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 Xuni, 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 2.3.3.193 and above
- Visual Studio 2015 Update 3

Windows System Requirements

- Windows 8.1 and above

Mac System Requirements

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

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:

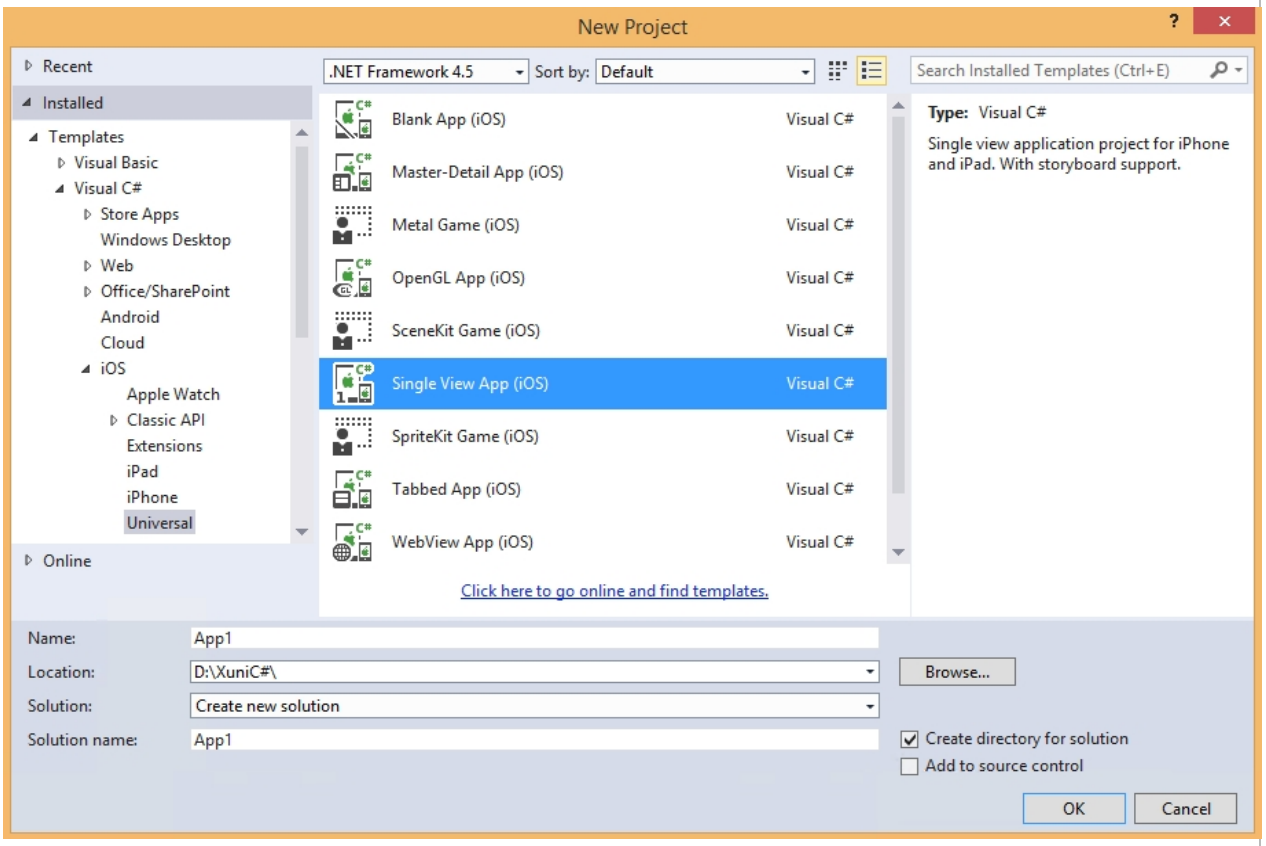
https://developer.xamarin.com/guides/ios/getting_started/

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

Visual Studio (Windows)

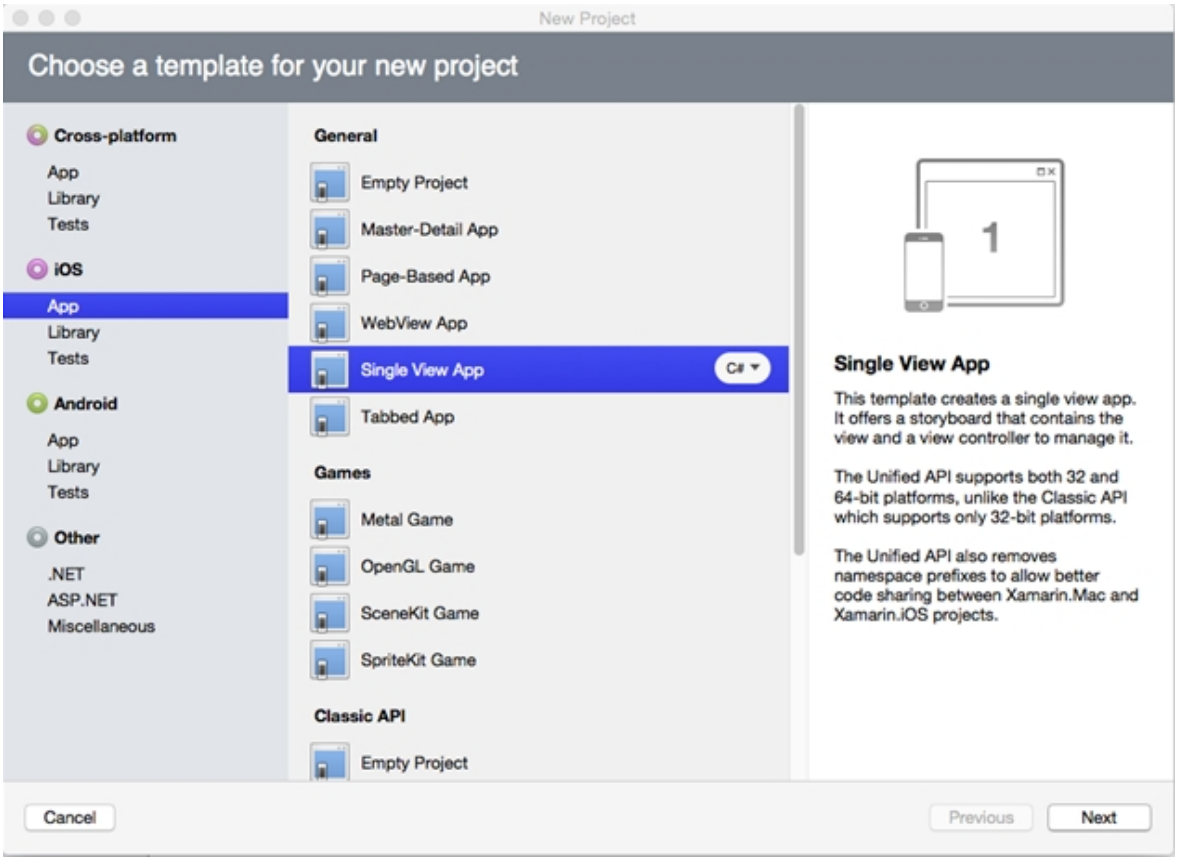
1. Select **File | New | Project**.
2. Under installed templates, select **Visual C# |Universal**.
3. In the right pane, select **Single View App (iOS)**.
4. Type a name for your app and select a location to save it.

5. Click OK.

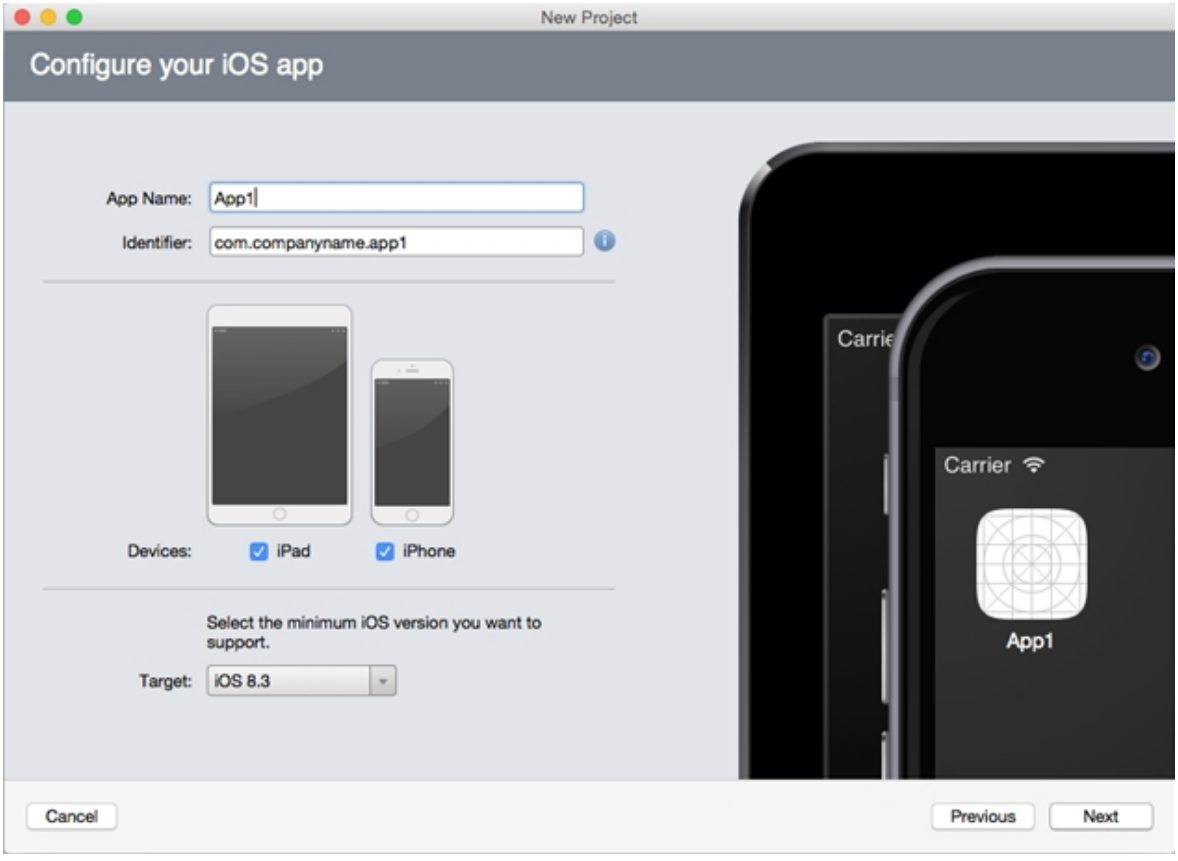


Visual Studio for Mac (macOS)

