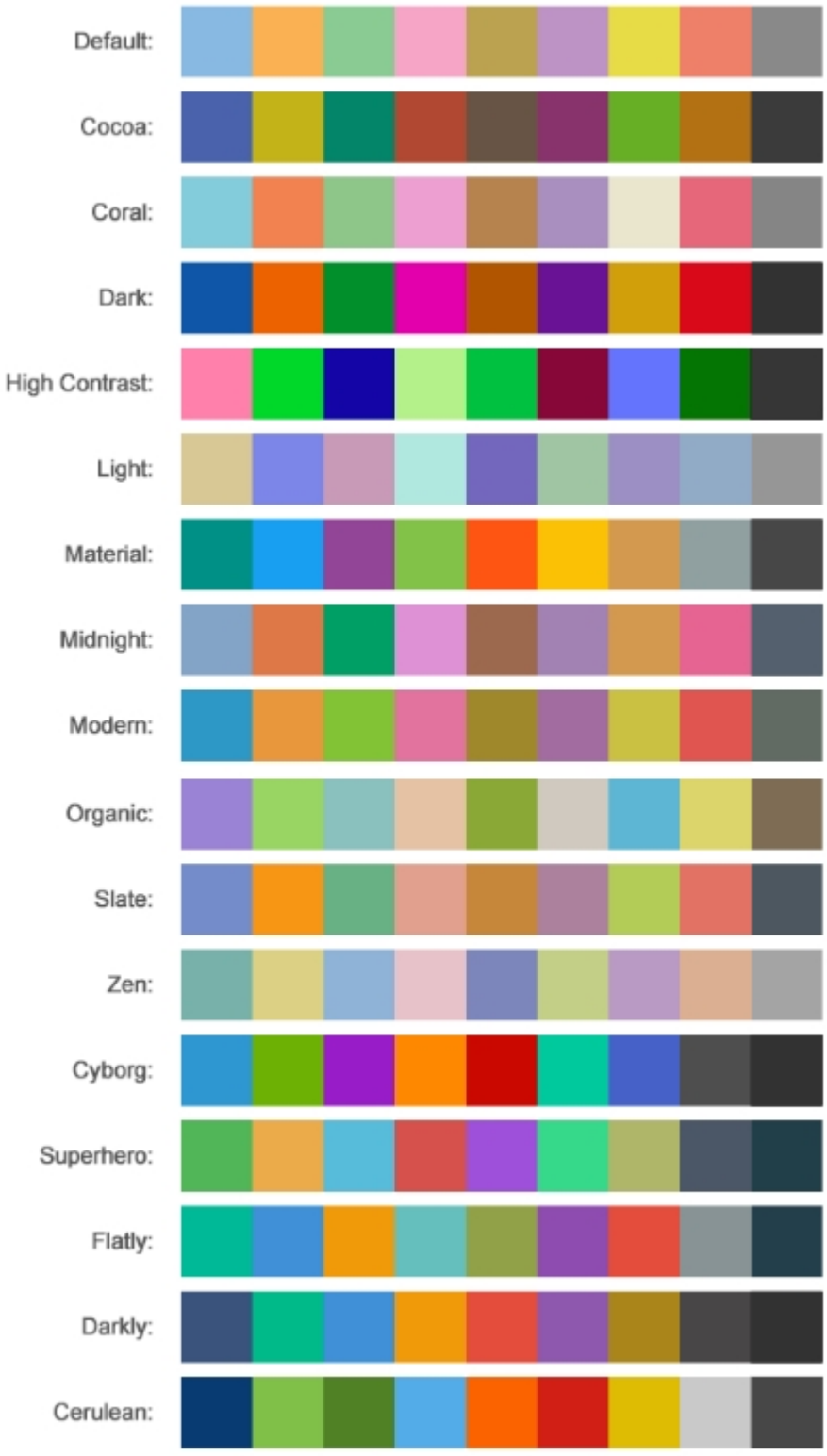


The following code example demonstrates how to set a theme. The example uses the sample created in the Quick Start section.

CS

```
pieChart.Palette = Palette.Light;
```

FlexPie comes with pre-defined templates that can be applied for quick customization. Here are the 17 pre-defined templates available in the C1.iOS.Chart.Palette enumeration.



Zooming and Panning

Zooming can be performed in FlexPie chart using ZoomBehavior class. To implement zooming, you need to create an object of ZoomBehavior class available in the C1.iOS.Chart.Interaction namespace and pass it as a parameter to the Add method. This method adds zoom behavior to the behavior collection by accessing it through Behaviors property of the ChartBase class.

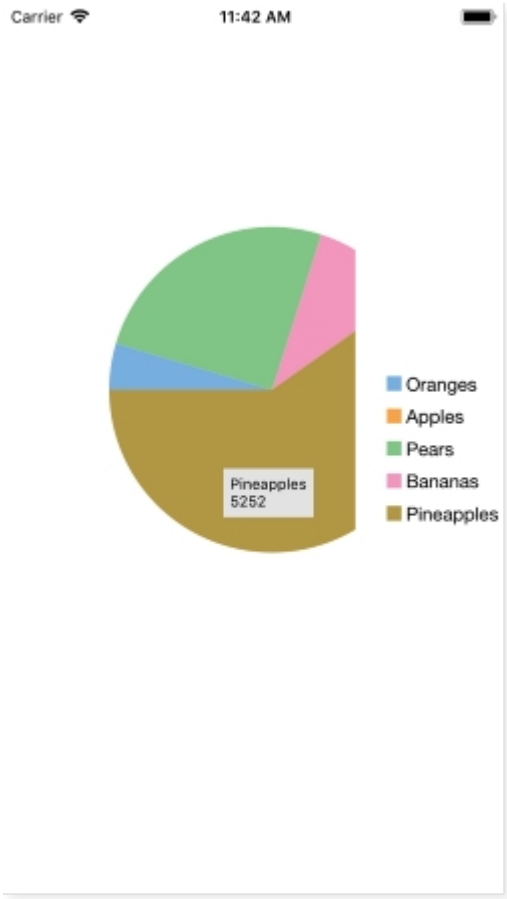
The following code examples demonstrate how to implement zooming in C#. These examples use the sample created in the Quick Start section.

C#

```
ZoomBehavior z = new ZoomBehavior();
pieChart.Behaviors.Add(z);
```

Similarly, panning can be implemented in FlexPie chart by creating an object TranslateBehavior class available in the C1.iOS.Chart.Interaction namespace and passing it as a parameter to the Add method. This method adds translation behavior to the behavior collection by accessing it through Behaviors property of the ChartBase class. In addition, you can use the TranslationX and TranslationY property of FlexPie class to set the translation x and translation y delta for the chart.

The image below shows how FlexPie appears on panning.



The following code examples demonstrate how to implement panning in C#. These examples use the sample created in the Quick Start section.

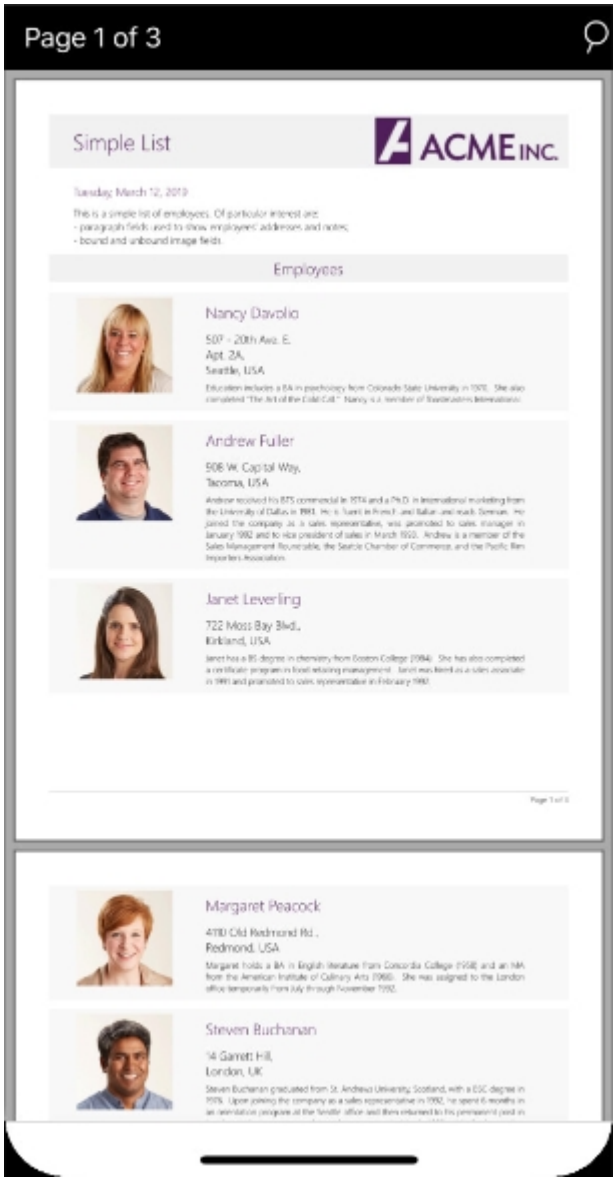
C#

```
TranslateBehavior t = new TranslateBehavior();
pieChart.Behaviors.Add(t);

pieChart.TranslationX = 10;
pieChart.TranslationY = 10;
```

Moreover, you can allow users to translate their custom views when pie is panning or zooming through TranslateCustomViews event handler available in FlexPie class.

FlexViewer

FlexViewer (Beta), as the name suggests, is a flexible, fast and powerful previewing control which comes with a modern, interactive and user-friendly UI. Currently, it uses GcPdf as its document source. FlexViewer allows you to view PDF documents, navigate through the PDF pages using page navigation option that let's you jump to a specific page by just typing specified page number in the Page textbox. In addition, it allows you to search text in the PDF document and zoom the PDF page from the UI itself, eliminating the need of writing any code.



Key Features

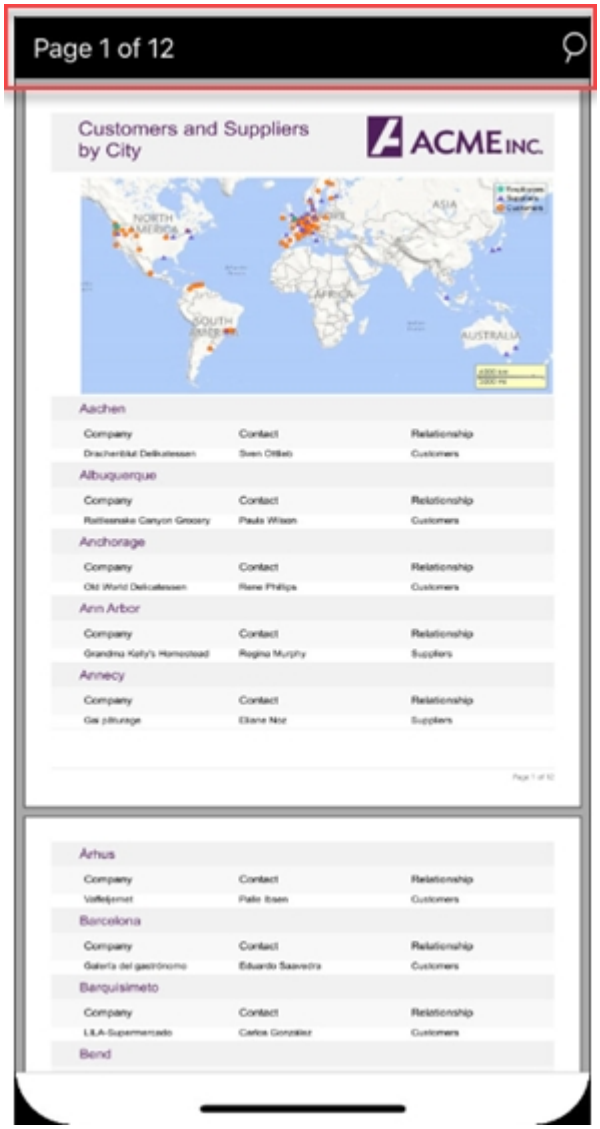
The key features of FlexViewer are as follows:

- **Modern user-friendly UI:** Interactive and user friendly UI that helps preview PDF documents with ease.
- **View PDF documents:** Load and display PDF document content, including images and shapes, in the FlexViewer control.
- **Page navigation:** Navigate to a specific page using the page label from the FlexViewer toolbar.
- **Search text:** Find text in document using Search toolbar and get highlighted search results.
- **Export PDF:** Save PDF documents to a file, stream or as images in different formats, such as BMP, JPEG, PNG, GIF, and TIFF.

- **Zoom:** Zoom in and out of the content of PDF document pages with ease.
- **Virtualize UI:** Open large documents without going out of memory.

FlexViewer Toolbar

FlexViewer toolbar appears at the top of the FlexViewer control with two options, a Page label on the left side and search icon on the right. Clicking the Page label transitions it to a textbox allowing you to navigate through the document pages and clicking the search icon opens a search toolbar replacing the main toolbar.



The toolbar consists of the following command buttons:

Command Button	Command Button Name	Description
Page 1 of 12	Page label	Displays the current page number and the total number of pages in the PDF document.
	Search	Allows you to search text in the PDF document.

Quick Start

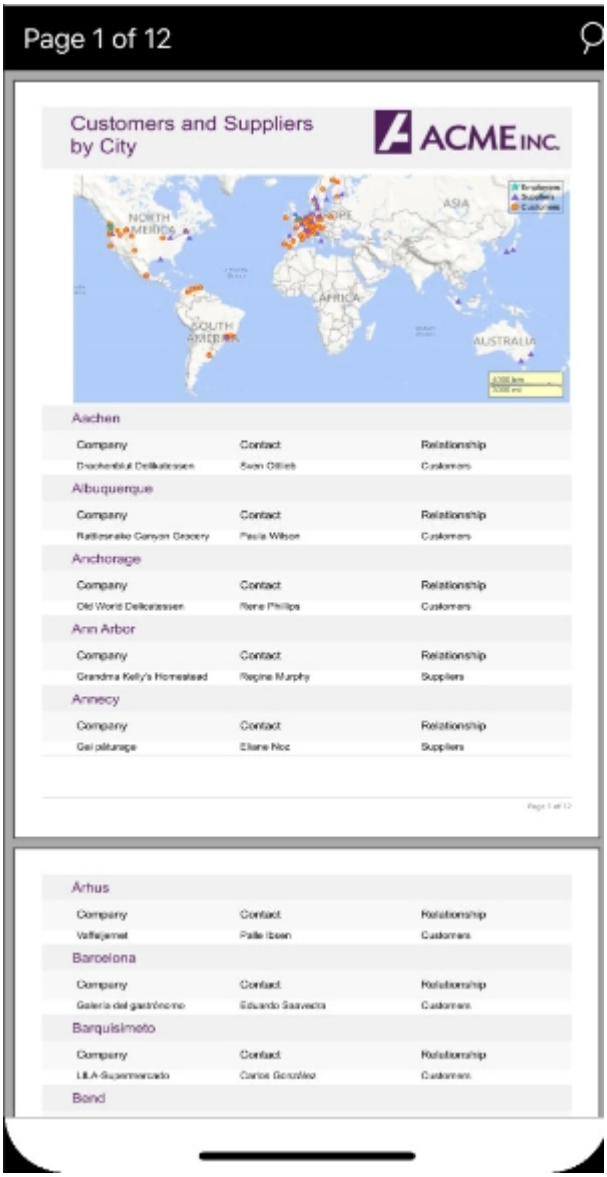
This quick start will guide you through the steps of adding FlexViewer control to the application, binding it with a document source, i.e., GcPdf, and loading the PDF in the FlexViewer control.