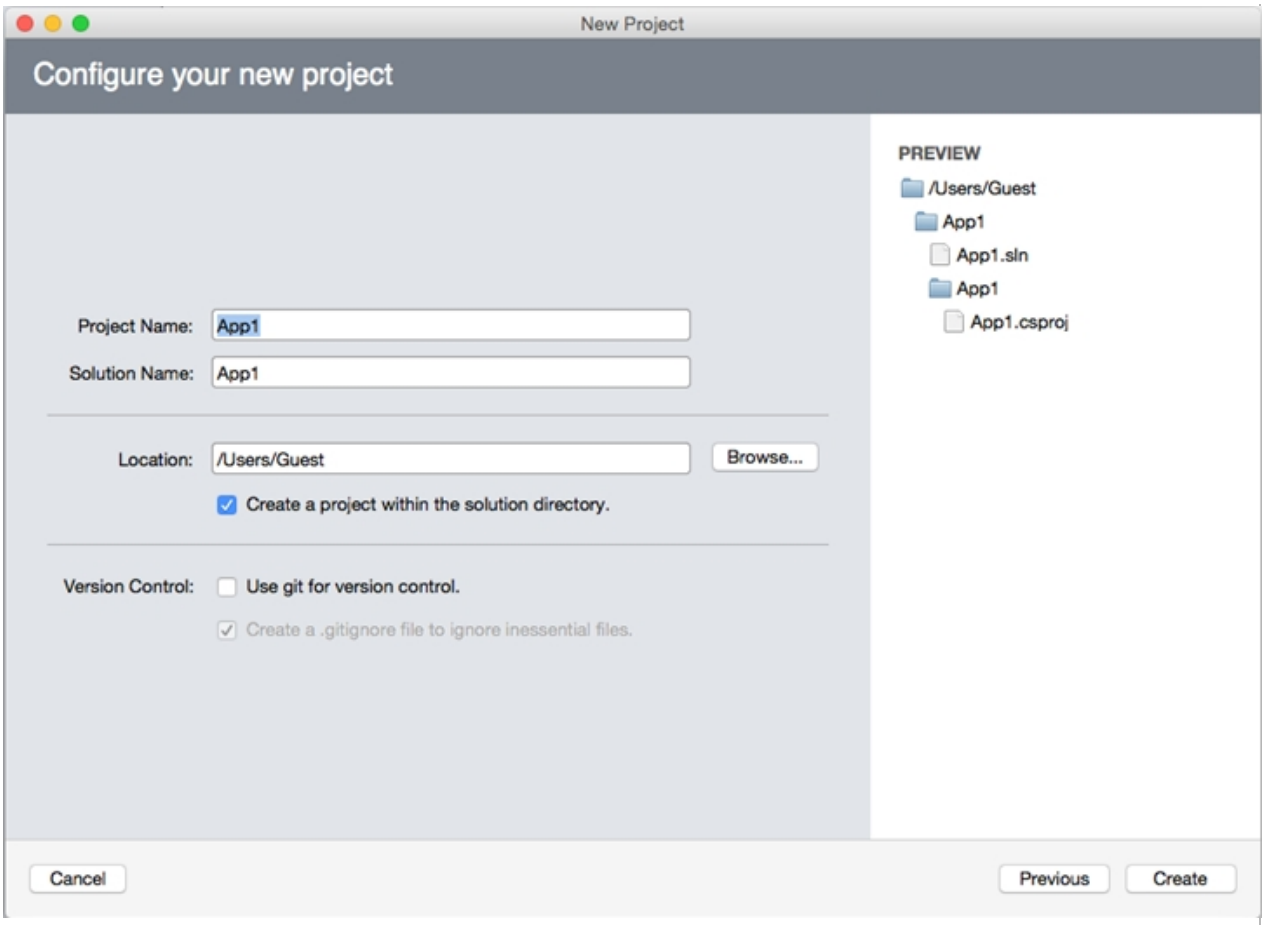
1. Select **File** | **New Solution**.
2. Select **iOS** | **App**.
3. In the right pane, select **Single View App**.
4. Click **Next**.



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 your app using GrapeCity License Manager Add-in
- Licensing you app using website

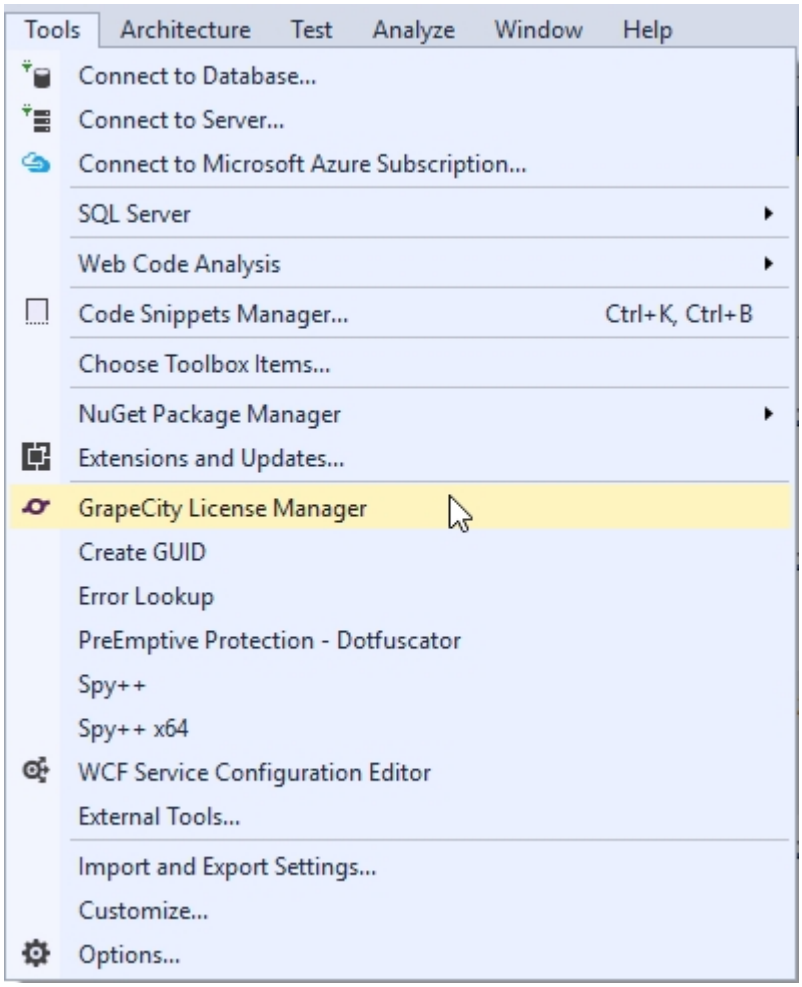
Licensing your app using GrapeCity License Manager Add-in

If you are using ComponentOne Xamarin Edition 2017v2 and newer versions with Visual Studio, you can use the GrapeCity License Manager Add-in to license your apps. If you are using a version with Xamarin Studio or Visual Studio for Mac, follow the instructions given in Licensing your app using website.

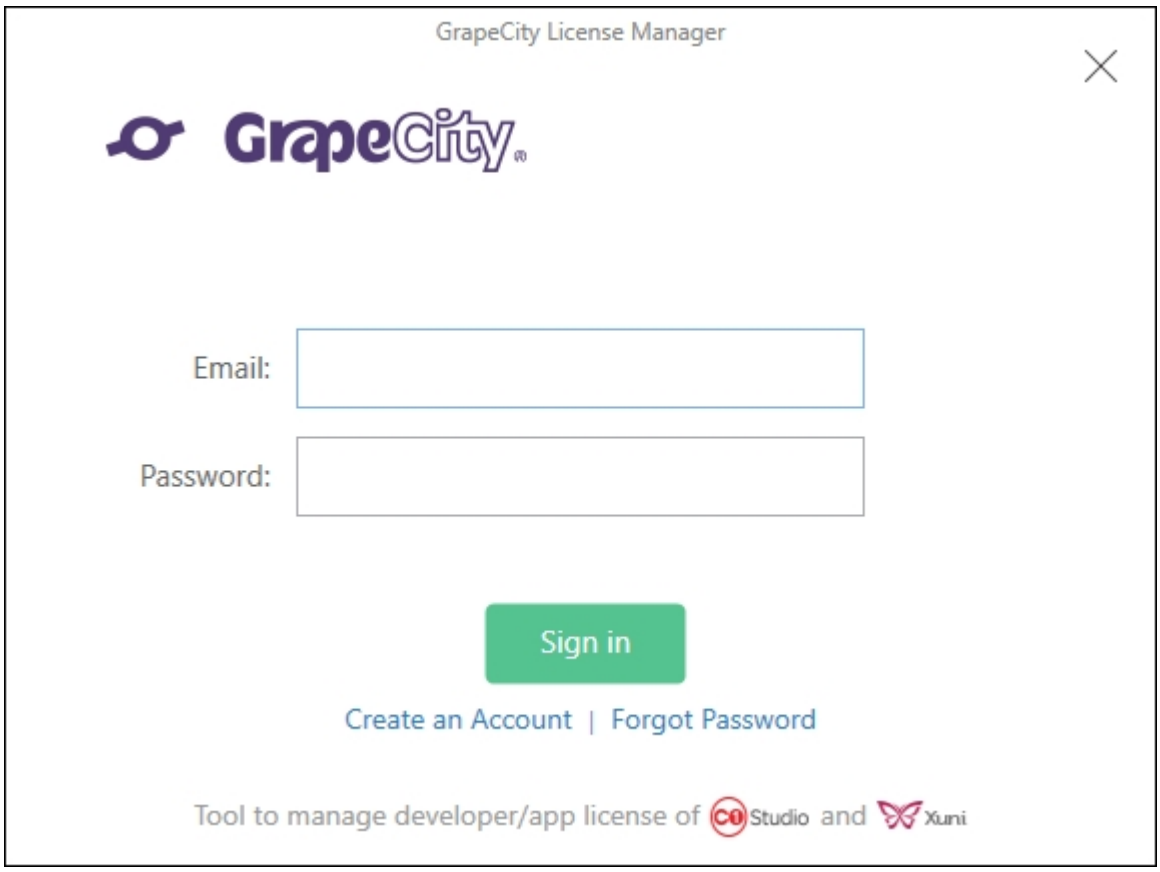
GrapeCity License Manager Add-in for Visual Studio

The GrapeCity License Manager Add-in generates XML keys to license your apps directly in Visual Studio. To license a Xamarin.iOS app, complete the steps given below:

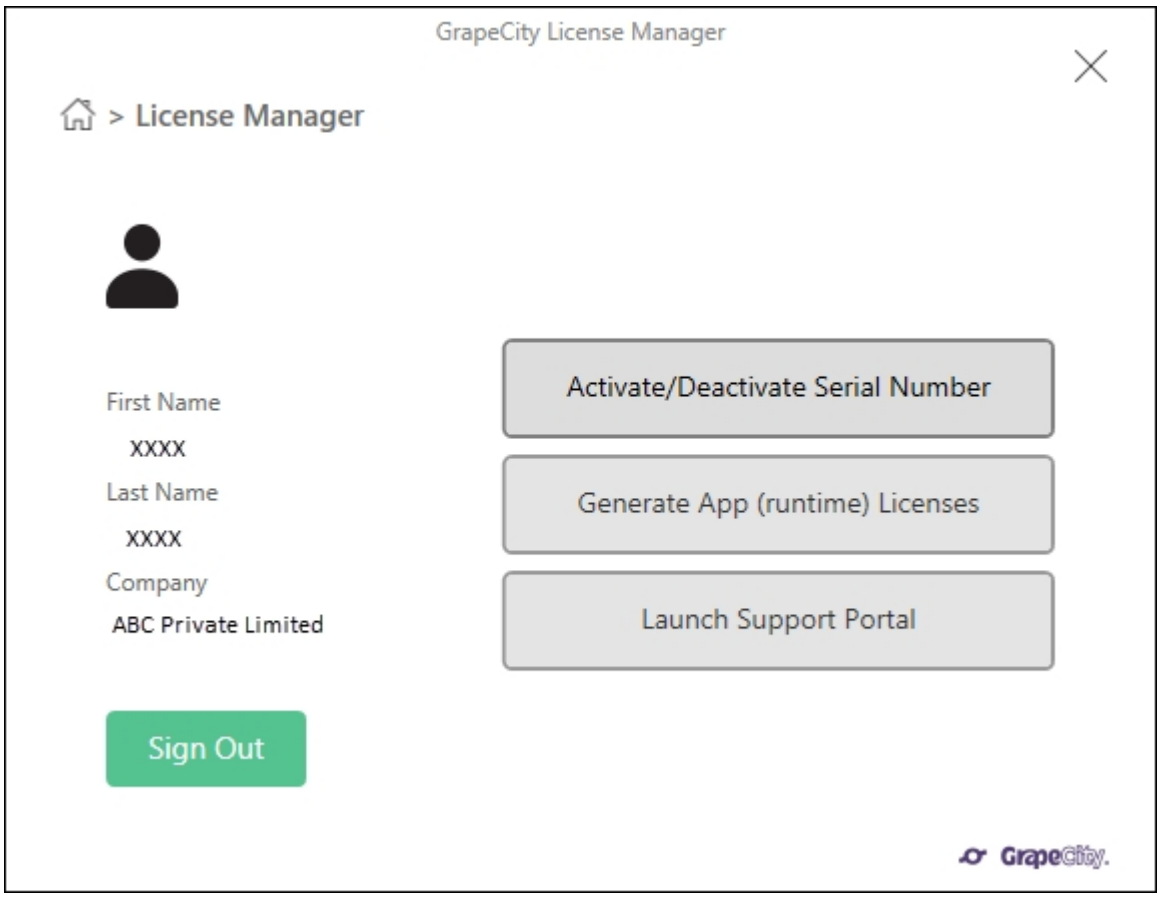
1. From the **Tools** menu in Visual Studio, select **GrapeCity License Manager** option.