> Note: To use GcPdf as the document source for your application, you need to install GcPdf NuGet package to add the GcPdf references to your application.

To achieve it, follow these steps:

1. **Add FlexViewer control**
2. **Load the PDF document**
3. **Run the Project**

The following image shows how the FlexViewer control appears after completing the steps above.



Step 1: Add FlexViewer control

1. In the **Solution Explorer**, click **Main.storyboard** to open the storyboard editor.
2. From the **Toolbox** under **Custom Components** tab, drag a FlexPie onto the **ViewController**.

3. In the **Properties** window, set the Name of the FlexViewer control. In this example, we named it as flexViewer.

Back to Top

Step 2: Load the PDF document

Load the PDF document in the FlexViewer control using the following code. In this example, we have added a PDF document to the Data folder to load it in the viewer.

C# copyCode

```
string path = "Data/DefaultDocument.pdf";
MemoryStream ms = new MemoryStream();
{
    using (FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read))
    {
        byte[] bytes = new byte[fs.Length];
        fs.Read(bytes, 0, (int)fs.Length);
        ms.Write(bytes, 0, (int)fs.Length);
        flexViewer.LoadDocument(ms);
    }
}
```

Back to Top

Step 3: Run the Project

Press **F5** to run the application.

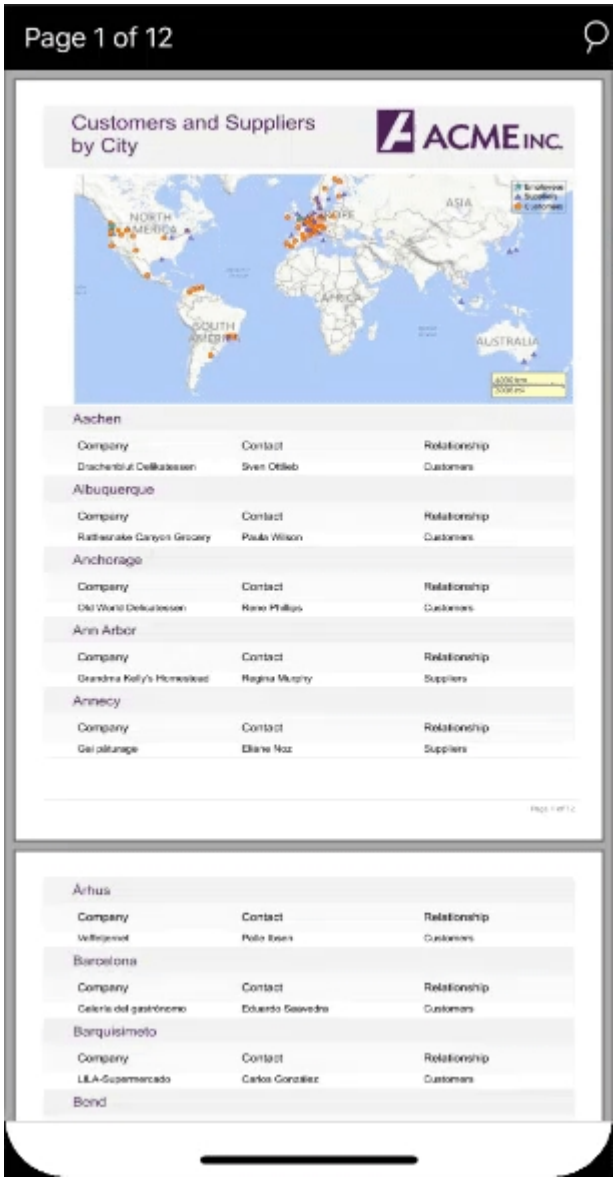
Back to Top

Features

Navigation

At times, depending on the document size, you might need to navigate through multiple pages in a document to view different sections in it. FlexViewer provides you with the interactive Page label that lets you move forward through the pages. On clicking the label, a text box appears allowing you to type a specific page number to jump to that particular page.

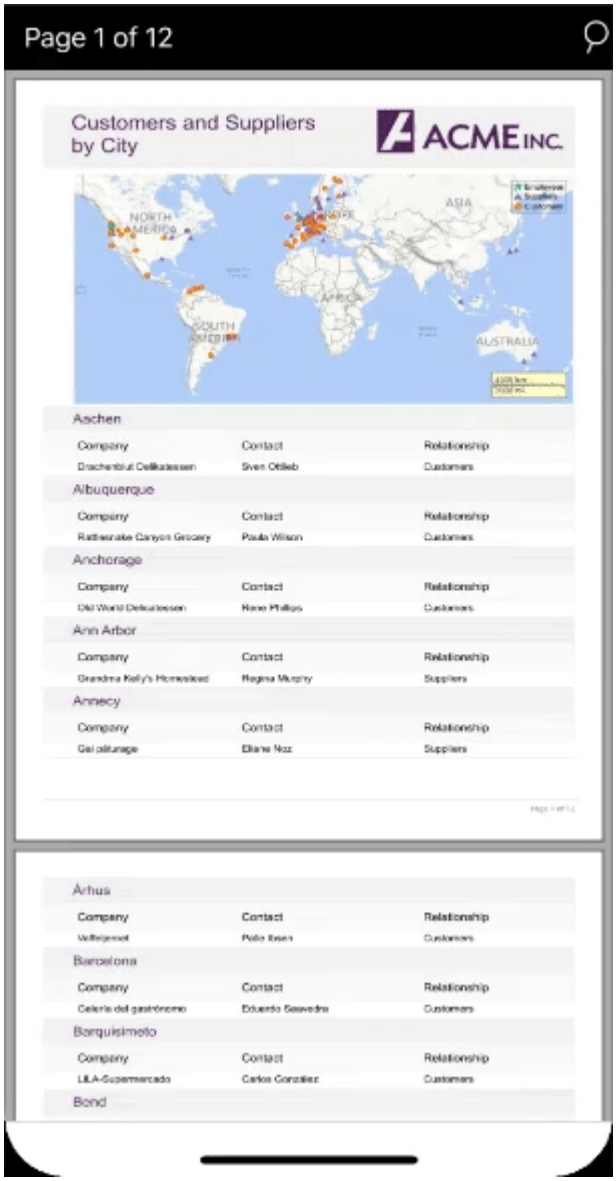
The following GIF depicts how you can navigate to a particular page using page navigation in FlexViewer.



Text Search

FlexViewer provides the flexibility to find text in a document. It provides a search icon at the top-right corner of the toolbar. On clicking the search icon, the search toolbar appears replacing the existing toolbar. In the search toolbar, you can simply type in the text and press Enter to search for it in the document. The FlexViewer control highlights all the matching instances making it easy for you to find all the occurrences of searched text in the document.

The following GIF depicts how text search can be performed in FlexViewer.



Appearance

Although Xamarin controls match the native controls on all three platforms by default and are designed to work with both, light and dark, themes available on all platforms. But, there are several properties specific to the FlexViewer control which can be used to customize the appearance of the loaded document and the area around it.
The FlexViewer class provides the following set of properties that can be used to change the overall look and feel of the document:

- **PageBackgroundColor**: Allows you to change the color for filling the page content.
- **PageBorderColor**: Provides a brush used for drawing page borders.
- **Padding**: Sets the amount of padding between pages and the preview window edges.
- **PageSpacing**: Sets the amount of padding between pages in the preview.

The following image shows FlexViewer with the customized appearance.

To change the appearance of the loaded PDF document, use the following code. This example uses the sample code created in Quick Start section.

C# copyCode

```
// Customize Appearance
flexViewer.PageBackgroundColor = UIColor.White;
flexViewer.BackgroundColor = UIColor.LightGray;
flexViewer.PageBorderColor = UIColor.Black;
flexViewer.Padding = new UIEdgeInsets(20, 20, 20, 20);
flexViewer.PageSpacing = 5;
```

Export

FlexViewer allows you to export a PDF document and save it to a file, stream or as an image(s). You can use Save method of the FlexViewer class to save a PDF document to a file or a stream. In addition, you can use various methods provided by the FlexViewer class to save a PDF document as an image. These methods are listed in the following table:

Methods	Description
SaveAsBmp	Saves the document pages as images in BMP format, i.e., one image for each page.
SaveAsGif	Saves the document pages as images in GIF format, i.e., one image for each page.
SaveAsJpeg	Saves the document pages as images in JPEG format, i.e., one image for each page.
SaveAsPng	Saves the document pages as images in PNG format, i.e., one image for each page.
SaveAsTiff	Saves the document pages as images in TIFF format, i.e., one image for each frame.

The following code illustrates how to export PDF pages to images. This example uses the sample code created in Quick Start section.

C# copyCode

```
string fullPath = null;
fullPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
"ExportedPdf" + "{0}") + "." + "PNG";
flexViewer.SaveAsPng(fullPath);
```

UI Virtualization

Virtualization is an optimization technique that renders page visuals only as they come into view. FlexViewer provides the ability to virtualize UI using low memory foot print to open large PDF files without going out of memory. This reduces the PDF loading time and improves performance, thus enhancing UI performance.

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.

Linear Gauge

Radial Gauge

Bullet Graph

Key Features

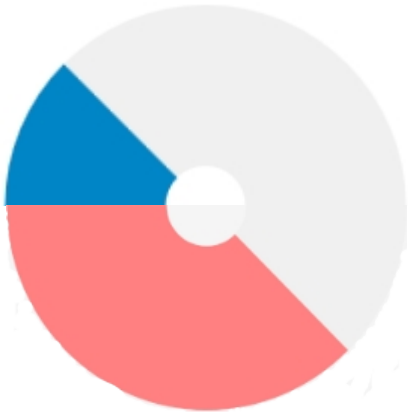
- **Easy Customization**: Restyle the Gauge by changing a property to create gauges with custom colors, fills and

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**: Place the LinearGauge and BulletGraph horizontally or vertically.
- **Pointer Customization**: Customize the pointer color, border, origin and more to make the Gauge more appealing.
- **Animation**: Use out-of-the-box animations to add effects to the Gauge control.
- **Drag Support:** Dragging the gauge with finger touch changes the value accordingly.

Gauge Types

C1Gauge comprises of three kinds of gauges: C1LinearGauge, C1RadialGauge and C1BulletGraph.

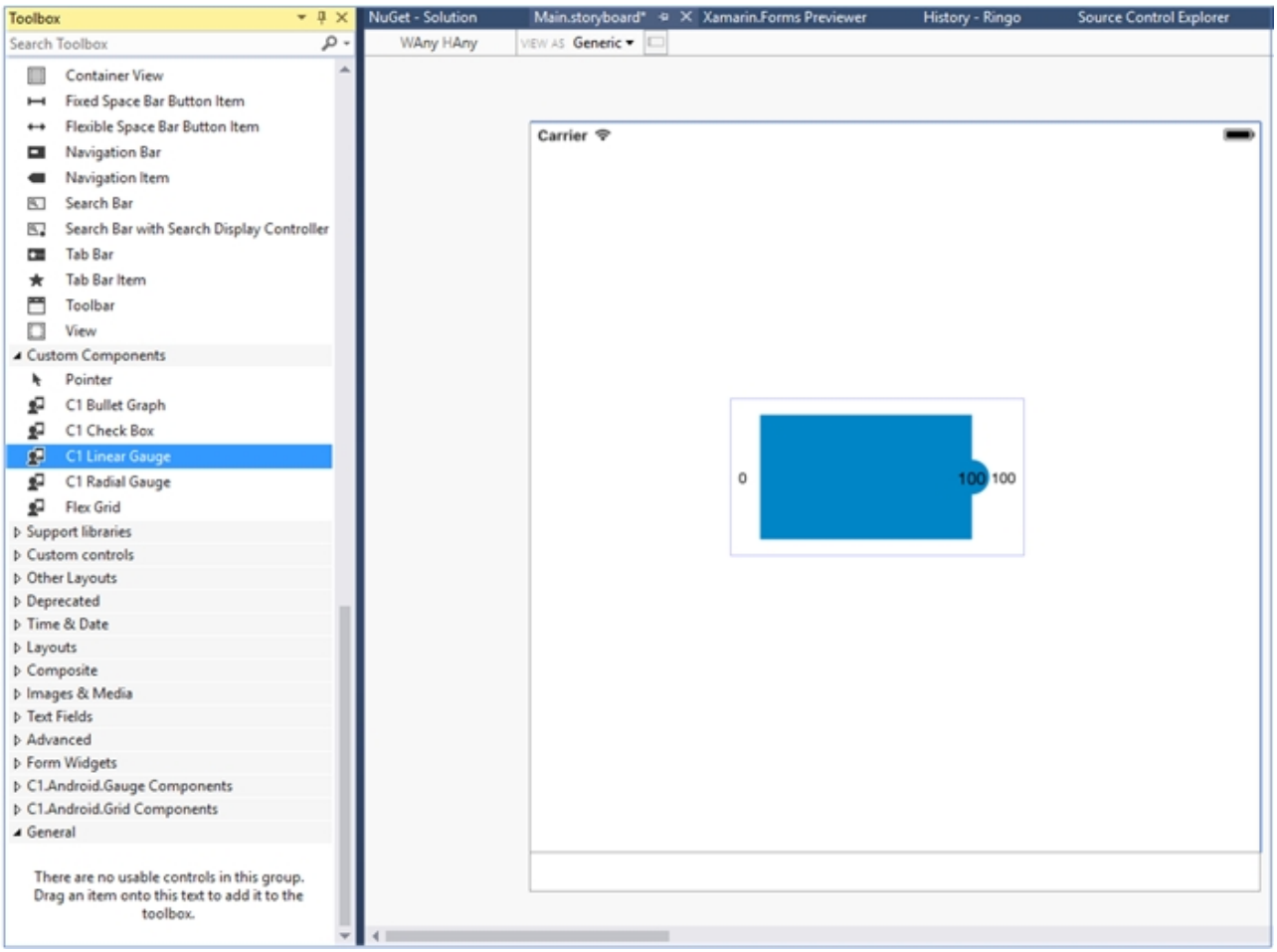
Type	Image	Usage
Linear Gauge: A linear gauge displays the value along a linear scale, using a linear pointer. The linear scale can be either horizontal or vertical, which can be set using the Direction property.		A linear gauge is commonly used to denote data as a scale value such as length, temperature, 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. It also provides IsReversed property to reorient the radial gauge so that it is drawn reversed (counter-clockwise).		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, bad or in some other state.		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 Gauge

This section describes how to add a C1Gauge controls to your iOS app and set its value.