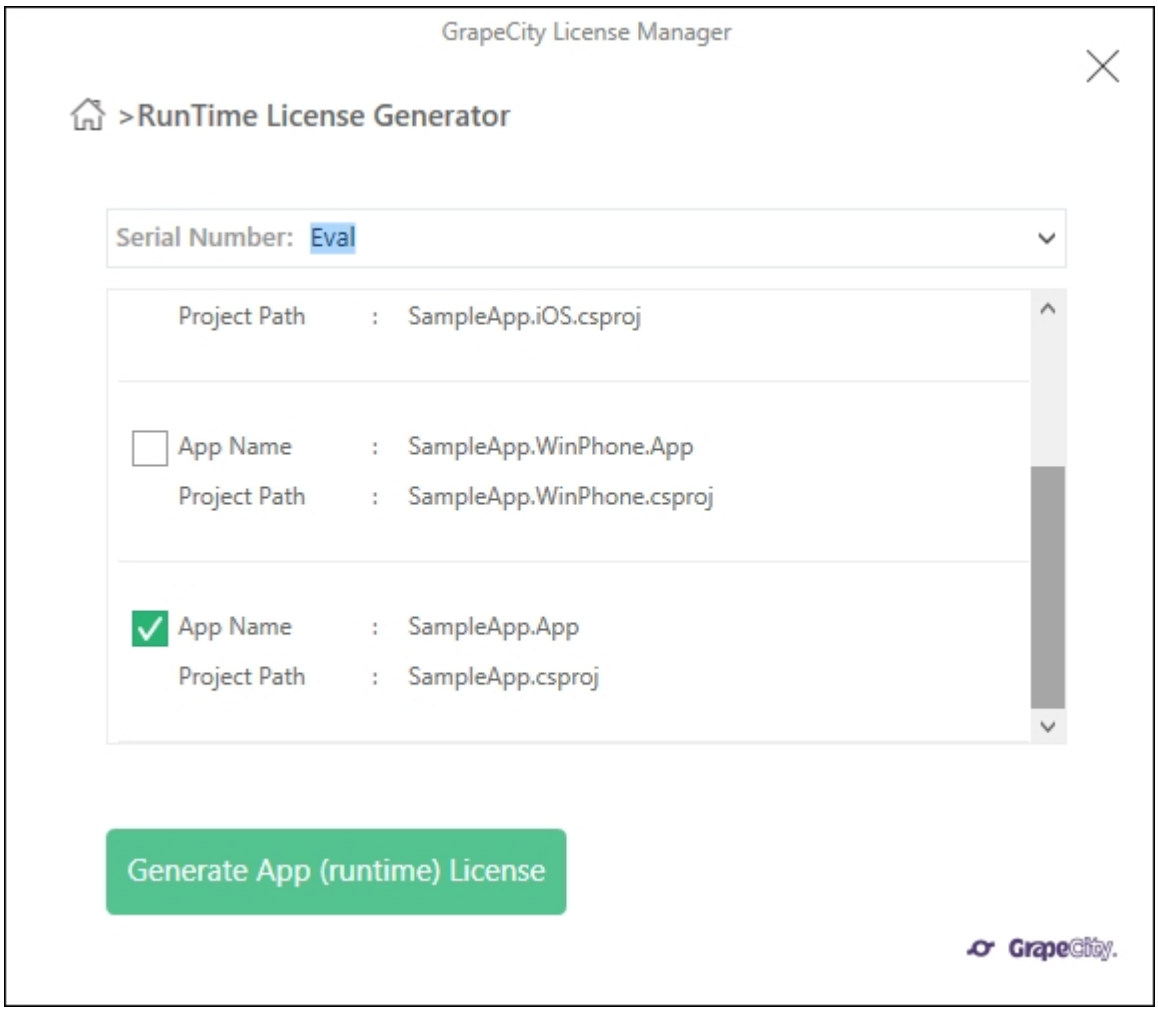
2. Sign-in into the **License Manager** using your email address and password. If you have not created a **GrapeCity Account** in the past, you can create an account at this step. If you are already signed in, skip the screen.



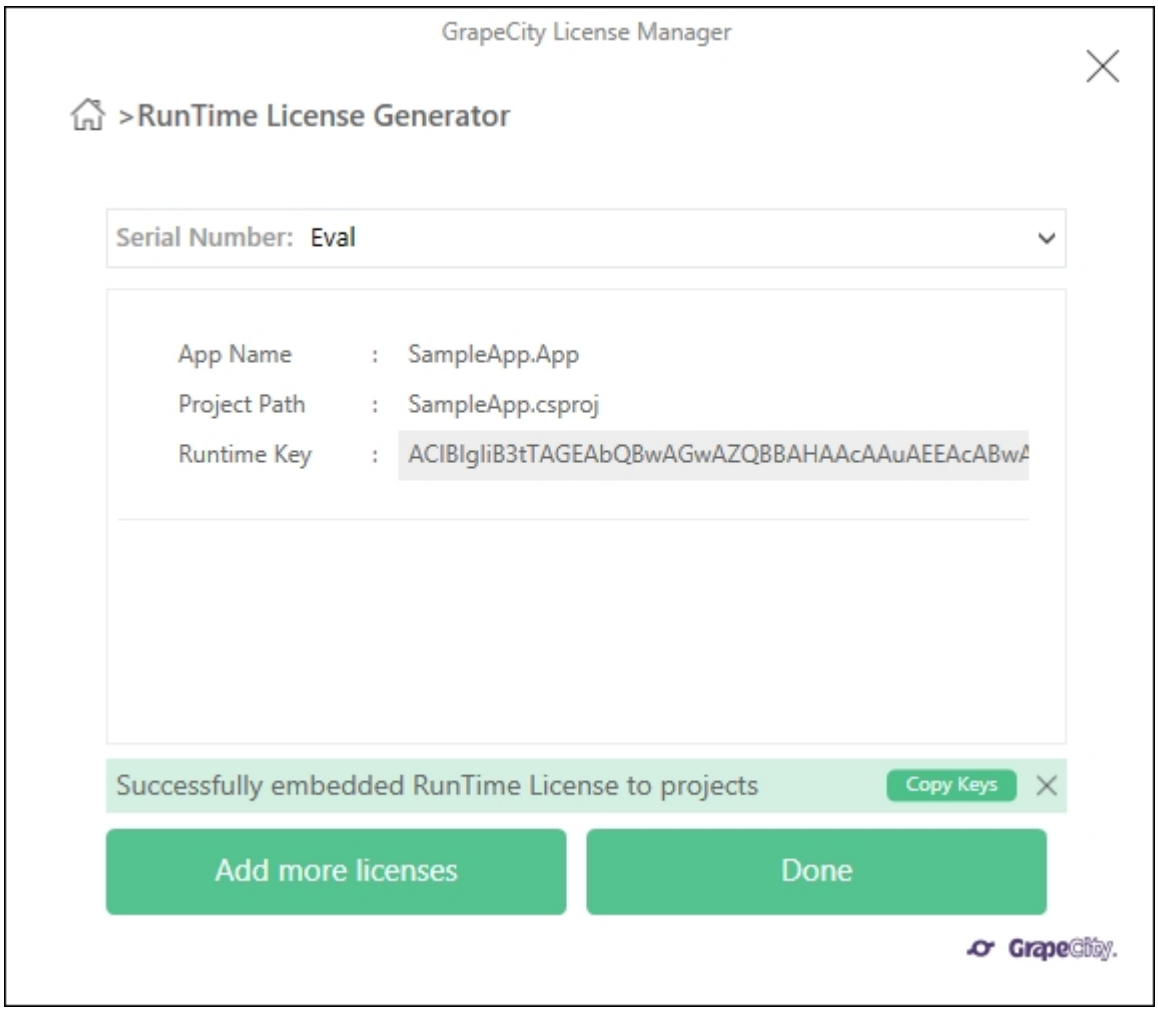
3. From the main License Manager Screen, you can activate or deactivate a serial number, generate a license key, or launch the support portal. To activate a full license serial key, click **Activate/Deactivate Serial Number**. To generate an app license using an already activated serial key or a trial key, click **Generate App (runtime) Licenses**.



4. Select your full license from the drop-down menu at the top. To generate a trial key, select **Eval**.
5. Click on the check box next to the PCL or shared project to be licensed.



6. Click **Generate App (runtime) License** button.
7. Click **Done** to add the **GCDTLicense.xml** file containing the license key to your PCL or shared project.



You can now build and run your licensed app.

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.componentone.com/Members/?ReturnUrl=%2fMyAccount%2fMyXuni.aspx.

   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.

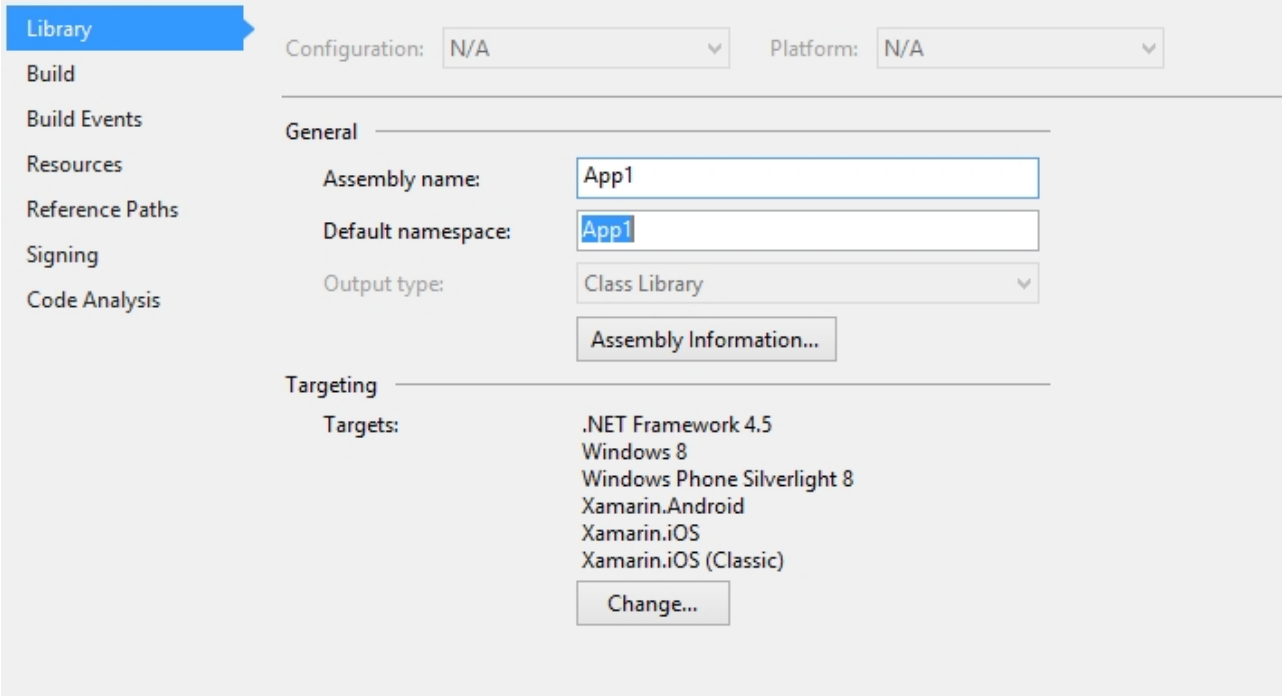
The trial period is limited to 30 days, which begins when you generate your first runtime license. The controls will stop working after your 30-day trial period is over.

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

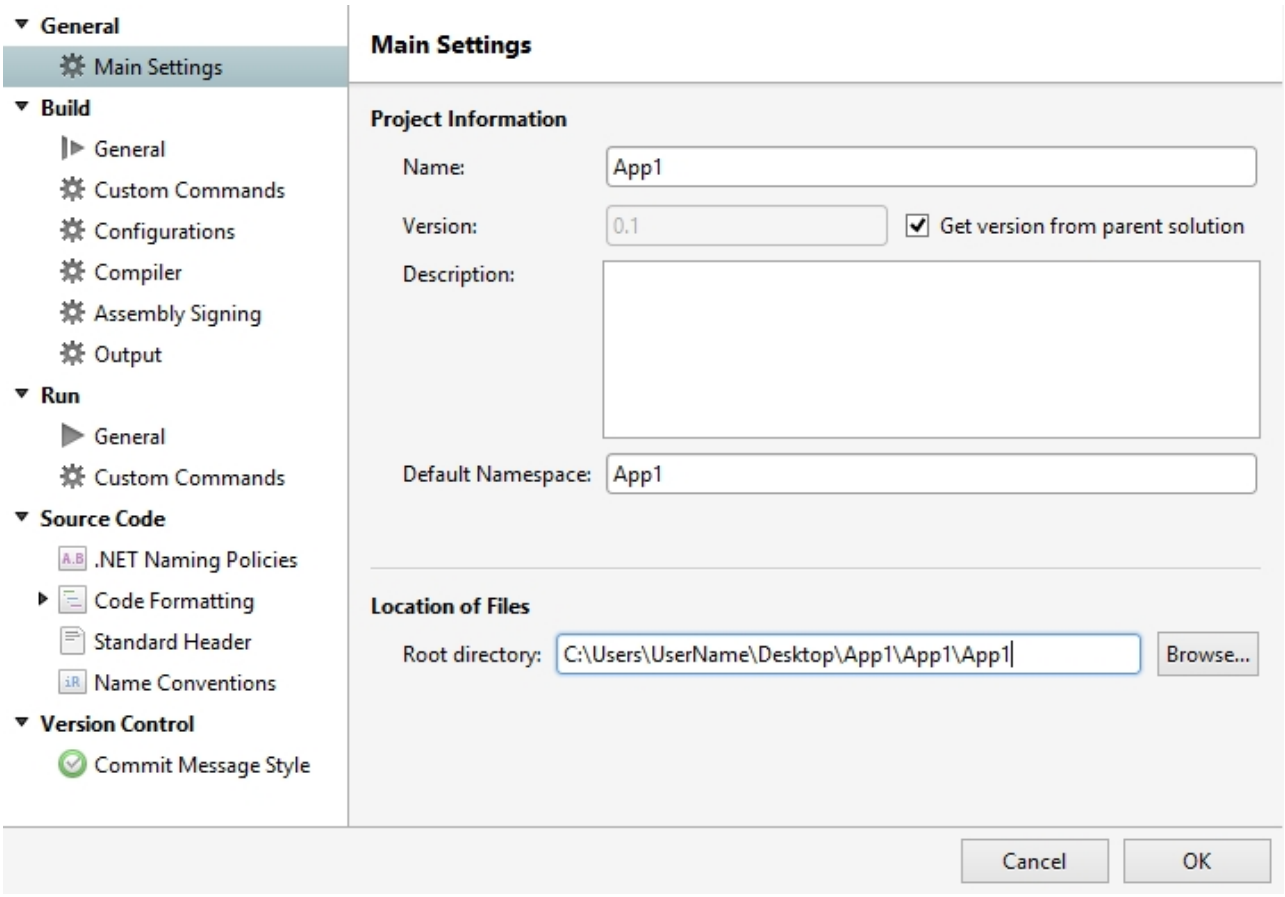
1. Open a pre-existing mobile application.
2. In the Solution Explorer, right-click the project **YourAppName** and select **Properties**.
3. Open the **Library** tab.
4. The application name is the same as the displayed **Default namespace**.



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 for viewing. If you have suggestions on how we can improve our documentation, please send a feedback to the Documentation team. Note that the feedback sent to the Documentation team is for documentation related issues only. Technical support and sales issues should be sent directly to their respective departments.

Note: You must create a 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

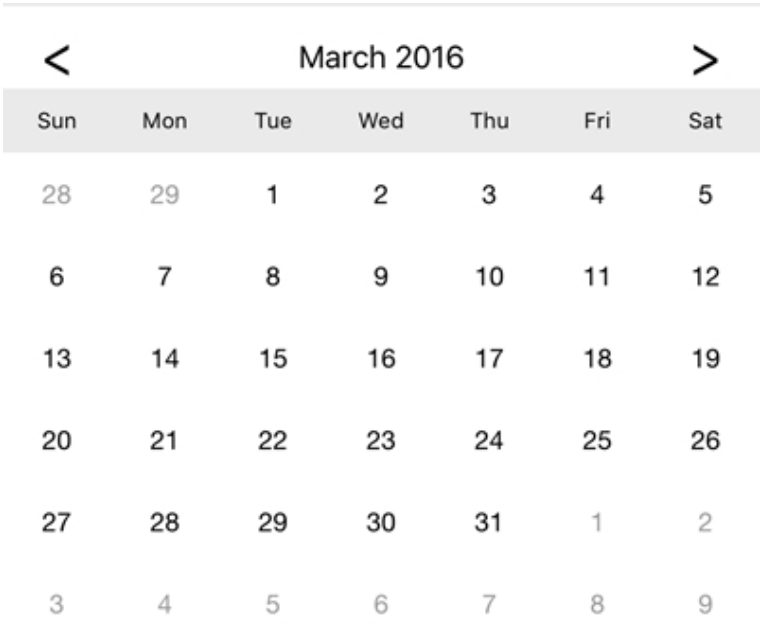
- **Custom Day Content**: Customize the appearance of day slots by inserting custom content.
- **View Modes**: Tap header to switch from month mode to year and decade mode.
- **Appearance**: Easily style different parts of the control with heavy customizations.
- **Date Range Selection**: Simply tap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

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. This topic comprises two steps:

- **Step 1: Add a C1Calendar Control**
- **Step 2: Run the Project**

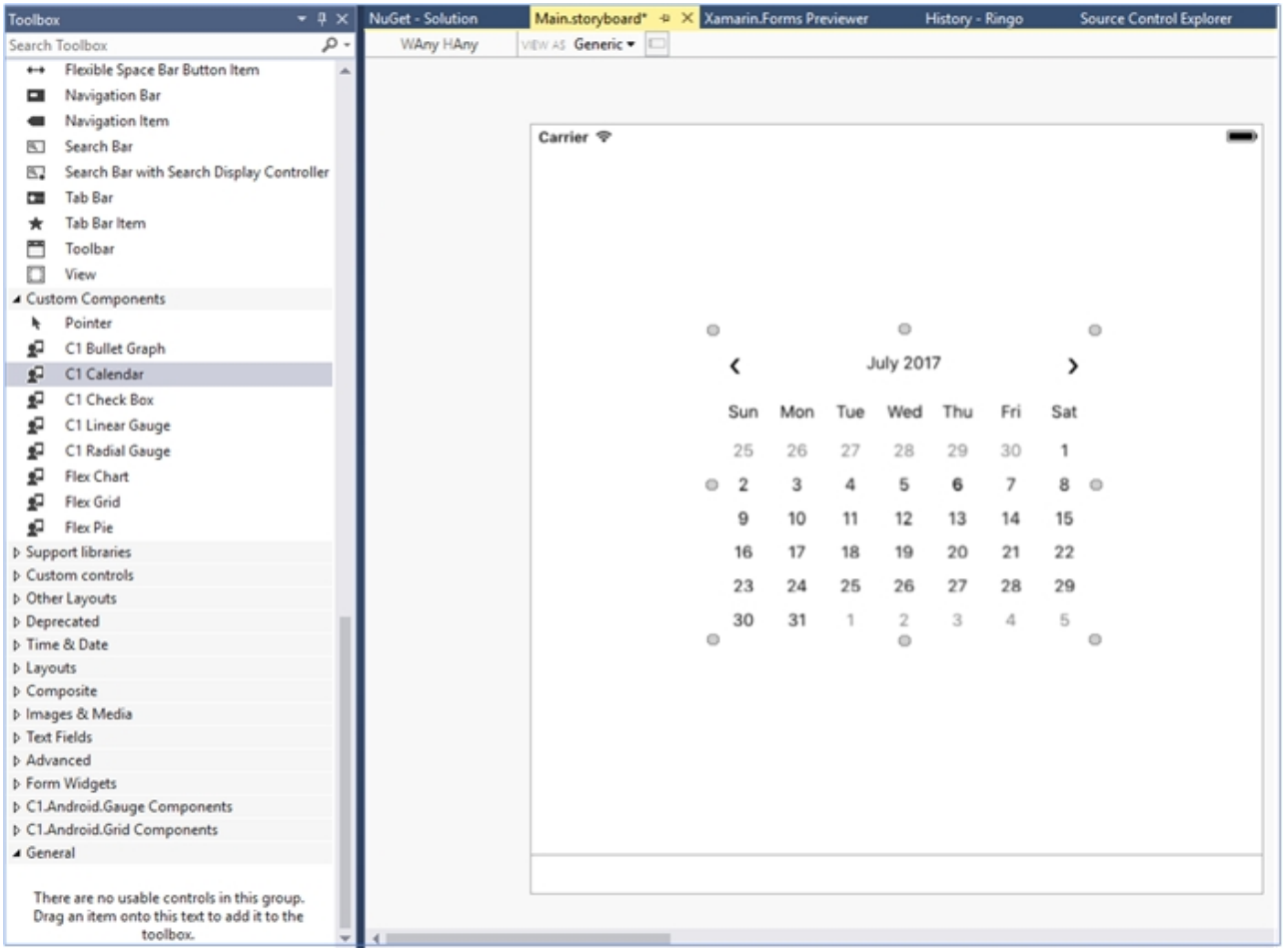
The following image shows how C1Calendar appears after completing the above steps.



Step 1: Add a C1Calendar Control

Add a Calendar 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 your Toolbox under the **Custom Components** tab, drag a C1Calendar onto your ViewController.