Add C1Gauge control in StoryBoard

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. Under the **Document Outline**, expand View Controller and click **View**.
3. In the **Toolbox** under **Custom Components** tab, drag a C1LinearGuage, C1RadialGauge, or C1BulletGraph onto the View Controller.



Initialize C1Gauge control in code

To initialize C1Gauge control, open the ViewController file from the Solution Explorer and replace its content with the code below. This overrides the ViewDidLoad method of the View controller in order to initialize a C1LinearGauge, C1BulletGraph, and C1RadialGauge.

C#

```
private const double DefaultValue = 25;
private const double DefaultMin = 0;
private const double DefaultMax = 100;

C1LinearGauge linearGauge;
C1RadialGauge radialGauge;
C1BulletGraph bulletGraph;
```

```
public override void ViewDidLoad()
    {
            base.ViewDidLoad();
            this.EdgesForExtendedLayout = UIRectEdge.None;
            linearGauge = new C1LinearGauge();
            radialGauge = new C1RadialGauge();
            bulletGraph = new C1BulletGraph();

            linearGauge.Value = DefaultValue;
            linearGauge.Min = bulletGraph.Min = radialGauge.Min = DefaultMin;
            linearGauge.Max = bulletGraph.Max = radialGauge.Max = DefaultMax;
            linearGauge.Value = bulletGraph.Value = radialGauge.Value = DefaultValue;
            bulletGraph.Bad = 20;
            bulletGraph.Good = 75;
            bulletGraph.Target = 70;
            this.View.BackgroundColor = linearGauge.BackgroundColor =
bulletGraph.BackgroundColor = radialGauge.BackgroundColor = UIColor.White;
            this.Add(linearGauge);
            this.Add(radialGauge);
            this.Add(bulletGraph);
        }

        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            linearGauge.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
                        this.View.Frame.Width, this.View.Frame.Height/6);

            bulletGraph.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Height / 3,
                        this.View.Frame.Width, this.View.Frame.Height / 6);

            radialGauge.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Height * 2 /
3, this.View.Frame.Width, this.View.Frame.Height/3);

        }
```

Quick Start: Add and Configure

BulletGraph Quick Start

This section describes how to add a C1BulletGraph control to your iOS app and set its value. For information on how to add Xamarin controls, see Creating a New Xamarin.iOS App.

This topic comprises of three steps:

- **Step 1: Add a BulletGraph control**
- **Step 2: Run the Application**

The following image shows how the BulletGraph appears, after completing the steps above.

Step 1: Add a BulletGraph control

The value property denotes the value of the Gauge. Multiple ranges can be added to a single Gauge and the position of the range is defined by the min and max properties of the range. If you set the Gauge's isReadOnly property to false, then the user is be able to edit the value by tapping on the Gauge.

> **Note**: The origin property can be used to change the origin of the BulletGraph pointer. By default, the origin is set to 0.

Complete the following steps to initialize a BulletGraph control.

Add a BulletGraph control in View

1. In the **Project Navigator**, click `MainStoryboard` to open the storyboard editor.
2. In the right-most pane of the storyboard editor, click icon in the toolbar to open the **Identity inspector**.
3. Under **Custom Class**, change the class from **UI View** to **BulletGraph** using drop-down.

Initialize the BulletGraph control in Code

To initialize the BulletGraph control, open the ViewController.cs file from the Project Navigator and replace its content with the code below. This overrides the **viewDidLoad** method of the View controller in order to initialize BulletGraph.

C#

```
using C1.iOS.Gauge;
using CoreGraphics;
using System;
using UIKit;
namespace GaugeTest
{
    public partial class ViewController : UIViewController
    {
        public ViewController(IntPtr handle) : base(handle)
        {        }
        private const double DefaultValue = 25;
        private const double DefaultMin = 0;
        private const double DefaultMax = 100;

        C1BulletGraph bulletGraph;
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            this.EdgesForExtendedLayout = UIRectEdge.None;
            bulletGraph = new C1BulletGraph();
            bulletGraph.Bad = 20;
            bulletGraph.Good = 75;
```

```
            bulletGraph.Target = 70;
            View.BackgroundColor = bulletGraph.BackgroundColor = UIColor.White;
            this.Add(bulletGraph);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            bulletGraph.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Height
/ 3,
                        this.View.Frame.Width, this.View.Frame.Height / 6);
        }
    }
}
```

Back to Top

Step 2: Run the Application

Press **F5** to run the application.

Back to Top

LinearGauge Quick Start

This section describes how to add a LinearGauge control to your iOS app and set its value. For information on how to add Xamarin controls, see Creating a New Xamarin.iOS App.

This topic comprises of two steps:

- **Step 1: Add a LinearGauge control**
- **Step 2: Run the Application**

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

Step 1: Add a LinearGauge control

The value property denotes the value of the gauge. Multiple ranges can be added to a single Gauge and the position of the range is defined by the min and max properties of the range. If you set the Gauge's isReadOnly property to false, then the user is able to edit the value by tapping on the gauge.

> **Note:** The origin property can be used to change the origin of the Gauge pointer. By default, the origin is set to 0.

Complete the following steps to initialize a LinearGauge control in C#.

Add a LinearGauge control in View

1. In the **Project Navigator**, click `MainStoryboard` to open the storyboard editor.
2. In the right-most pane of the storyboard editor, click icon in the toolbar to open the **Identity inspector**.
3. Under **Custom Class**, change the class from **UI View** to **LinearGauge** using drop-down.

Initialize the LinearGauge control in Code

To initialize the **LinearGauge** control, open the ViewController.cs file from the Project Navigator and replace its content with the code below. This overrides the **viewDidLoad** method of the View controller in order to initialize LinearGauge.

C#

```
using C1.iOS.Gauge;
using CoreGraphics;
using System;
using UIKit;
namespace GaugeTest
{
    public partial class ViewController : UIViewController
    {
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        private const double DefaultValue = 25;
        private const double DefaultMin = 0;
        private const double DefaultMax = 100;

        C1LinearGauge linearGauge;
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            this.EdgesForExtendedLayout = UIRectEdge.None;
            linearGauge = new C1LinearGauge();
            linearGauge.Value = DefaultValue;
            linearGauge.Thickness = 0.1;
            linearGauge.Min = DefaultMin;
            linearGauge.Max = DefaultMax;
                        linearGauge.ShowRanges = true;
            linearGauge.PointerColor = UIColor.Blue;
            this.View.BackgroundColor = linearGauge.BackgroundColor = UIColor.White;
            GaugeRange low = new GaugeRange();
            GaugeRange med = new GaugeRange();
            GaugeRange high = new GaugeRange();

            //Customize Ranges
            low.Color = UIColor.Red;
            low.Min = 0;
            low.Max = 40;
            med.Color = UIColor.Yellow;
            med.Min = 40;
            med.Max = 80;
            high.Color = UIColor.Green;
            high.Min = 80;
            high.Max = 100;

            //Add Ranges to Gauge
```

```
            linearGauge.Ranges.Add(low);
            linearGauge.Ranges.Add(med);
            linearGauge.Ranges.Add(high);
            this.Add(linearGauge);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            linearGauge.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
                        this.View.Frame.Width, this.View.Frame.Height / 6);
        }

    }
}
```

Back to Top

Step 2: Run the Application

Press **F5** to run the application.

Back to Top

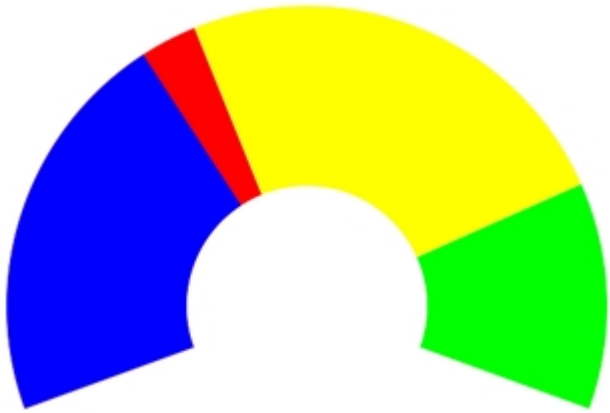
RadialGauge Quick Start

This section describes how to add a C1RadialGauge control to your iOS app and set its value. For information on how to add Xuni controls in, see Creating a New Xamarin.iOS App.

This topic comprises of two steps:

- **Step 1: Add a RadialGauge control**
- **Step 2: Run the Application**

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



Step 1: Add a RadialGauge control

The value property denotes the value of the Gauge. Multiple ranges can be added to a single Gauge and the position of the range is defined by the min and max properties of the range. If you set the Gauge's isReadOnly property to false, then the user is be able to edit the value by tapping on the Gauge.

The StartAngle property specifies the RadialGauge's starting angle and the SweepAngle property specifies an angle representing the length of the RadialGauge's arc. These properties can be used to specify the start and end points of the radial gauge arc. The angle for both properties are measured clockwise, starting at the 9 o'clock position. When the SweepAngle is negative is formed counter clockwise.

RadialGauge also offers an autoScale property. When this property is set to true, the RadialGauge is automatically scaled to fill its containing element.

> **Note**: The Origin property can be used to change the origin of the Gauge pointer. By default, the origin is set to 0.

Complete the following steps to initialize a RadialGauge control in C#.

Add a RadialGauge control in View

1. In the **Project Navigator**, click `MainStoryboard` to open the storyboard editor.
2. In the right-most pane of the storyboard editor, click  icon in the toolbar to open the **Identity inspector**.
3. Under **Custom Class**, change the class from **UI View** to **RadialGauge** using drop-down.

Initialize the RadialGauge control in Code

To initialize the RadialGauge control, open the ViewController.m or ViewController.swift file from the Project Navigator and replace its content with the code below. This overrides the **viewDidLoad** method of the View controller in order to initialize RadialGauge.

Example Title

```
using C1.iOS.Gauge;
using CoreGraphics;
using System;
using UIKit;
namespace GaugeTest
{
    public partial class ViewController : UIViewController
    {
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        private const double DefaultValue = 25;
        private const double DefaultMin = 0;
        private const double DefaultMax = 100;

        C1RadialGauge radialGauge;
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            this.EdgesForExtendedLayout = UIRectEdge.None;
            radialGauge = new C1RadialGauge();
            radialGauge.Value = 35;
            radialGauge.Min = 0;
            radialGauge.Max = 100;
            radialGauge.StartAngle = -20;
            radialGauge.SweepAngle = 220;
```

```
            radialGauge.AutoScale = true;
            radialGauge.ShowText = GaugeShowText.None;
            radialGauge.PointerColor = UIColor.Blue;
            radialGauge.ShowRanges = true;
            View.BackgroundColor = radialGauge.BackgroundColor = UIColor.White;

            GaugeRange low = new GaugeRange();
            GaugeRange med = new GaugeRange();
            GaugeRange high = new GaugeRange();

            //Customize Ranges
            low.Color = UIColor.Red;
            low.Min = 0;
            low.Max = 40;
            med.Color = UIColor.Yellow;
            med.Min = 40;
            med.Max = 80;
            high.Color = UIColor.Green;
            high.Min = 80;
            high.Max = 100;

            //Add Ranges to Gauge
            radialGauge.Ranges.Add(low);
            radialGauge.Ranges.Add(med);
            radialGauge.Ranges.Add(high);
            this.Add(radialGauge);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            radialGauge.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
            this.View.Frame.Width, this.View.Frame.Height / 6);
        }
    }
}
```

Back to Top

Step 2: Run the Application

Press **F5** to run the application.

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. This example uses the sample created in the LinearGauge Quick Start section.

In Code

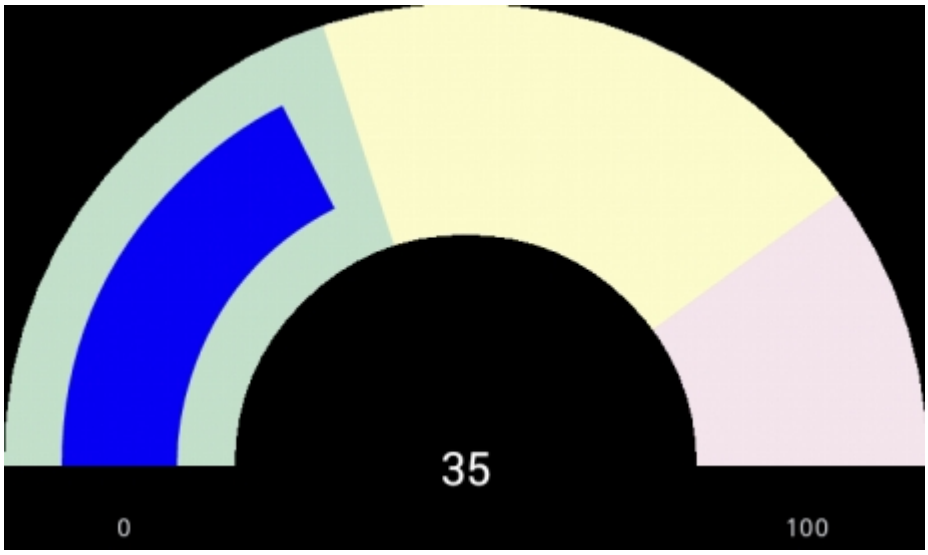
C#

```
linearGauge.IsAnimated = true;
linearGauge.LoadAnimation.Duration = new TimeSpan(0, 2, 0);
linearGauge.UpdateAnimation.Duration = new TimeSpan(0, 2, 0);
```

Customize Appearance

Xamarin for iOS controls are designed to work with both light and dark themes. But, there are several properties available that allows you customize the gauge elements to make it visually attractive. The following code example demonstrates how to set different styling properties to customize a RadialGauge.

The following image shows how the RadialGauge appears after its appearance is customized.



The following code example demonstrates how to customize the gauge appearance. The example uses the sample created in the RadialGauge Quick Start section.

In Code

Example Title

```
radialGauge.ShowText = GaugeShowText.All;
radialGauge.Thickness = 0.5;
radialGauge.FaceBorderColor = UIColor.Gray;
radialGauge.FaceBorderWidth = 5;
radialGauge.MinTextColor = UIColor.Green;
radialGauge.MinTextColor = UIColor.FromRGBA(255, 0, 255, 1);
```

Direction

The Direction property allows you to change the orientation of the gauge as well as the direction in which the pointer moves. For example, setting the direction to Down changes the gauge orientation from horizontal to vertical and the pointer starts from the top of the gauge and move towards the bottom.

The following image shows how the C1LinearGauge appears after this property has been set.

The following code example demonstrates how to set direction. This example uses the sample created in the LinearGauge Quick Start section.

In Code

C#

```
linearGauge.Direction = LinearGaugeDirection.Right;
```

Export Image

C1Gauge allows you to export your gauge in the form of an image and save it to the Camera Roll of your iOS device, or you can select your preferred location as per the requirement. It works exactly how you take a screen shot on your iPhones and iPads. The example below adds a snapshot button on the top of the XuniGauge. Users can tap the button to save the chart in the gallery.