Initialize Calendar control in code

To initialize a C1Calendar control, open the ViewController file from the Project Navigator 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#

```
public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            calendar = new C1Calendar();
            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.

CollectionView

The C1CollectionView is a powerful data binding component that is designed to be used with data controls, such as FlexGrid. The C1CollectionView control provides currency, filtering, grouping and sorting services for your data collection. The ICollectionView interface also includes the IEditableCollectionView that defines methods and properties for editing.

The C1CollectionView class implements the following interface:

- ICollectionView: provides current record management, custom sorting, filtering, and grouping.

Key Features

- Provides **filtering**, **grouping** and **sorting** on a data set.
- Can be used with the data collection controls, such as **FlexGrid**.
- Provides **currency for master-detail** support for **iOS** apps.
- Based on the **.NET implementation** of ICollectionView.

C1.CollectionView is .NET Standard compliant while C1.iOS.CollectionView provides the ability to quickly connect your C1CollectionView to a UITableView.

Quick Start

This section describes how to use the CollectionView to provide on demand data loading in a UITableView. It demonstrates how you can use CollectionView for incremental loading within your app.

C#

```
public partial class SimpleOnDemandController : UITableViewController
    {
        public SimpleOnDemandController(IntPtr handle) : base(handle)
        {
        }

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

            // instantiate our on demand collection view
            RefreshControl = new UIRefreshControl();
            var myCollectionView = new SimpleOnDemandCollectionView();
            var myCollectionViewSource = new
SimpleOnDemandCollectionViewSource(TableView, myCollectionView, RefreshControl);
            TableView.Source = myCollectionViewSource;
        }
    }

    public class SimpleOnDemandCollectionView : C1CursorCollectionView<MyDataItem>
    {
        public SimpleOnDemandCollectionView()
        {
            PageSize = 10;
        }
```

```
        public int PageSize { get; set; }
        protected override async Task<Tuple<string, IReadOnlyList<MyDataItem>>>
GetPageAsync(string pageToken, int? count = null)
        {
            var newItems = new List<MyDataItem>();
            await Task.Run(() =>
            {
                // create new page of items
                for (int i = 0; i < this.PageSize; i++)
                {
                    newItems.Add(new MyDataItem(this.Count + i));
                }
            });
            return new Tuple<string, IReadOnlyList<MyDataItem>>("token not used",
newItems);
        }
    }

    public class SimpleOnDemandCollectionViewSource : C1TableViewSource<MyDataItem>
    {
        private string CellIdentifier = "Default";

        public SimpleOnDemandCollectionViewSource(UITableView tableView,
ICollectionView<MyDataItem> collectionView, UIRefreshControl refreshControl = null)
            : base(tableView, collectionView, refreshControl)
        {
        }

        public override UITableViewCell GetItemCell(UITableView tableView, MyDataItem
item)
        {
            UITableViewCell cell = tableView.DequeueReusableCell(CellIdentifier);
            if (cell == null)
                cell = new UITableViewCell(UITableViewCellStyle.Subtitle,
CellIdentifier);

            cell.TextLabel.Text = item.ItemName;
            cell.DetailTextLabel.Text = item.ItemDateTime.ToLongTimeString();

            return cell;
        }
    }

    public class MyDataItem
    {
        public MyDataItem(int index)
        {
            this.ItemName = "My Data Item #" + index.ToString();
            this.ItemDateTime = DateTime.Now;
        }
```

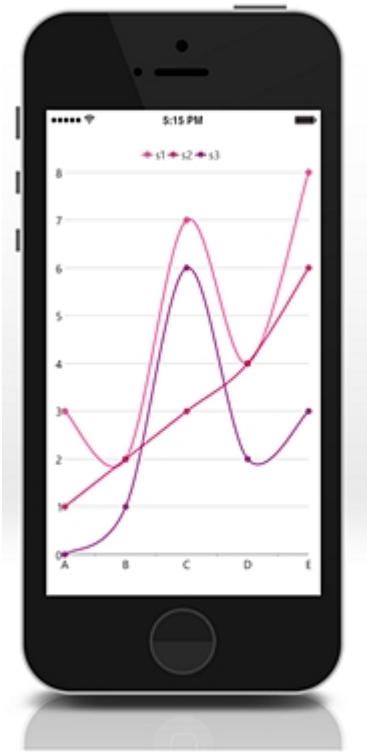
```
        public string ItemName { get; set; }
        public DateTime ItemDateTime { get; set; }

    }
```

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.

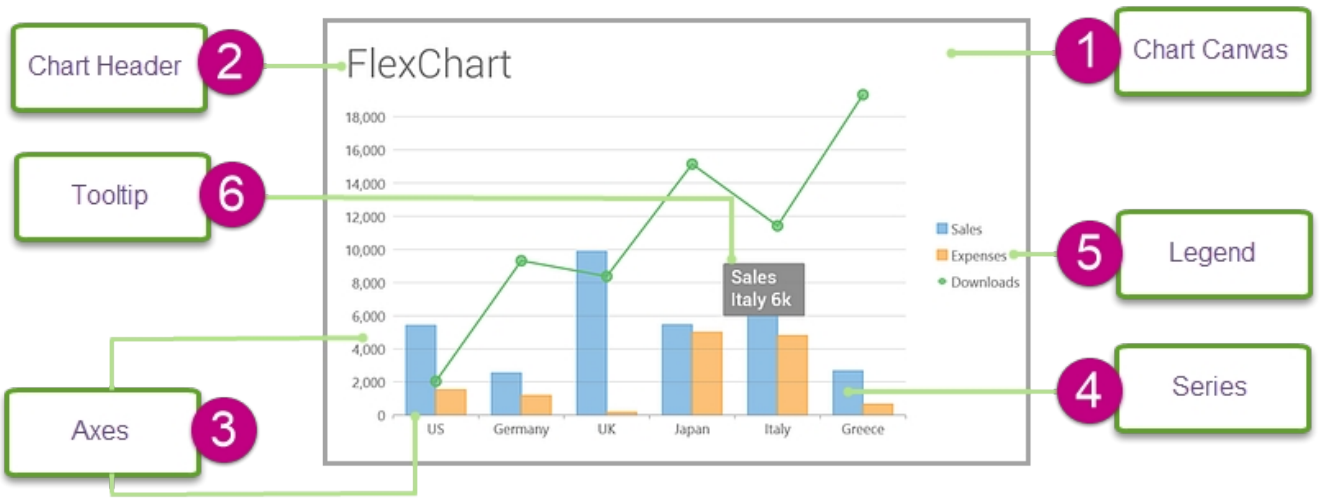


Key Features

- **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

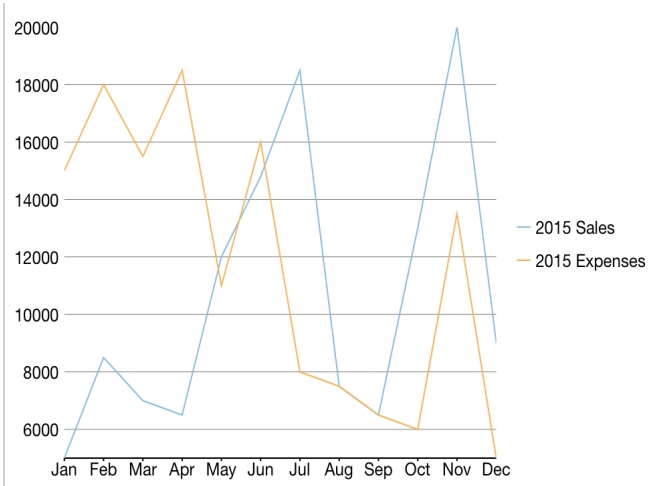
C#

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

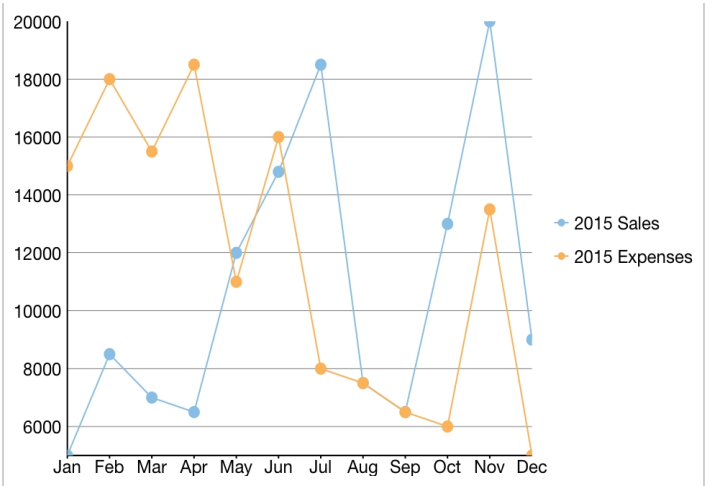
Line and LineSymbol chart

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

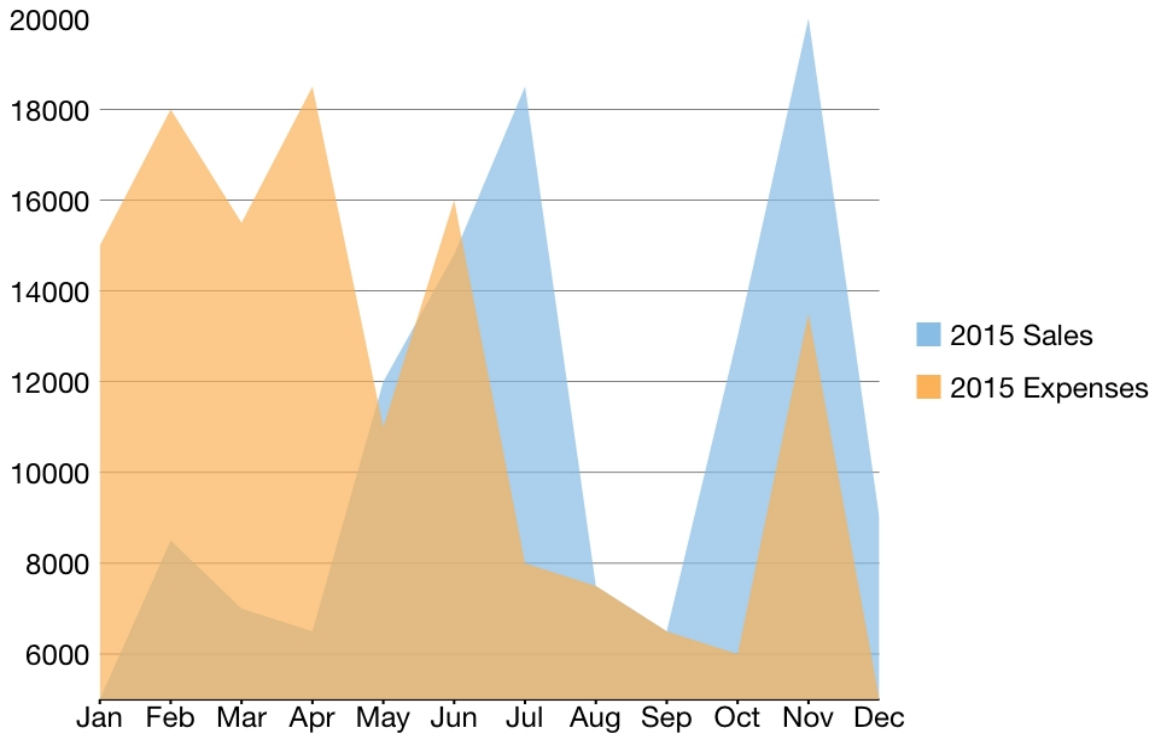


LineSymbol Chart

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.

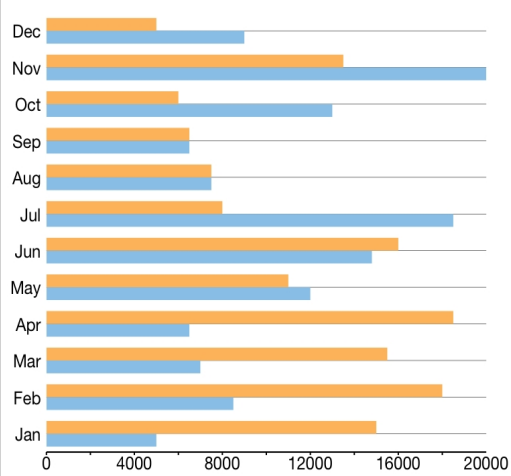
These charts are commonly used to show trends between associated attributes over time.



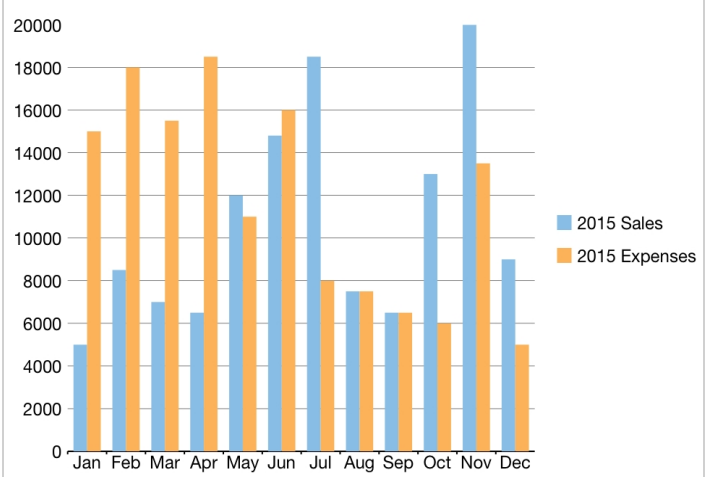
Bar and Column chart

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

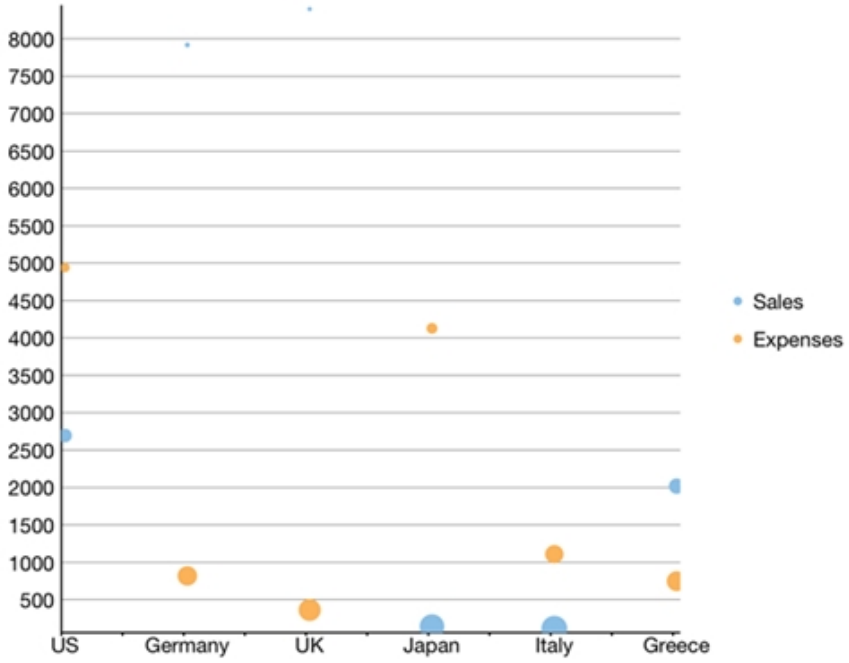


Column Chart

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.

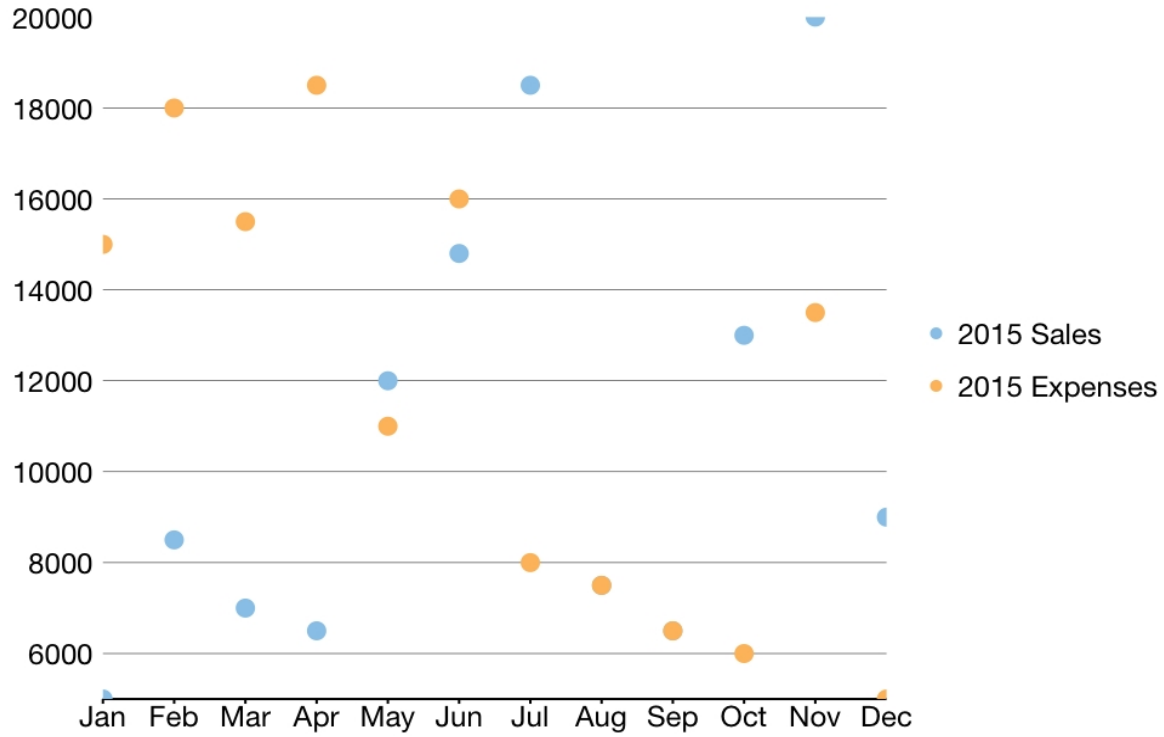
These charts are used to compare entities based on their relative positions on the axis as well as their size.



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

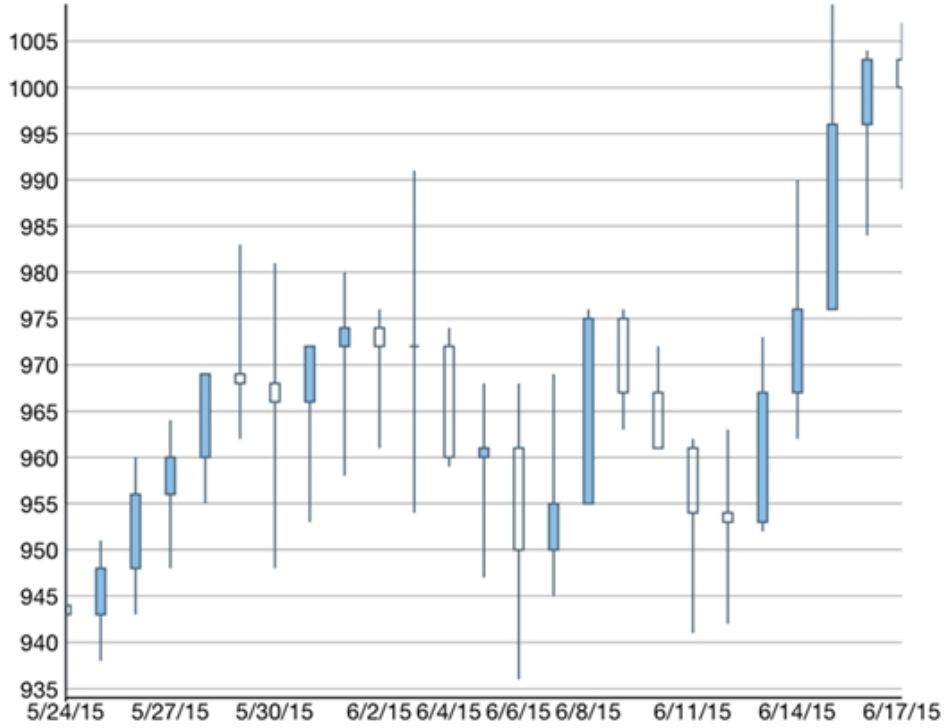
The Candlestick chart is made up of the following elements: **candle**, **wick**, and **tail**.

- **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.

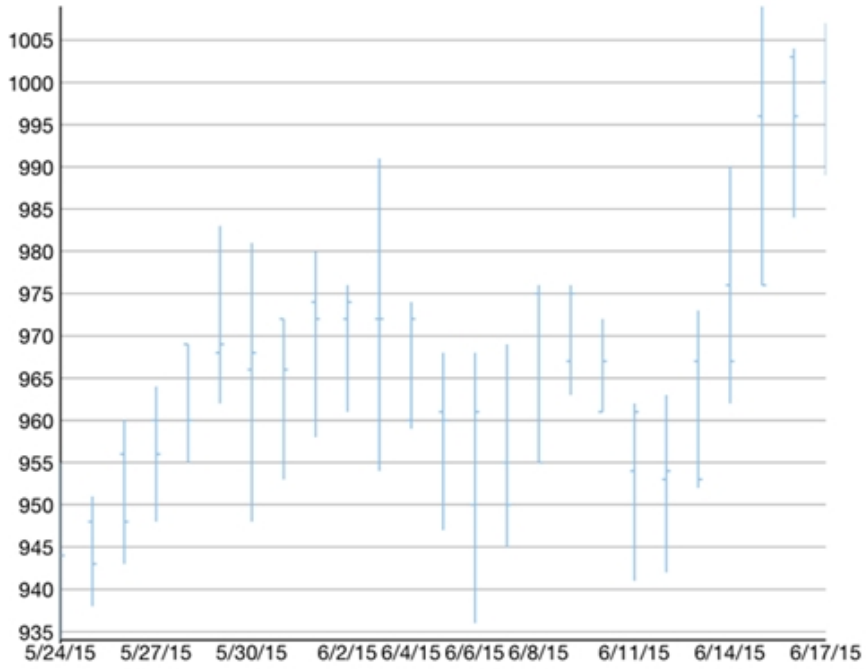
- **x:** Determines the date position along the x axis.
- **high:** Determines the highest price for the day, and plots it as the top of the candle along the y axis.
- **low:** Determines the lowest price for the day, and plots it as the bottom of the candle along the y axis.
- **open:** Determines the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

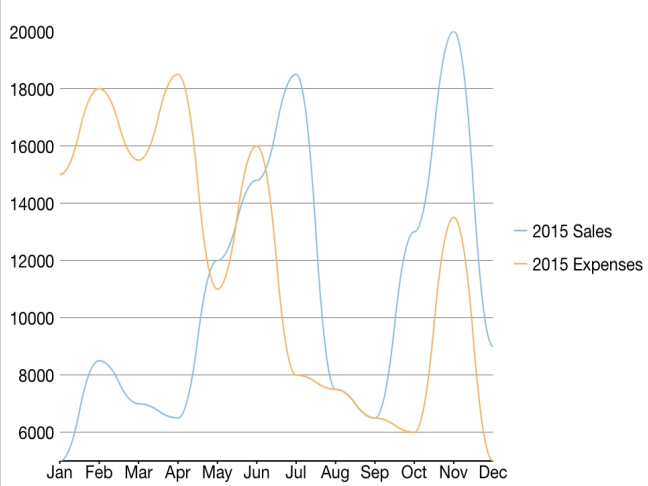
HiLoOpenClose are financial charts that combine four independent values to supply high, low, open and close data for a point in a series. In addition to showing the high and low value of a stock, the Y2 and Y3 array elements represent the stock's opening and closing price respectively.



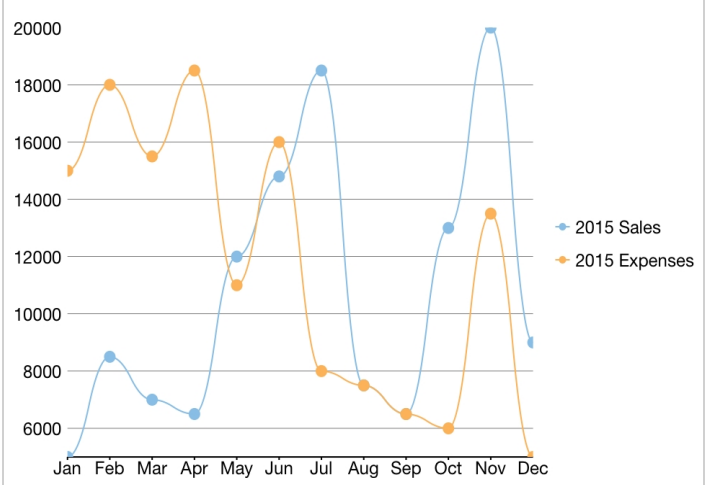
Spline and SplineSymbol chart

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.

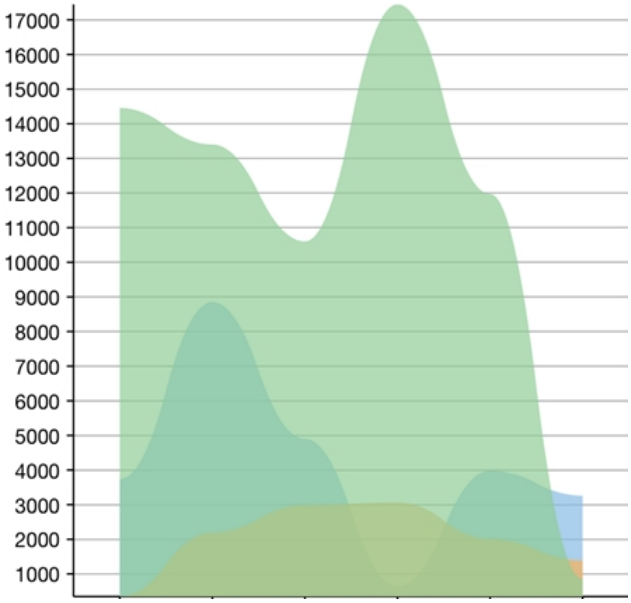
These charts are commonly used to show trends and performance over time, such as product life-cycle.



Spline Chart



SplineSymbol Chart



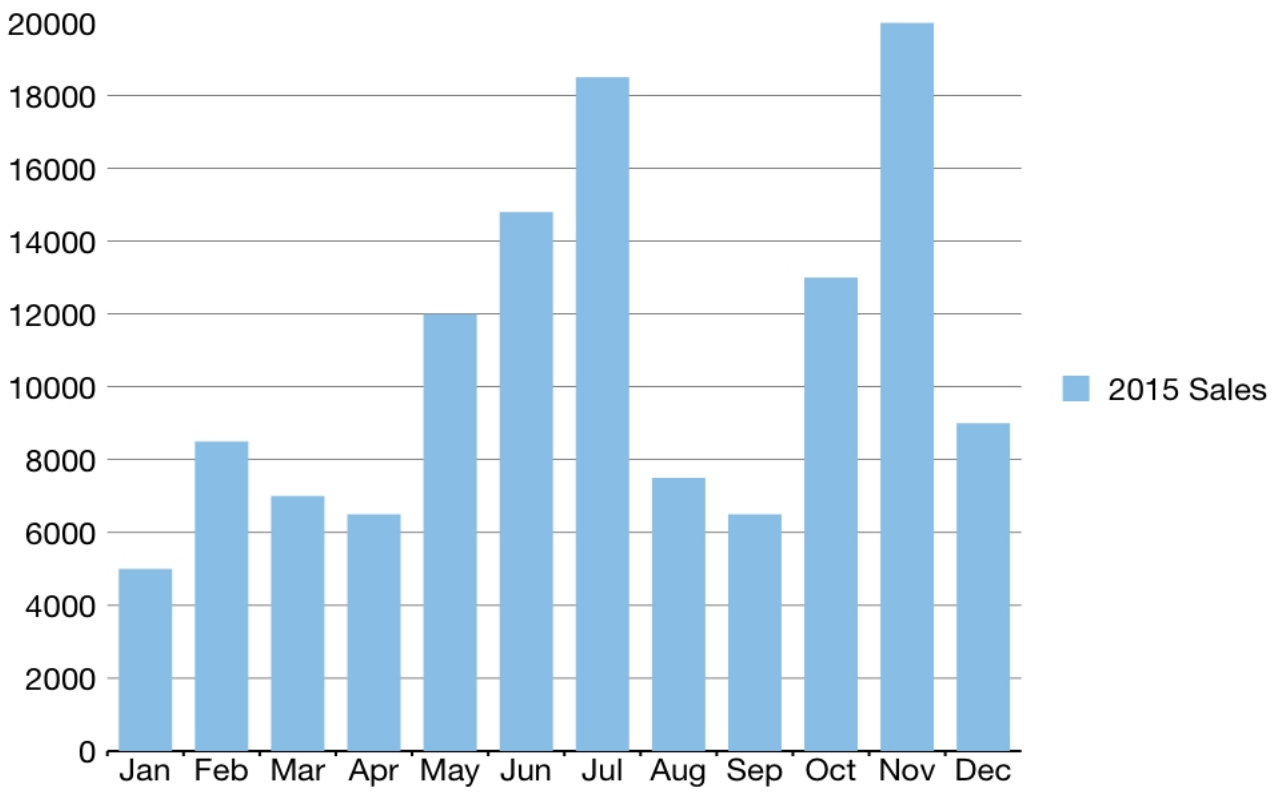
SplineArea Chart

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; }
    }
}
```