In Code

C#

```
using C1.iOS.Core;
using C1.iOS.Gauge;
using CoreGraphics;
using Foundation;
```

```
using System;
using UIKit;
namespace GaugeTest
{
    public partial class ViewController : UIViewController
    {
        C1RadialGauge RadialGauge = new C1RadialGauge();
        public ViewController(IntPtr handle) : base(handle)
        {
        }

        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            // Perform any additional setup after loading the view, typically from a
nib.
        }

        public override void DidReceiveMemoryWarning()
        {
            base.DidReceiveMemoryWarning();
            // Release any cached data, images, etc that aren't in use.
        }

        partial void DoSnapshot(UIBarButtonItem sender)
        {
            SaveImageToDisk();
        }

        async void SaveImageToDisk()
        {
            var gaugeImage = new UIImage(NSData.FromArray(await
RadialGauge.GetImage()));
            gaugeImage.SaveToPhotosAlbum((image, error) =>
            {
                string message, title;

                if (error == null)
                {
                    title =
Foundation.NSBundle.MainBundle.LocalizedString("ImageSavedTitle", "");
                    message =
Foundation.NSBundle.MainBundle.LocalizedString("ImageSavedDescription", "");

                }
                else
                {
                    title =
Foundation.NSBundle.MainBundle.LocalizedString("PermissionDenied", "");
                    message = error.Description;
                }
```

```
                UIAlertView alert = new UIAlertView(title, message, null, "OK",
null);
                alert.Show();
            });
        }

    }
}
```

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 example demonstrates how to add ranges to a Gauge and set its properties.

In Code

Create new instances of type GaugeRange, set their properties and add the newly created ranges to the LinearGauge (or RadialGauge/BulletGraph).

C#

```
GaugeRange low = new GaugeRange();
GaugeRange med = new GaugeRange();
GaugeRange high = new GaugeRange();

//Customize Ranges
low.Color = UIColor.Red;
low.Min = 0;
low.Max = 40;
med.Color = UIColor.Yellow;
med.Min = 40;
med.Max = 80;
high.Color = UIColor.Green;
high.Min = 80;
high.Max = 100;

//Add Ranges to Gauge
linearGauge.Ranges.Add(low);
linearGauge.Ranges.Add(med);
linearGauge.Ranges.Add(high);
this.Add(linearGauge);
```

Input

AutoComplete

The C1AutoComplete is an editable input control designed to show possible text suggestions automatically as the user types text. The control filters a list of pre-defined items dynamically as a user types to provide suggestions that best or completely matches the input. The suggestions that match the user input appear instantly in a drop-down list.

Key Features

- **Customize Appearance** - Use basic appearance properties to customize the appearance of the drop-down list.
- **Delay** - Use delay feature to provide some time gap (in milliseconds) between user input and suggestion.
- **Highlight Matches** - Highlight the input text with matching string in the suggestions.

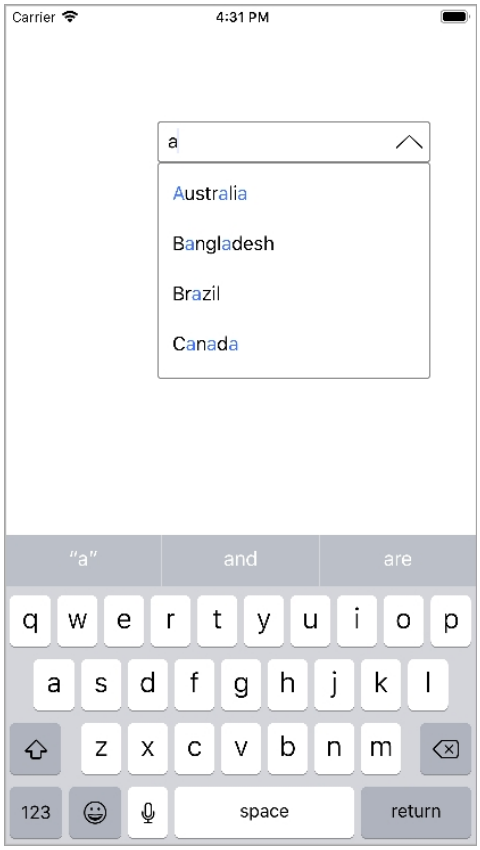
Quick Start: Populating C1AutoComplete with data

This section describes adding a C1AutoComplete control to your iOS application and populating it with data. The data is shown as a list in the drop-down part of the control.

Complete the following steps to display a C1AutoComplete control.

- **Step 1: Create a data source**
- **Step 2: Add a C1AutoComplete control to ViewController**
- **Step 3: Run the project**

The following image shows a C1AutoComplete control displaying input suggestions as the user types.



Step 1: Create a data source

Add a new class to the application that serves as the data source for C1AutoComplete.

C#

```
class Countries : NSObject
    {
```

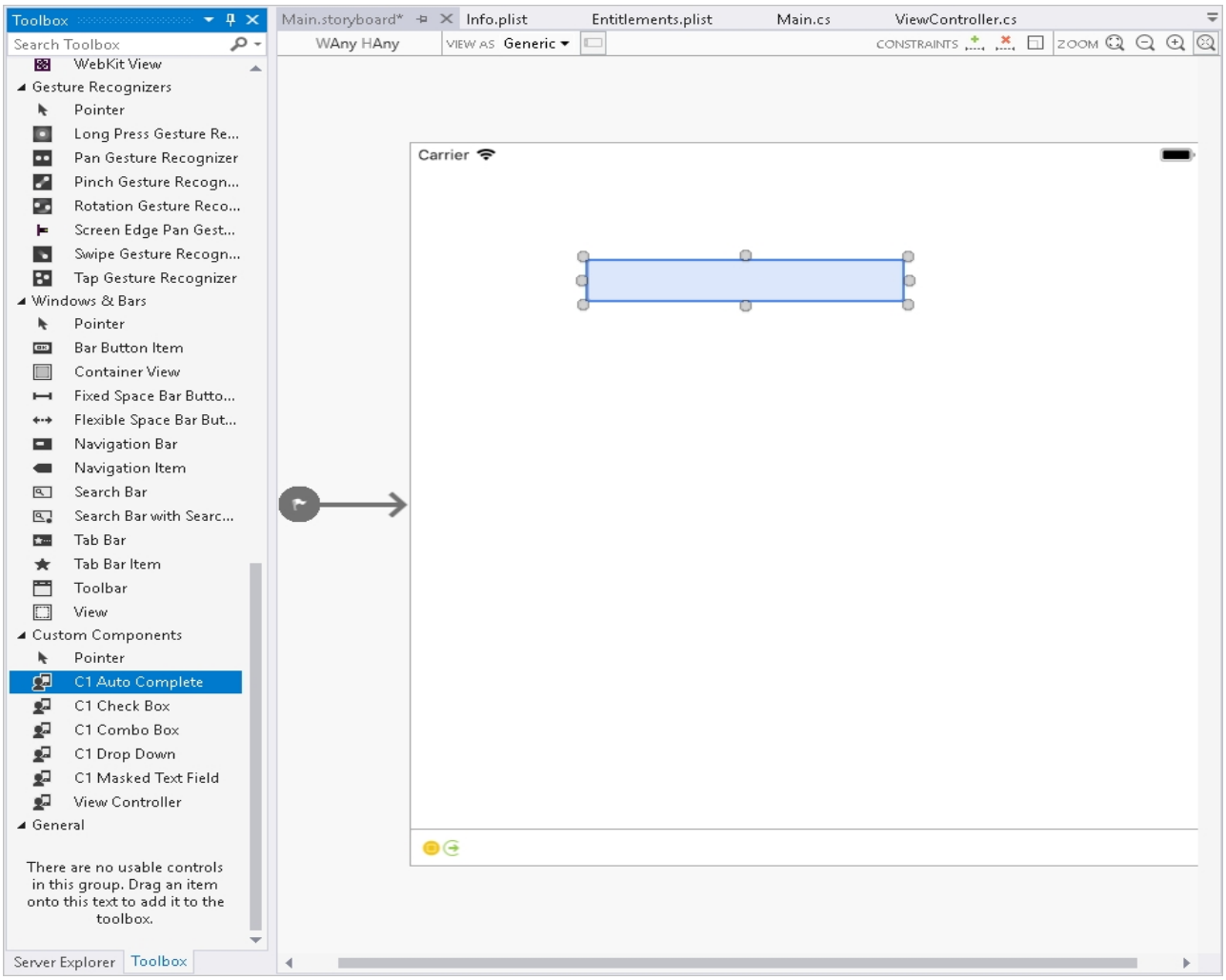
```
        [Export("Name")]
        public string Name { get; set; }

        public Countries()
        {
            this.Name = string.Empty;
        }

        public Countries(string name)
        {
            this.Name = name;
        }

        public static IEnumerable<object> GetDemoDataList()
        {
            List<object> array = new List<object>();
            var quarterNames = "Australia,Bangladesh,Brazil,Canada,China".Split(',');

            for (int i = 0; i < quarterNames.Length; i++)
            {
                array.Add(new Countries
                {
                    Name = quarterNames[i]
                });
            }
            return array as IEnumerable<object>;
        }
    }
```

Back to Top

Step 2: Add a C1AutoComplete control to ViewController

Add C1AutoComplete control to StoryBoard

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. From the Toolbox under the **Custom Components** tab, drag the C1AutoComplete control onto the ViewController.



3. In the **Properties** window, set the **Name** of the control as HighlightDropdown.

Initialize C1AutoComplete in code

To initialize C1AutoComplete control, open the ViewController file from the Solution Explorer and replace its content with the code below. This overrides the ViewDidLoad method of the View controller in order to initialize C1AutoComplete.

C#

```
public override void ViewDidLoad()
{
   base.ViewDidLoad();

   HighlightDropdown.DropDownHeight = 200;
   HighlightDropdown.DisplayMemberPath = "Name";
   HighlightDropdown.IsAnimated = true;
   HighlightDropdown.ItemsSource = Countries.GetDemoDataList();
}
```

Back to Top

Step 3: Run the Project

Press **F5** to run your application

Back to Top

Features

Data Binding

The **C1AutoComplete** control uses ItemsSource and DisplayMemberPath properties to bind the control to data. The ItemsSource property lets you bind the control to an enumerable collection of items, and the DisplayMemberPath property sets the path to a value on the source object for displaying data.

The following code illustrates how to set these properties in code to achieve data binding.

C#

```
HighlightDropdown.DisplayMemberPath = "Name";
HighlightDropdown.ItemsSource = Countries.GetDemoDataList();
```

Delay

The C1AutoComplete control provides instant text suggestions by searching the best possible match for the user input. However, you can change this default behavior and add some time gap between user input and the search. For this, you need to use the Delay property and set time delay in milliseconds.

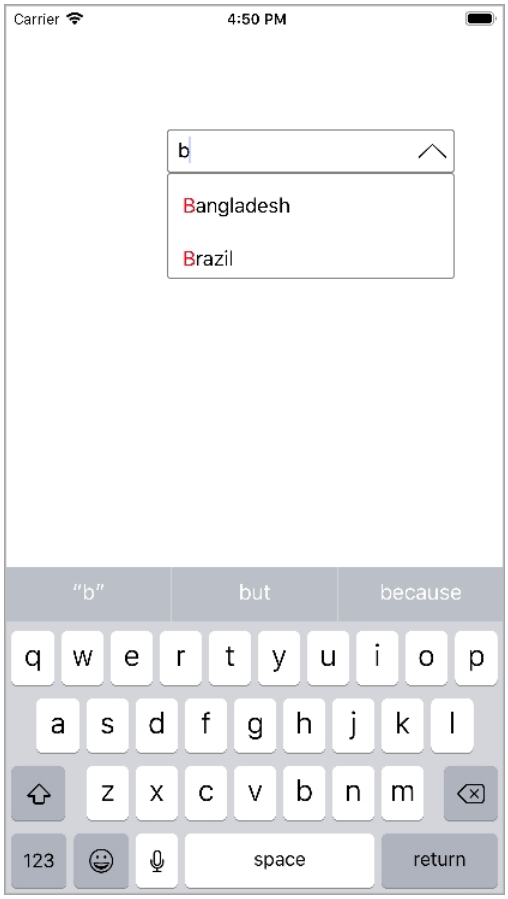
The following code example shows how you can set time delay in C1AutoComplete.

C#

```
HighlightDropdown.Delay = TimeSpan.FromSeconds(1.5);
```

Highlight Matches

The C1AutoComplete control enables quick identification of user input in the search result by highlighting the matching text. For this, the C1AutoComplete class provides the HighlightedColor property that highlights the matching characters in a specified color. You can set this property to a specific color so that the user input string gets highlighted in the search results as shown in the following image.



The following code example shows how you can highlight the matching text in the search result.

C#

```
HighlightDropdown.HighlightedColor = UIColor.Red;
```

AutoComplete Mode

AutoComplete supports multiple filtering criteria's that can be used to filter the items list based on the user input. These filtering criteria's are defined in the AutoCompleteMode enumeration and can be set using the **AutoCompleteMode** property. The AutoCompleteMode property works as a series of flags, therefore, you can also set multiple filtering criteria. The property AutoCompleteMode accepts values from **AutoCompleteMode** enumeration which specifies the mode for automatic completion in the control through following values:

1. StartsWith
2. Contains
3. EndsWith
4. MatchCase
5. MatchWholeWord

The following image shows how the AutoComplete control appears after setting the **AutoCompleteMode** property.

The following code example shows setting the auto complete mode feature in the AutoComplete control.

In Code

C#

```
autoComplete.AutoCompleteMode = AutoCompleteMode.Contains |
AutoCompleteMode.StartsWith;
```

CheckBox

The C1CheckBox control provides an iOS implementation of the classic checkbox control. The C1CheckBox is a visualization control and provides input for Boolean values. Users can select and clear the control by simply tapping it.

The following image shows the C1CheckBox control.

You can set the value of the checkbox by setting the IsChecked property. You can also customize the style by setting the Color property. The image given below shows a modified C1CheckBox in a FlexGrid.

	Active	Name	Order Total
	▶ India (15 items)		$70,573.71
	▶ Japan (11 items)		$50,893.43
	▶ Mexico (11 items)		$52,420.97
	◢ China (4 items)		$22,647.73
	☐	Mark Heath	$6,333.38
	✓	Herb Krause	$448.51
	☐	Ulrich Lehman	$7,100.58
	✓	Jack Ulam	$8,765.26
	◢ Indonesia (6 items)		$22,132.60
	✓	Ulrich Quaid	$265.49
	☐	Ted Richards	$1,581.26
	☐	Xavier Neiman	$4,353.91
	☐	Charlie Evers	$7,787.81
	✓	Vic Lehman	$1,488.28
	☐	Larry Bishop	$6,655.85

ComboBox

The C1ComboBox is an input control that combines the features of a standard text box and a list view. The control is used to display and select data from the list that appears in a drop-down. Users can also type the text into the editable text box that appears in the header to provide input. The control also supports automatic completion to display input suggestions as the user types in the text box.