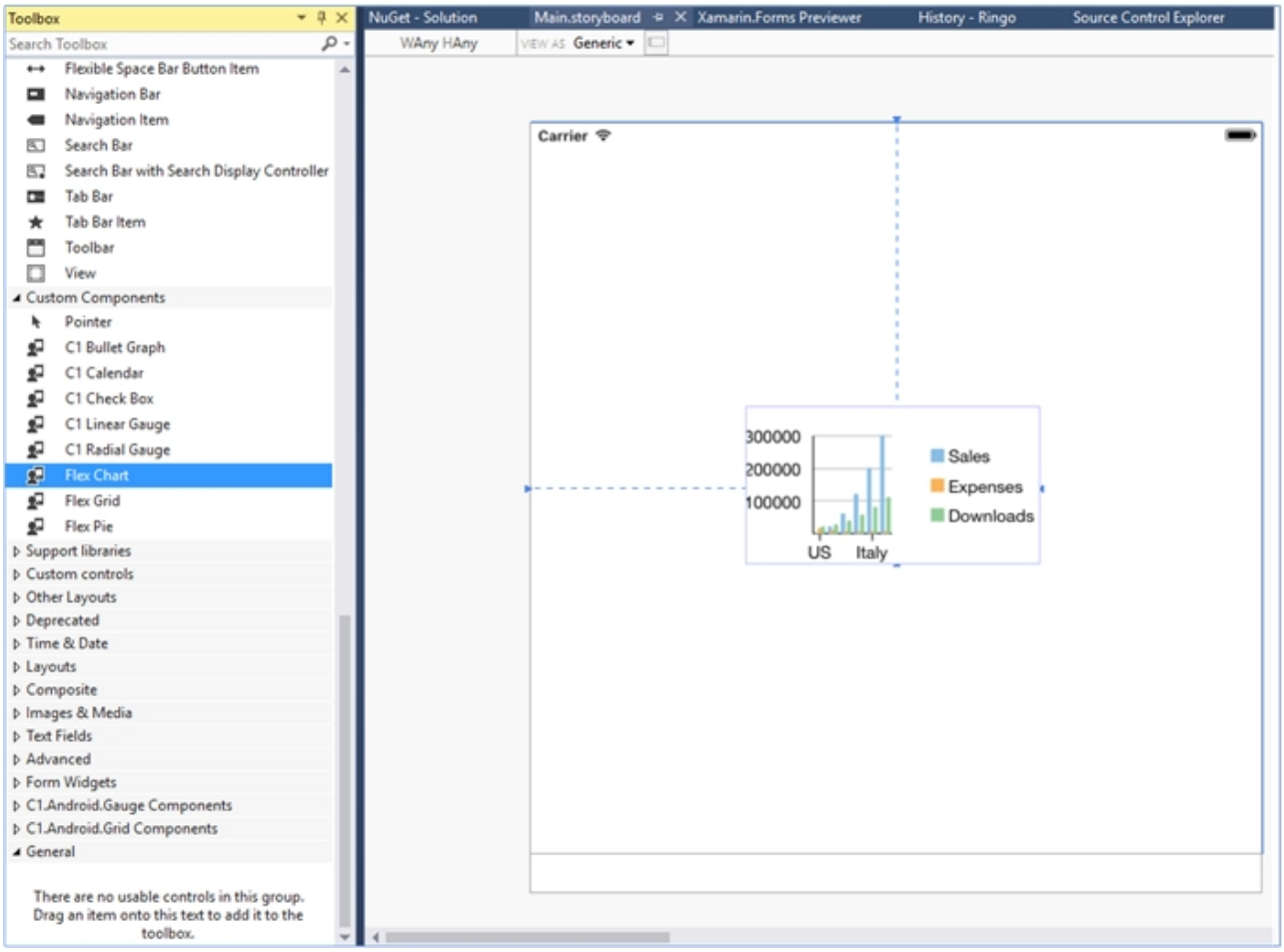
Back to Top

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. Under the **Document Outline**, expand **View Controller** and click **View**.
3. 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

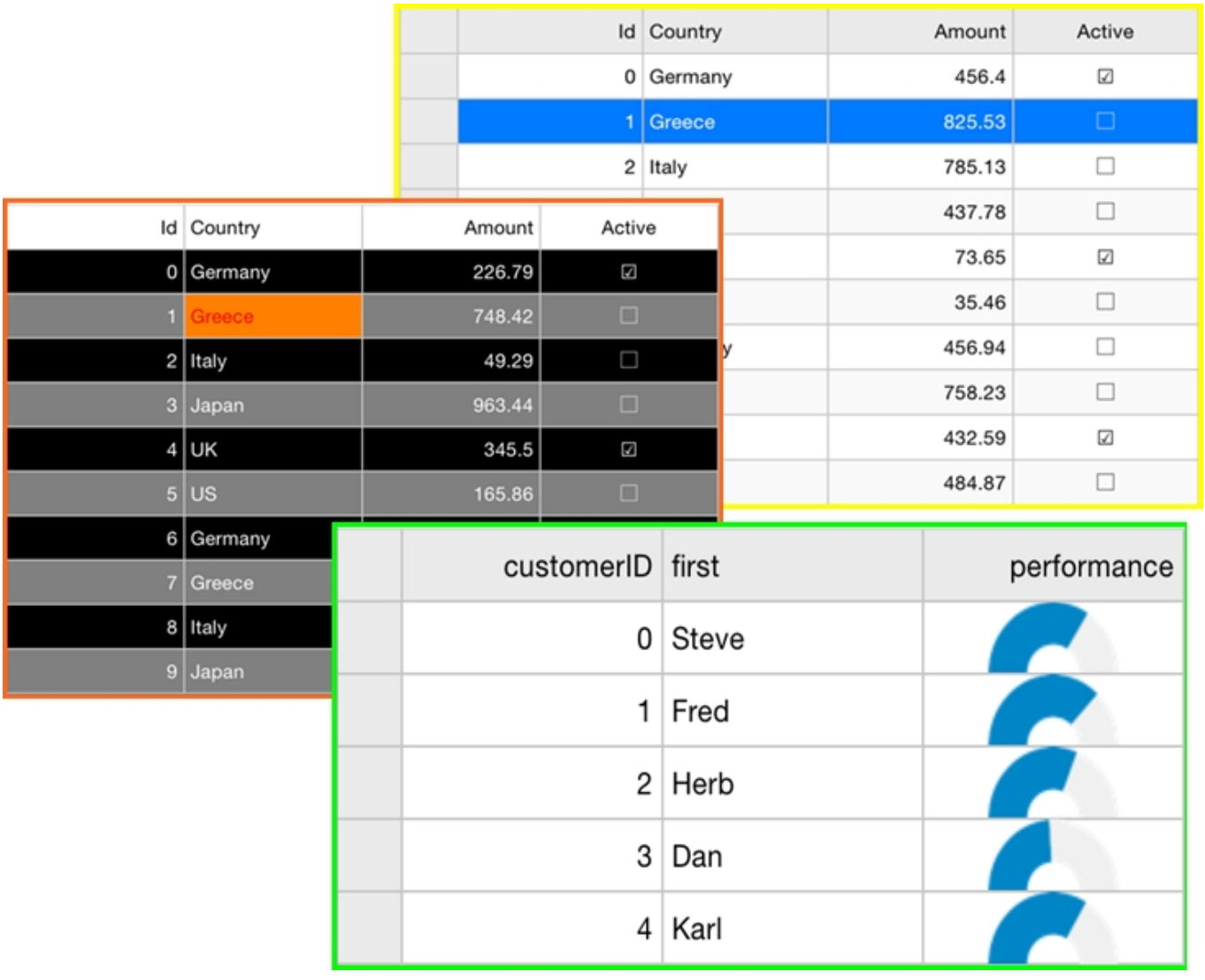
Press **F5** to run the application.