Key Features

- **Automatic Completion** - The C1ComboBox control supports automatic completion feature that provides relevant suggestions to user while typing the text in the text area.
- **Edit Mode** - By default, the C1ComboBox control editable. However, you can make it non-editable to modify the input.

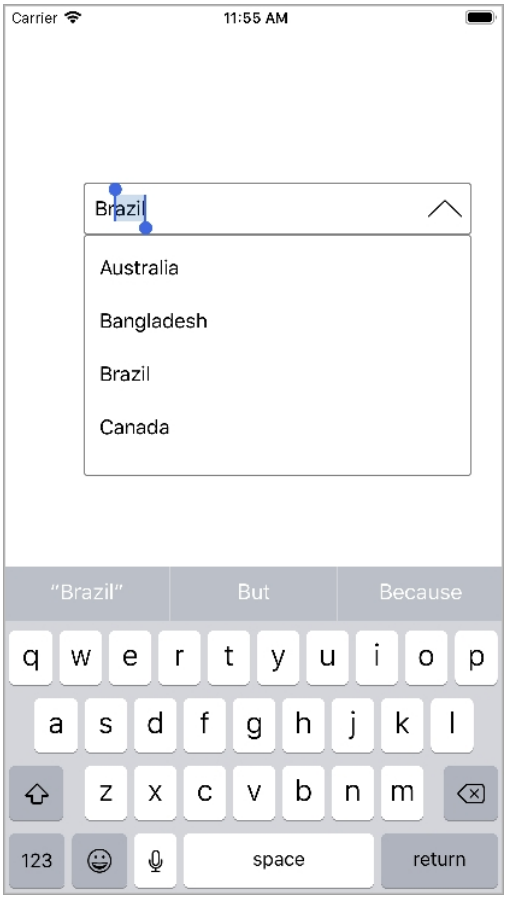
Quick Start: Display a C1ComboBox Control

This section describes adding a C1ComboBox control to your iOS application and displaying a list of items in the drop-down as input suggestions for users.

Complete the following steps to display a C1ComboBox control.

- **Step 1: Create a data source**
- **Step 2: Add a C1ComboBox control to ViewController**
- **Step 3: Run the Project**

The following image shows a C1ComboBox displaying input suggestions as the user types.



Step 1: Create a data source

Add a new class to the application that serves as the data source for C1AutoComplete.

C#

```
class Countries : NSObject
{
```

```
    [Export("Name")]
    public string Name { get; set; }

    public Countries()
    {
        this.Name = string.Empty;
    }

    public Countries(string name)
    {
        this.Name = name;
    }

    public static IEnumerable<object> GetDemoDataList()
    {
        List<object> array = new List<object>();
        var quarterNames = "Australia,Bangladesh,Brazil,Canada,China".Split(',');

        for (int i = 0; i < quarterNames.Length; i++)
        {
            array.Add(new Countries
            {
                Name = quarterNames[i]
            });
        }
        return array as IEnumerable<object>;
    }
}
```

Back to Top

Step 2: Add a C1ComboBox control to ViewController

1. Initialize a C1ComboBox control in the ViewDidLoad method.

C#

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();

    ComboBoxEdit.DisplayMemberPath = "Name";
    ComboBoxEdit.ItemsSource = Countries.GetDemoDataList();
    ComboBoxEdit.DropDownHeight = 200;
    ComboBoxEdit.Placeholder = "Please Enter...";
}
```

Step 3: Run the Project

Press **F5** to run the application.

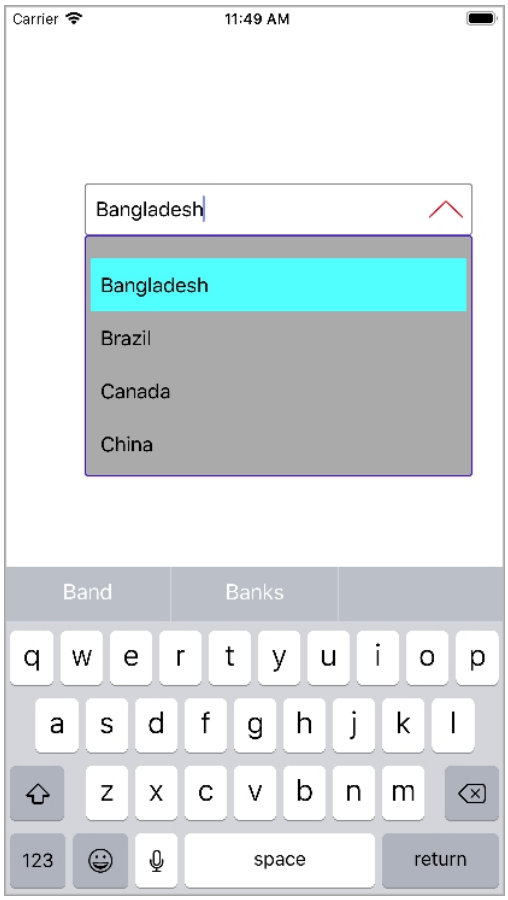
Features

Custom Appearance

The C1ComboBox control comes with various properties to customize appearance and help deliver enhanced user experience. The C1ComboBox class provides a set of properties listed below to achieve customization in the control's overall look and feel.

- SelectedBackgroundColor - Sets the selected background color.
- DropDownBackgroundColor - Sets the background color in drop-down list.
- DropDownBorderColor - Sets the color of drop-down border.
- DropDownMode - Sets the mode in which the drop down opens and accepts values from the DropDownMode.

The image given below shows a customized C1ComboBox control.



Add the given code in ViewController.m file to render a customized C1ComboBox control. This code example uses the sample created in the Quick Start.

CS

```
ComboBoxEdit.SelectedBackgroundColor = UIColor.Cyan;
ComboBoxEdit.ButtonColor = UIColor.Red;
ComboBoxEdit.DropDownBackgroundColor = UIColor.LightGray;
ComboBoxEdit.DropDownBorderColor = UIColor.Blue;
ComboBoxEdit.DropDownMode = C1.iOS.Input.DropDownMode.ForceBelow;
```

Data Binding

The C1ComboBox provides ItemsSource and DisplayMemberPath properties to bind the control to data. The

ItemsSource property lets you bind the control to a collection of items, and the DisplayMemberPath property sets the path to a value on the source object for displaying data.

The following code snippet illustrates how to set these properties in code (ViewController.m) to achieve data binding.

CS

```
ComboBoxEdit.DisplayMemberPath = "Name";
ComboBoxEdit.ItemsSource = Countries.GetDemoDataList();
```

Editing

The C1ComboBox allows users to input data by either selecting an item from the drop-down or by typing text into the text box. The control comes with editing support to help users achieve customization. The C1ComboBox class provides the IsEditable property to enable or disable editing in the ComboBox.

The IsEditable property accepts Boolean values, true or false. You can set this property to true to enable editing, while setting it to false disables editing.

CS

```
ComboBoxEdit.IsEditable = false;
```

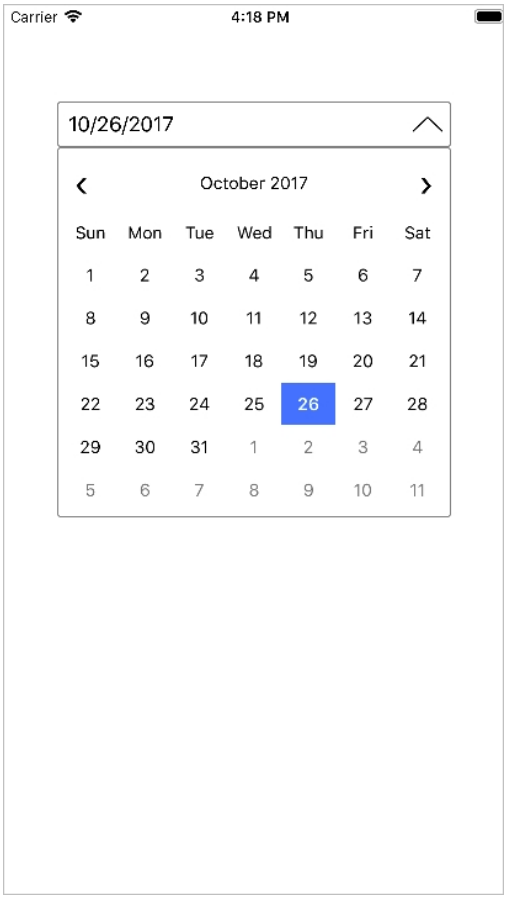
DropDown

The C1DropDown is a basic drop-down control that can be used as a base to create custom drop-down controls such as date picker, auto complete menus, etc. The control comprises three major elements including a Header view, a button, and a DropDown view. The header includes the entire width of the control, while the button is placed on the top of the header, indicating that the control can be expanded. The drop-down includes the entire length of the control and gets expanded or collapsed.

Creating a Custom Date Picker using C1DropDown

This topic provides you a walkthrough to creating a custom date picker using the C1DropDown control. For this, you begin by creating an iOS application, and initializing a C1DropDown, a C1Calendar control, and a C1MaskedTextField control. To create a date picker, you need to set the Header property to the object of the MaskedTextField and DropDown property to the object of the C1Calendar class.

The image below shows how a custom date picker created using the C1DropDown appears.



Add the following code to your ViewController to display the control.

C#

```
public static C1MaskedTextField maskedField;
public C1Calendar calendar;
public static C1DropDown d;
public C1DropDown DropDown;

public override void ViewDidLoad()
{
    base.ViewDidLoad();

    DropDown.DropDownHeight = 300;
    DropDown.DropDownWidth = DropDown.Frame.Size.Width;
    DropDown.DropDownMode = DropDownMode.ForceBelow;
    DropDown.IsAnimated = true;

    C1MaskedTextField maskedField = new C1MaskedTextField();
    maskedField.Mask = "00/00/0000";
    maskedField.BackgroundColor = UIColor.Clear;
    maskedField.BorderStyle = UITextBorderStyle.None;
    DropDown.Header = maskedField;

    C1Calendar calendar = new C1Calendar();
    calendar.SelectionChanged += (object sender, CalendarSelectionChangedEventArgs e)
=>
```

```
    {
        DateTime dateTime = calendar.SelectedDates[0];
        string strDate = dateTime.ToString("MM-dd-yyyy");
        maskedField.Text = strDate;
    };
    DropDown.DropDown = calendar;
    this.View.Add(DropDown);
}
```

MaskedTextField

The C1MaskedTextField control is designed to capture properly formatted user input. The control prevents users from entering invalid values in an input field, and other characters like slash or hyphen. The control also provides data validation by skipping over invalid entries as the user types. The control uses special elements called mask symbols or mask inputs to specify the format in which the data should be entered in an input field, .

For example, you can use the MaskedTextField control to create an input field that accepts phone numbers with area code only, or Date field that allows users to enter date in dd/mm/yyyy format only.

Mask Symbols

The C1MaskedTextField control provides an editable mask that supports a set of special mask characters/symbols. These characters are used to specify the format in which the data should be entered in an text field. For this, all you need to do is use the Mask property to specify the data format.

For example, setting the Mask property for a C1MaskedTextField control to "90/90/0000" lets users enter date in international date format. Here, the "/" character works as a logical date separator.

The following table enlists mask symbols supported by the C1MaskedTextField control.

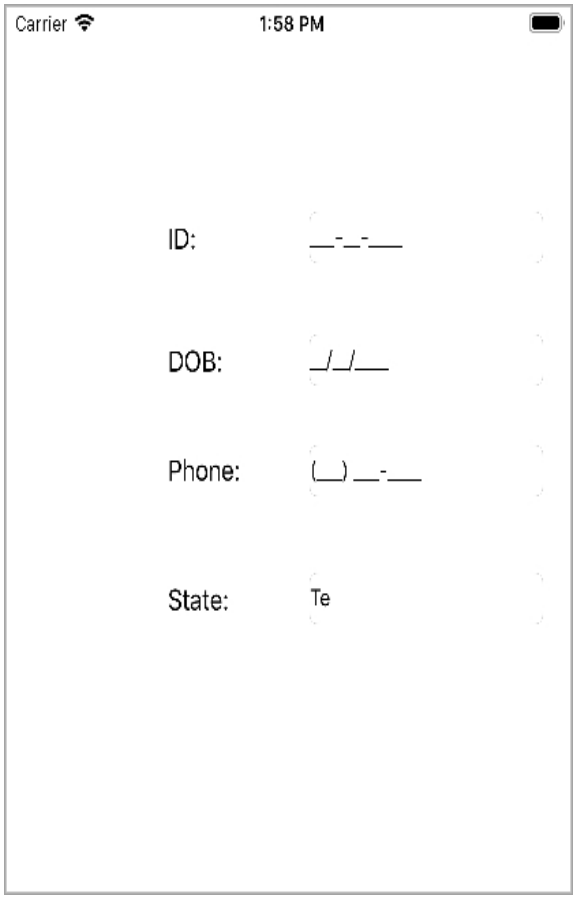
Mask Symbol	Description
0	Digit
9	Digit or space
#	Digit, sign, or space
L	Letter
?	Letter, optional
C	Character, optional
&	Character, required
I	Letter or space
A	Alphanumeric
a	Alphanumeric or space
.	Localized decimal point
,	Localized thousand separator
:	Localized time separator
/	Localized date 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
All others	Literals

Quick Start: Display C1MaskedTextField Controls

This section describes adding C1MaskedTextField controls to an iOS application for specifying four input fields, namely ID, Date of Birth, Phone, and State. The ID input field accepts a nine-digit number separated by hyphens, the Date of Birth field accepts a date in mm/dd/yyyy format, the Phone field accepts a 10-digit number with area code, and the State field accepts abbreviated postal code of a state.

The following image shows the input fields configured after completing the above steps.



Add the following code in ViewDidLoad method to display C1MaskedTextField controls.

C#

```
public override void ViewDidLoad()
   {
        base.ViewDidLoad();

        MaskedID.Mask = "000-00-0000";
```

```
        MaskedDOB.Mask = "90/90/0000";
        MaskedPhone.Mask = "(999) 000-0000";
        MaskedState.Mask = "LL";

        this.View.Add(MaskedID);
        this.View.Add(MaskedDOB);
        this.View.Add(MaskedPhone);
        this.View.Add(MaskedState);
    }
```

Toggle Button

C1ToggleButton provides a cross-platform implementation of the ToggleButton control. The control represents a two state button that a user can select (check) or clear (uncheck). It can be used to visualize Boolean values much like a CheckBox control, and allows users to make changes in the state of the control by tapping it. Moreover, the **C1ToggleButton** control offers more style options than a Checkbox control. It provides you the ability to customize the color, text, or even a custom image or view for each state. You can change the state of the control by setting the IsChecked property. Text can be controlled with the CheckedText and UncheckedText properties, images can be controlled with the CheckedImageSource and UncheckedImageSource properties, and views can be controlled with CheckedContent and UncheckedContent properties.

Quick Start: Change State and Customize the Control

This section describes adding the C1ToggleButton control to your portable or shared application and changing color of the control on the basis of change in state of the control.

Complete the following steps to change color of the control on changing its state.

- **Step 1: Initialize C1ToggleButton in code**
- **Step 2: Run the Project**

Step 1: Initialize C1ToggleButton in code

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. From the Toolbox under the **Custom Components** tab, drag the C1ToggleButton control onto the ViewController.
3. In the **Properties** window, set the **Name** of the control as **tb**.
4. Open the ViewController file from the Solution Explorer and replace its content with the code below to initialize C1ToggleButton control. This overrides the ViewDidLoad method of the View controller.

C#

```
public override void ViewDidLoad()
{
   base.ViewDidLoad();

   C1ToggleButton tb = new C1ToggleButton();
   if (tb.IsChecked == true)
   {
        tb.BackgroundColor = UIColor.Green;
   }
   else if (tb.IsChecked == false)
   {
```

```
            tb.BackgroundColor = UIColor.Red;
        }
    }
```

Back to Top

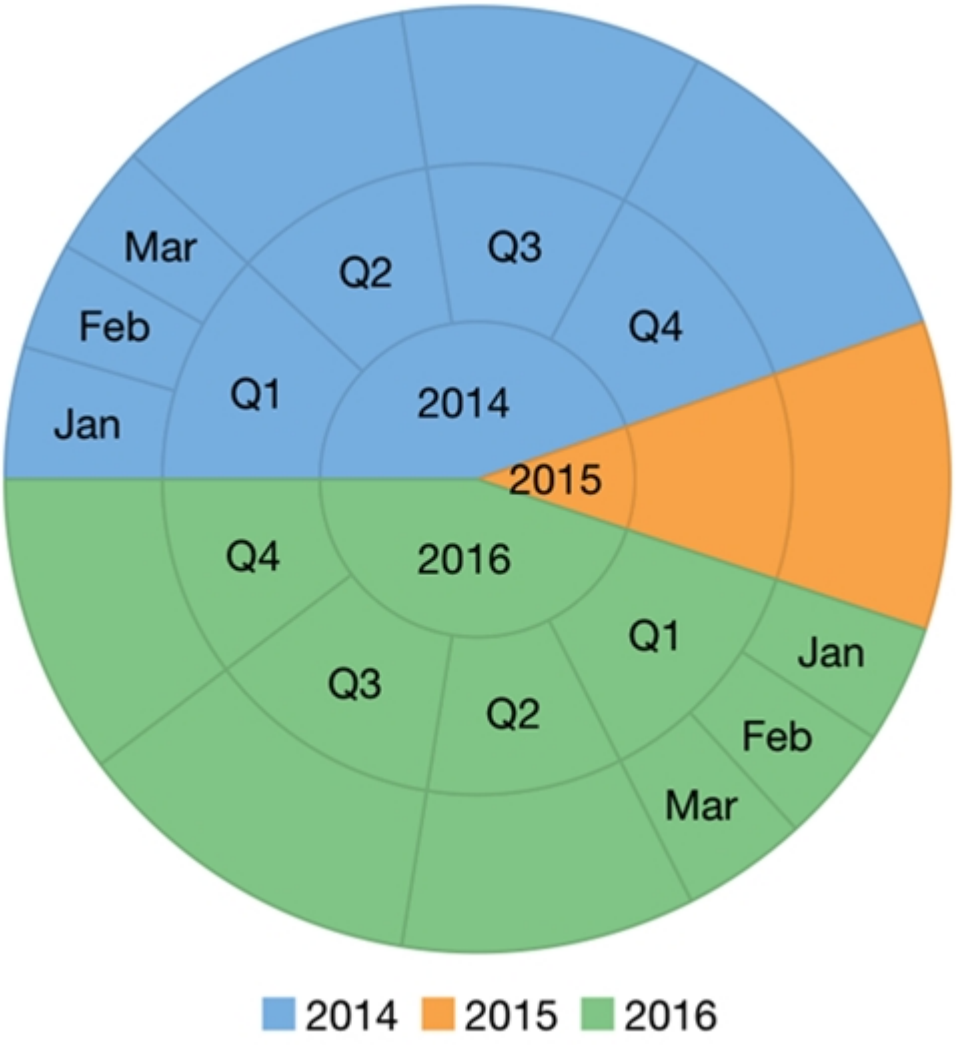
Step 2: Run the Project

Press **F5** to run your application

Back to Top

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.
- **Header and Footer**: Specifies the title header and footer text.

For more information on Sunburst chart, see QuickStart.

QuickStart

This section describes how to add a Sunburst control to your iOS app and add data to it. This topic consists of three steps:

- **Step 1: Create a data source for Sunburst**
- **Step 2: Initialize Sunburst control in Code**
- **Step 3: Run the Application**

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

Step 1: Create a data source for Sunburst

Add a new class (For example: DataService.cs) to serve as the data source for Sunburst.

C#

```
public class DataService
{
    Random rnd = new Random();
```

```
        static DataService _default;

        public static DataService Instance
        {
            get
            {
                if (_default == null)
                {
                    _default = new DataService();
                }

                return _default;
            }
        }

        public static List<SunburstDataItem> CreateHierarchicalData()
        {
            Random rnd = Instance.rnd;

            List<string> years = new List<string>();
            List<List<string>> times = new List<List<string>>()
            {
                new List<string>() { "Jan", "Feb", "Mar"},
                new List<string>() { "Apr", "May", "June"},
                new List<string>() { "Jul", "Aug", "Sep"},
                new List<string>() { "Oct", "Nov", "Dec" }
            };

            List<SunburstDataItem> items = new List<SunburstDataItem>();
            var yearLen = Math.Max((int)Math.Round(Math.Abs(5 -
Instance.rnd.NextDouble() * 10)), 3);
            int currentYear = DateTime.Now.Year;
            for (int i = yearLen; i > 0; i--)
            {
                years.Add((currentYear - i).ToString());
            }
            var quarterAdded = false;

            foreach (string y in years)
            {
                var i = years.IndexOf(y);
                var addQuarter = Instance.rnd.NextDouble() > 0.5;
                if (!quarterAdded && i == years.Count - 1)
                {
                    addQuarter = true;
                }
                var year = new SunburstDataItem() { Year = y };
                if (addQuarter)
                {
                    quarterAdded = true;
```

```
                    foreach (List<string> q in times)
                    {
                        var addMonth = Instance.rnd.NextDouble() > 0.5;
                        int idx = times.IndexOf(q);
                        var quar = "Q" + (idx + 1);
                        var quarters = new SunburstDataItem() { Year = y, Quarter =
quar };
                        if (addMonth)
                        {
                            foreach (string m in q)
                            {
                                quarters.Items.Add(new SunburstDataItem()
                                {
                                    Year = y,
                                    Quarter = quar,
                                    Month = m,
                                    Value = rnd.Next(20, 30)
                                });
                            };
                        }
                        else
                        {
                            quarters.Value = rnd.Next(80, 100);
                        }
                        year.Items.Add(quarters);
                    };
                }
                else
                {
                    year.Value = rnd.Next(80, 100);
                }
                items.Add(year);
            };

            return items;
        }

        public static List<FlatDataItem> CreateFlatData()
        {
            Random rnd = Instance.rnd;
            List<string> years = new List<string>();
            List<List<string>> times = new List<List<string>>()
            {
                new List<string>() { "Jan", "Feb", "Mar"},
                new List<string>() { "Apr", "May", "June"},
                new List<string>() { "Jul", "Aug", "Sep"},
                new List<string>() { "Oct", "Nov", "Dec" }
            };

            List<FlatDataItem> items = new List<FlatDataItem>();
            var yearLen = Math.Max((int)Math.Round(Math.Abs(5 - rnd.NextDouble() *
```

```
10)), 3);
            int currentYear = DateTime.Now.Year;
            for (int i = yearLen; i > 0; i--)
            {
                years.Add((currentYear - i).ToString());
            }
            var quarterAdded = false;
            foreach (string y in years)
            {

                var i = years.IndexOf(y);
                var addQuarter = rnd.NextDouble() > 0.5;
                if (!quarterAdded && i == years.Count - 1)
                {
                    addQuarter = true;
                }
                if (addQuarter)
                {
                    quarterAdded = true;
                    foreach (List<string> q in times)
                    {
                        var addMonth = rnd.NextDouble() > 0.5;
                        int idx = times.IndexOf(q);
                        var quar = "Q" + (idx + 1);
                        if (addMonth)
                        {
                            foreach (string m in q)
                            {
                                items.Add(new FlatDataItem()
                                {
                                    Year = y,
                                    Quarter = quar,
                                    Month = m,
                                    Value = rnd.Next(30, 40)
                                });
                            };
                        }
                        else
                        {
                            items.Add(new FlatDataItem()
                            {
                                Year = y,
                                Quarter = quar,
                                Value = rnd.Next(80, 100)
                            });
                        }
                    };
                }
                else
                {
                    items.Add(new FlatDataItem()
```

```
                {
                    Year = y.ToString(),
                    Value = rnd.Next(80, 100)
                });
            }
        };

        return items;
    }

public class FlatDataItem
{
    public string Year { get; set; }
    public string Quarter { get; set; }
    public string Month { get; set; }
    public double Value { get; set; }
}
public class SunburstDataItem
{
    List<SunburstDataItem> _items;

    public string Year { get; set; }
    public string Quarter { get; set; }
    public string Month { get; set; }
    public double Value { get; set; }
    public List<SunburstDataItem> Items
    {
        get
        {
            if (_items == null)
            {
                _items = new List<SunburstDataItem>();
            }

            return _items;
        }
    }
}
public class Item
{
    public int Year { get; set; }
    public string Quarter { get; set; }
    public string MonthName { get; set; }
    public int MonthValue { get; set; }
    public double Value { get; set; }
}
```

Back to Top

Step 2: Initialize Sunburst control in Code

To initialize the Sunburst control, open the ViewController file from the Solution Explorer and replace its content with

the code below. This overrides the **ViewDidLoad** method of the View controller in order to initialize Sunburst.

C#

```
public override void ViewDidLoad()
        {
            base.ViewDidLoad();
           // Perform any additional setup after loading the view, typically from a
nib.
            C1Sunburst sunburst = new C1Sunburst();
            sunburst.Binding = "Value";
            sunburst.BindingName = "Year,Quarter,Month";
            sunburst.ToolTipContent = "{}{name}\n{y}";
            sunburst.DataLabel.Position = PieLabelPosition.Center;
            sunburst.DataLabel.Content = "{}{name}";
            sunburst.ItemsSource = DataService.CreateFlatData();
            this.Add(sunburst);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            CGRect rect = new CGRect(this.View.Frame.X, this.View.Frame.Y + 80,
                                     this.View.Frame.Width, this.View.Frame.Height -
80);
            sunburst.Frame = new CGRect(rect.X, rect.Y, rect.Width, rect.Height  -
10);
        }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

Legend

You can specify the position where you want to display the legend using the **LegendPosition** property of the Sunburst chart. Legend helps in displaying the series of a chart with a predefined symbol and name of the series.

The position of legend is by default set to "Auto", which means the legend positions itself automatically depending on the real estate available on the device. This allows the Sunburst to efficiently occupy the available space on the device. Users have the option to customize the appearance of the legend and enhance the visual appeal of the Sunburst Chart control.

The image below shows customized legend in the Sunburst Chart control.

The following code example demonstrates how to set these properties. This example uses the sample created in the Quick Start section.

CS

```
sunburst.LegendPosition = ChartPositionType.Top;sunburst.LegendOrientation = Orientation.Horizontal;sunburst.LegendStyle.Stroke = UIColor.Blue;sunburst.Header = "Product By Value";sunburst.Footer = "2014 GrapeCity.inc.";
```

Selection

The Sunburst Chart control allows you to select data points by clicking or touching a sunburst slice. Use the **SelectionMode** property to specify whether you want to allow selection by data point or no selection at all(default). 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.

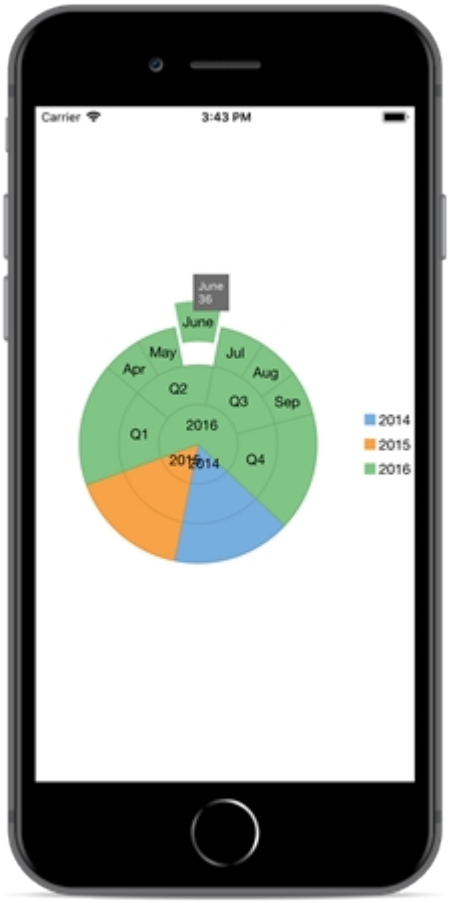
When the SelectionMode is set to **Point**, you can change the position of the selected sunburst slice by setting the **SelectedItemPosition** property. Also, you can set the **SelectedItemOffset** property to move the selected sunburst slice away from the center. Setting the SelctionMode property to Point causes the Sunburst to update the selection property when the user clicks or touch on a sunburst slice.

The Sunburst offers two additional properties to customize the selection:

- **SelectedItemOffset**: Specifies the offset of the selected sunburst slice from the center of the control.

- **SelectedItemPosition**: Specifies the position of the selected sunburst slice. The available options are Top, Bottom, Left, Right, and None (default).

The image below show how the Sunburst chart appears after you set the **SelectionMode** property.



The following code example demonstrates how to set these properties using C#. This examples uses the data created in the Quick Start section.

CS

```
sunburst.SelectionMode = ChartSelectionModeType.Point;
sunburst.SelectedItemPosition = ChartPositionType.Top;
sunburst.SelectedItemOffset = 0.2;
```

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.GroupDescription** property. GroupDescription objects are flexible, allowing you to group data based on value or on grouping functions. The below code uses the **GroupChanged** event and **SortChanged** event for performing grouping operation on Sunburst Chart.

The image below shows how to set Grouping in Sunburst Chart control.

The following code example demonstrates how to set these properties using C#. This examples uses the data created in the Quick Start section.

CS

```
public partial class ViewController : UIViewController
    {
        private C1CollectionView<Item> cv;
        public ViewController(IntPtr handle) : base(handle)
        {
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            // Perform any additional setup after loading the view, typically from a
nib.
            sunburst = new C1Sunburst();
            sunburst.Binding = "Value";
            sunburst.BindingName = "Year,Quarter,Month";
            sunburst.ToolTipContent = "{}{name}\n{y}";
            sunburst.DataLabel.Position = PieLabelPosition.Center;
            sunburst.DataLabel.Content = "{}{name}";
            cv = DataService.CreateGroupCVData();
            this.Add(sunburst);

            cv.GroupChanged += View_GroupChanged;
            cv.SortChanged += Cv_SortChanged;
```

```
            //Sort cannot work synchronize with group in current CollectionView
            SortDescription yearSortDescription = new SortDescription("Year",
SortDirection.Ascending);
            SortDescription quarterSortDescription = new SortDescription("Quarter",
SortDirection.Ascending);
            SortDescription monthSortDescription = new SortDescription("MonthValue",
SortDirection.Ascending);
            SortDescription[] sortDescriptions = new SortDescription[] {
yearSortDescription, quarterSortDescription, monthSortDescription };
            cv.SortAsync(sortDescriptions);
        }
        private void Cv_SortChanged(object sender, System.EventArgs e)
        {
            GroupDescription yearGroupDescription = new GroupDescription("Year");
            GroupDescription quarterGroupDescription = new
GroupDescription("Quarter");
            GroupDescription monthGroupDescription = new
GroupDescription("MonthName");
            GroupDescription[] groupDescriptions = new GroupDescription[] {
yearGroupDescription, quarterGroupDescription, monthGroupDescription };
            cv.GroupAsync(groupDescriptions);
        }

        private void View_GroupChanged(object sender, System.EventArgs e)
        {
            this.sunburst.ItemsSource = cv;
            CGRect rect = new CGRect(this.View.Frame.X, this.View.Frame.Y + 80,
                                     this.View.Frame.Width, this.View.Frame.Height -
80);

            sunburst.Frame = new CGRect(rect.X, rect.Y, rect.Width, rect.Height -
10);
        }
        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();

            sunburst.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y,
                                     this.View.Frame.Width, this.View.Frame.Height);
        }
```

Zooming and Panning

Zooming can be performed in Sunburst chart using ZoomBehavior class. To implement zooming, you need to create an object of ZoomBehavior class available in the C1.iOS.Chart.Interaction namespace and pass it as a parameter to the Add method. This method adds zoom behavior to the behavior collection by accessing it through Behaviors property of ChartBase class.

The following code examples demonstrate how to implement zooming in C#. These examples use the sample created in the Quick Start section.

```
C#
ZoomBehavior z = new ZoomBehavior();
sunburst.Behaviors.Add(z);
```

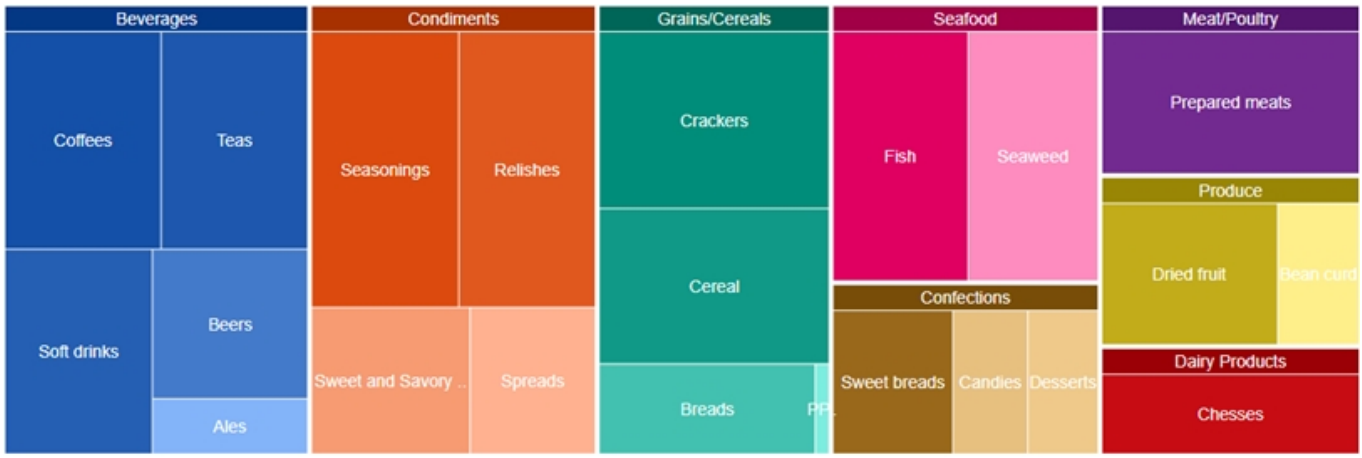
Similarly, panning can be implemented in Sunburst chart by creating an object of TranslateBehavior class available in the C1.iOS.Chart.Interaction namespace and passing it as a parameter to the Add method. This method adds translation behavior to the behavior collection by accessing it through Behaviors property of the ChartBase class.

The following code examples demonstrate how to implement panning in C#. These examples use the sample created in the Quick Start section.

```
C#
TranslateBehavior t = new TranslateBehavior();
sunburst.Behaviors.Add(t);
```

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.

Key Features

TreeMap provides many different features that enable the developers to build intuitive and professional-looking applications. The main features of TreeMap are as follows:

- **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 itss appearance as per their preference. A set of varied color palettes are available to clearly display categories in a TreeMap.

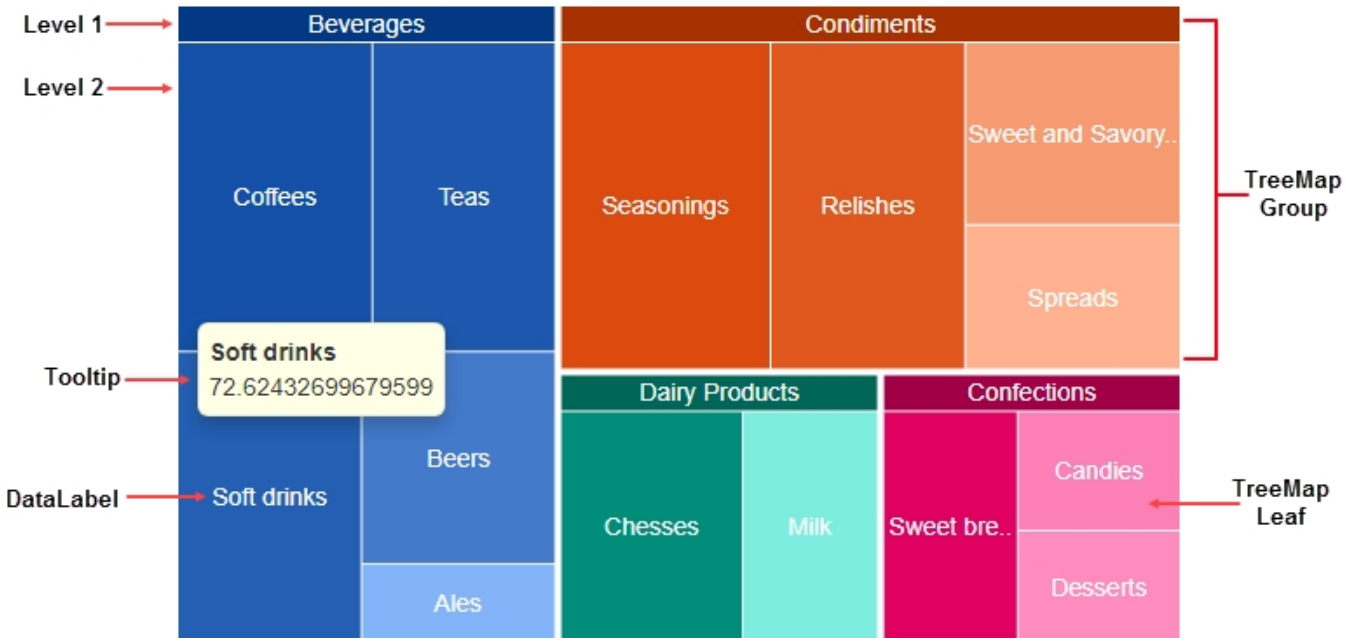
- **Custom Hierarchical Levels**
  TreeMap enables users to vary the depth of data to be visualized and further drill down (or reverse drill down) the data for analysis and comparison.

- **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.

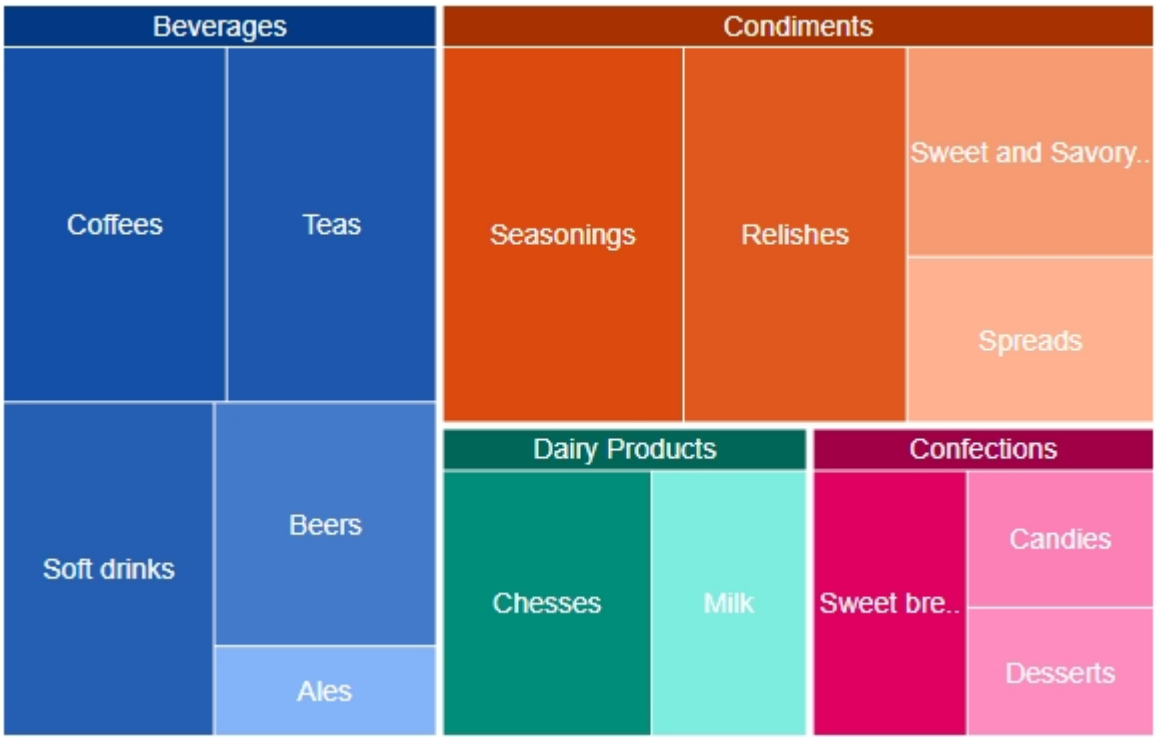


Layouts

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 fromTreeMapType 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.

QuickStart

This section describes how to add a TreeMap control to your iOS app and add data to it. This topic consists of three steps:

- **Step 1: Create a data source for TreeMap**
- **Step 2: Add a TreeMap control**
- **Step 3: Run the Application**

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

Step 1: Create a data source for TreeMap

Add a new class **ThingSale.cs** to serve as the data source for TreeMap.

ThingSale.cs

```
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;
        }
    }
```

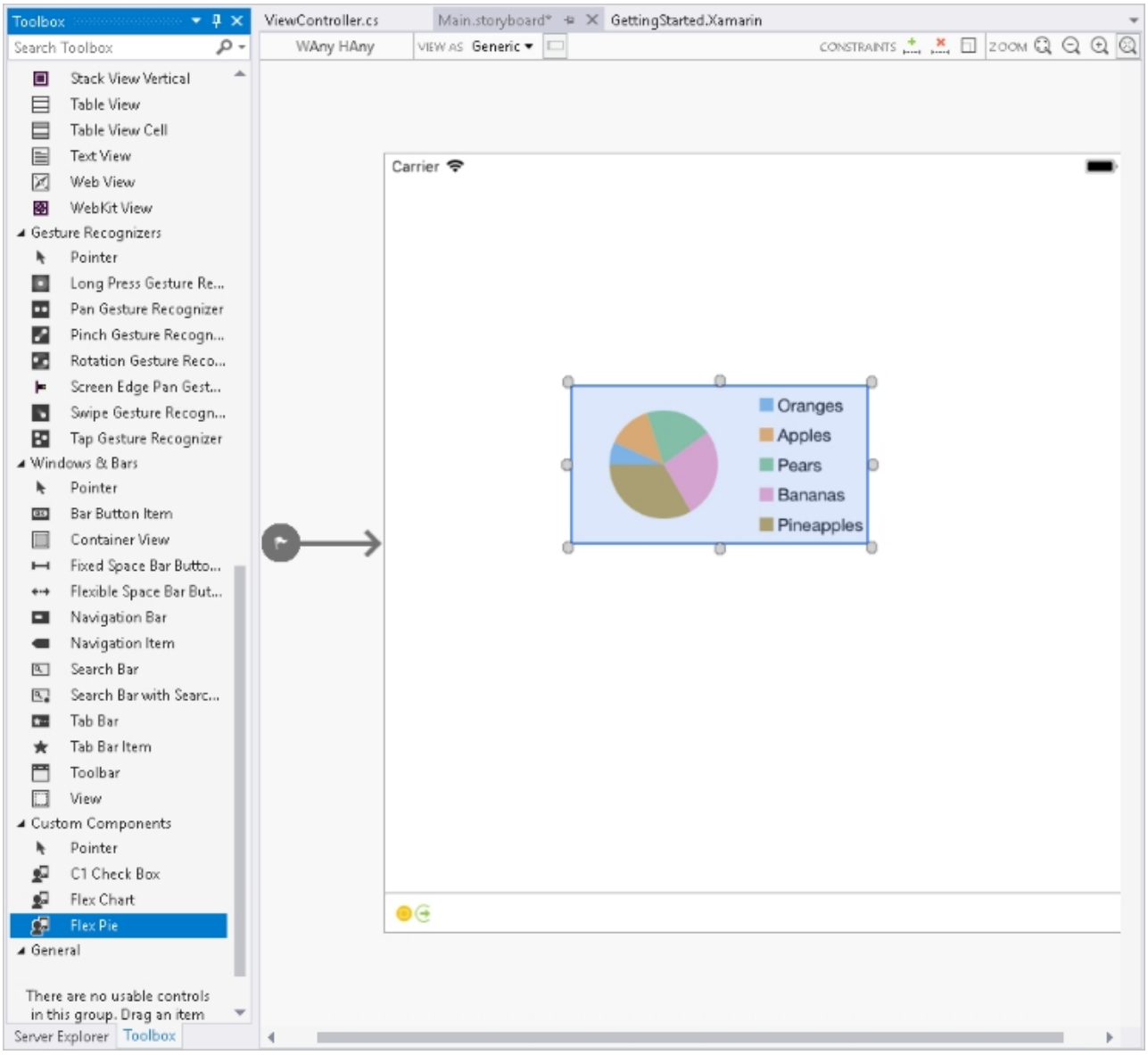
Back to Top

Step 2: Add a TreeMap control

Complete the following steps to initialize a TreeMap control in C#.

Add TreeMap control in StoryBoard

1. In the **Solution Explorer**, click **Main.storyboard** to open the storyboard editor.
2. From the **Toolbox** under **Custom Components** tab, drag a TreeMap onto the **ViewController**.



Initialize TreeMap control in Code

To initialize the TreeMap control, open the ViewController file from the Solution Explorer and replace its content with the code below. This overrides the **ViewDidLoad** method of the View controller in order to initialize TreeMap.

C#

```
public partial class ViewController : UIViewController
    {
        C1TreeMap treeMap;
        public ViewController (IntPtr handle) : base (handle)
        {
        }
        public override void ViewDidLoad ()
        {
```

```
            base.ViewDidLoad ();
            treeMap = new C1TreeMap();
            treeMap.ChartType = C1.iOS.Chart.TreeMapType.Squarified;
            treeMap.Binding = "sales";
            treeMap.BindingName = "type";
            treeMap.MaxDepth = 2;
            treeMap.ChildItemsPath = "items";
            treeMap.ItemsSource = SalesData.CreateHierarchicalData();
            this.Add(treeMap);

            treeMap.DataLabel = new ChartDataLabel() { Content = "{}{type}", Position =
ChartLabelPosition.Center };
            treeMap.DataLabel.Style.FontSize = 10;
        }

        public override void ViewDidLayoutSubviews()
        {
            base.ViewDidLayoutSubviews();
            treeMap.Frame = new CGRect(this.View.Frame.X, this.View.Frame.Y + 80,
                                       this.View.Frame.Width, this.View.Frame.Height - 80);
        }
    }
```

Back to Top

Step 3: Run the Application

Press **F5** to run the application.

Back to Top

Features

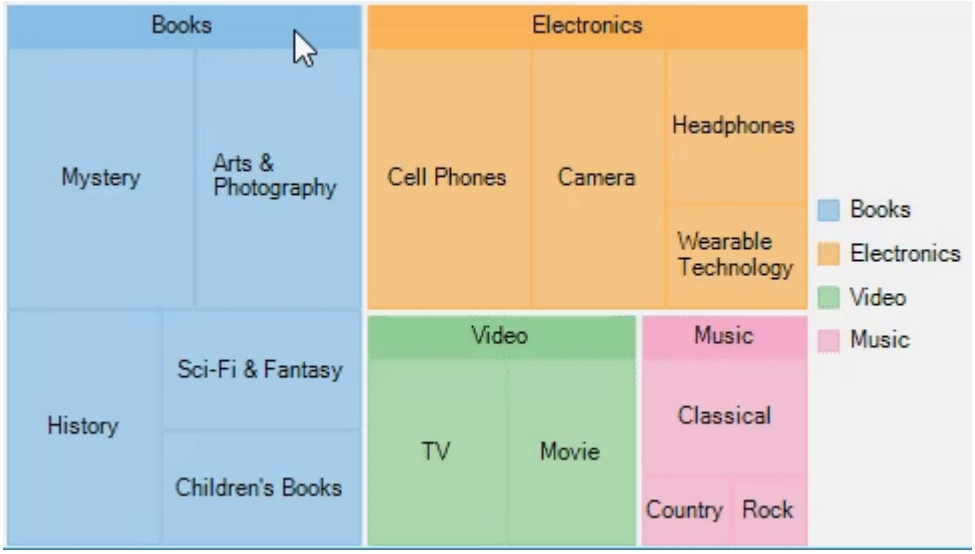
Drilldown

TreeMap allows drilling down the data items of its data further for detailed analysis. End users can access the lower levels in the data hierarchy by simply clicking the desired node. Whereas, to move back up in the hierarchy, users simply need to right-click in the plot area.

To implement the drilldown functionality in the TreeMap control, set the **MaxDepth** property to a value greater that 0. This property defines the levels of hierarchical data in the TreeMap control.

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 gif image demonstrates drilling-down by showing data points of the clicked TreeMap node.



The following code example demonstrates how to set the MaxDepth property of the TreeMap in C# to enable DrillDown. This example uses the sample created in the QuickStart section.

Theming.cs

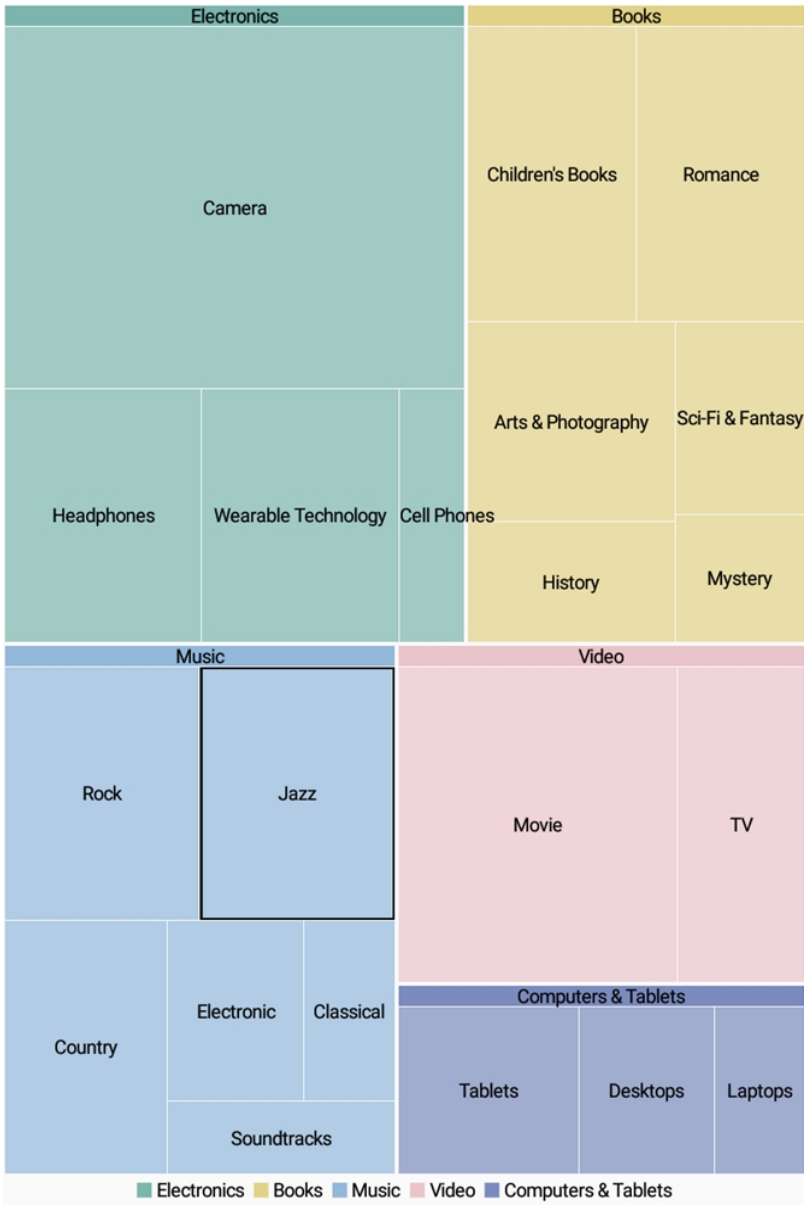
```
treeMap.MaxDepth = 2;
```

Selection

TreeMap lets you enable selection of its data items and groups. User can select a node and draw focus on it by simply clicking it. You need to set the SelectionMode property provided by the ChartBase class to either of the following values in the ChartSelectionMode enumeration:

- **None (default)**: Selection is disabled.
- **Point**: A point is selected.

The following image illustrates default selection of a data point along with its children nodes in the hierarchy.



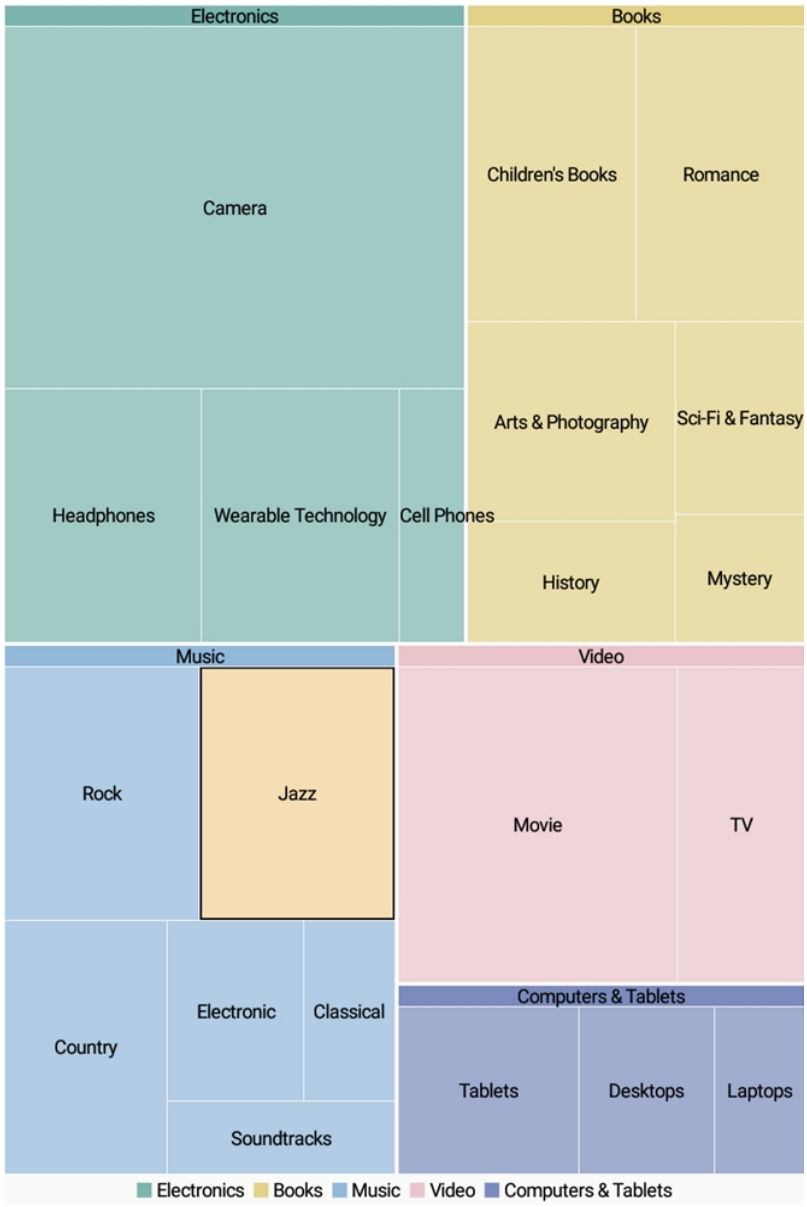
The following code snippet shows how to set the **SelectionMode** property for a tree map control.

Theming.cs

```
// Implement selection mode

treeMap.SelectionMode = ChartSelectionModeType.Point;
```

Customized TreeMap Selection

To customize the TreeMap selection, you can use SelectionStyle property and style the selected item as illustrated in the following image.



The following code snippet demonstrates using of **SelectionStyle** property to change fill color of the selected TreeMap node.

Theming.cs

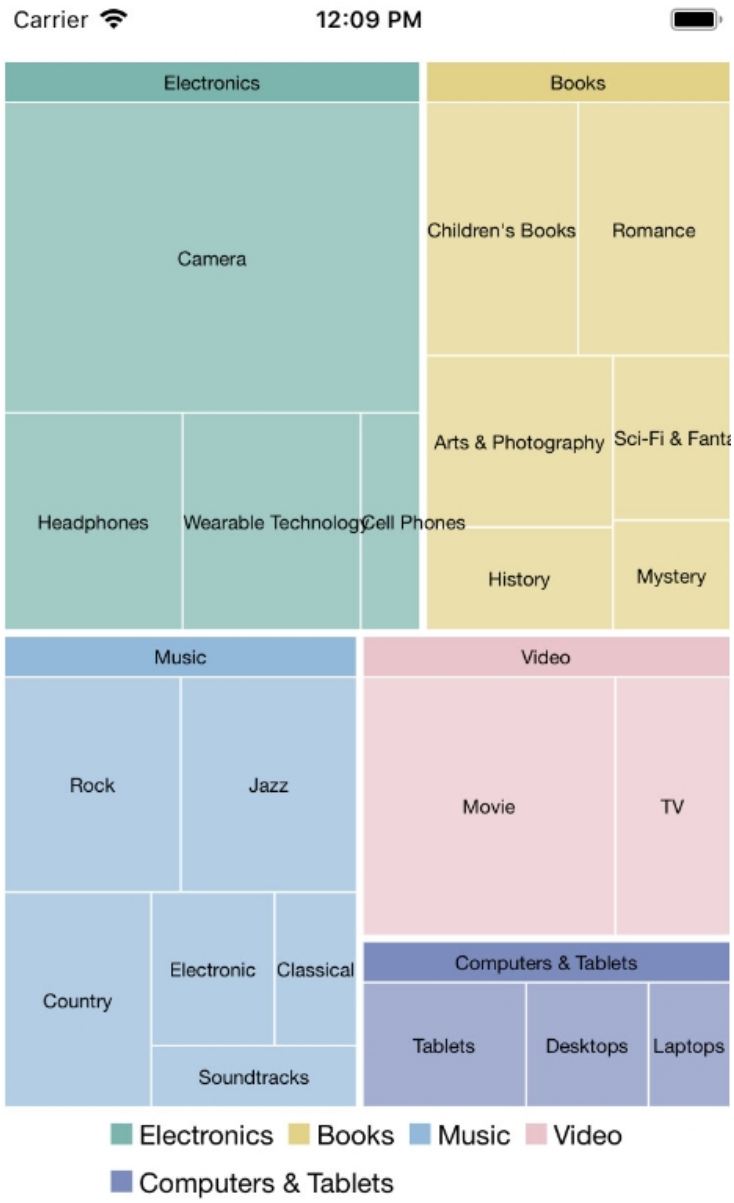
```
// Apply custom color to the selection

treeMap.SelectionStyle.Fill = Color.Wheat;
```

Additionally, you can customize the behavior of TreeMap selection by handling **SelectionChanged** event. Also, you can utilize **SelectedIndex** and **SelectedItem** properties, and reuse the obtained information in your application.

Theming

The TreeMap control allows you to customize its appearance by using the **Palette** property. This property accepts value from the **Palette** enumeration provided by the **ChartBase** class.

The following image shows how a TreeMap control appears after applying a theme using the Palette property.



The following code examples demonstrate how to set **Palette** property in C#. These examples use the sample created in the Quick Start section.

Theming.cs

```
treeMap.Palette = Palette.Zen;
```