Back to Top

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.



Key Features

- **Auto Generate Columns**: Generates grid columns automatically when set 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.

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 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
    }
```

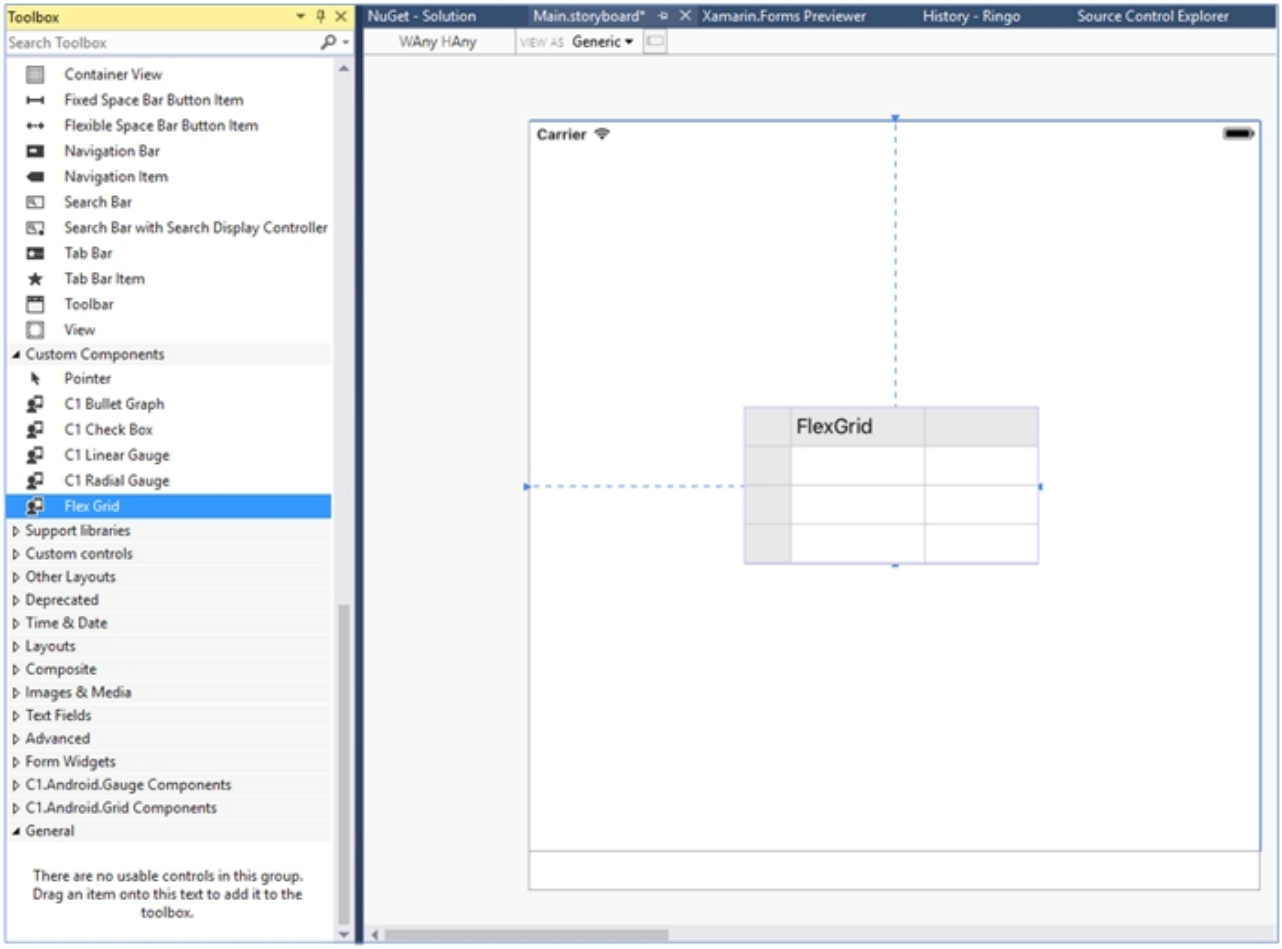
Back to Top

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. Under the **OutlineDocument** , expand **View Controller** and click **View**.
3. In 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
    {
        FlexGrid grid;
```

```
        public GettingStartedController (IntPtr handle) : base (handle)
        {
        }
        public override void ViewDidLoad()
        {
            base.ViewDidLoad();
            grid = new FlexGrid();
            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

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

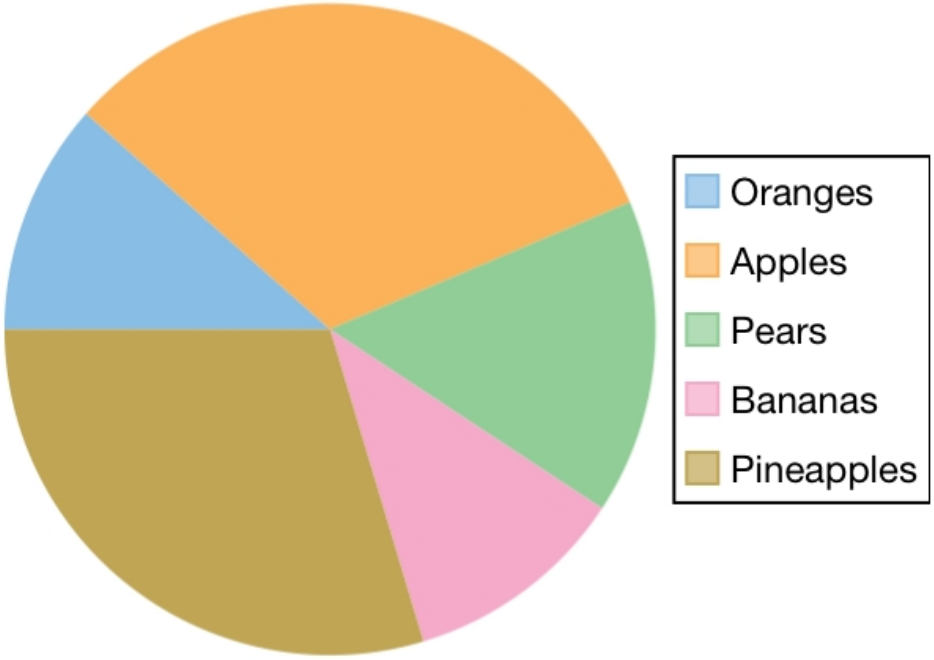
- **Touch Based Labels**: Displays values using touch based labels.
- **Exploding and Donut Pie Charts**: Converts a standard pie chart into an exploding or a donut pie chart.

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;
              }
          }
```

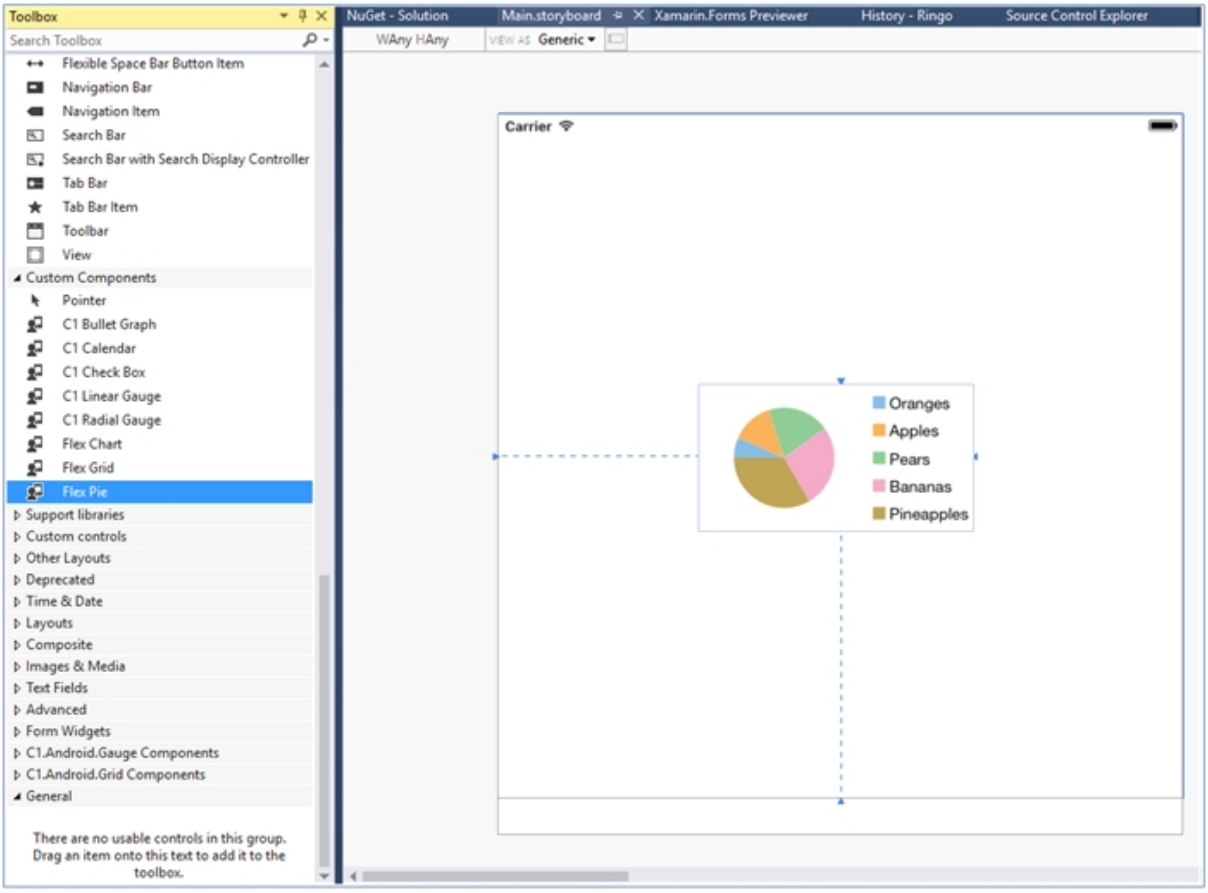
Back to Top

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 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 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

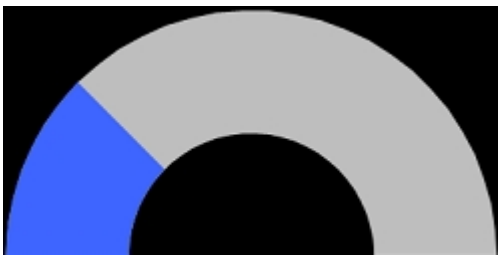
Press **F5** to run the application.

Back to Top

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

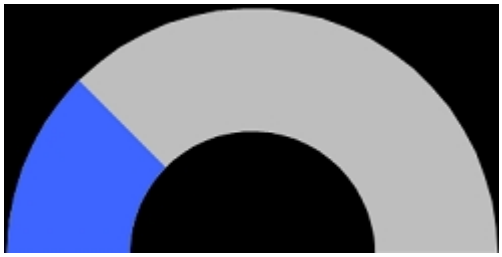
Key Features

- **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.

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.

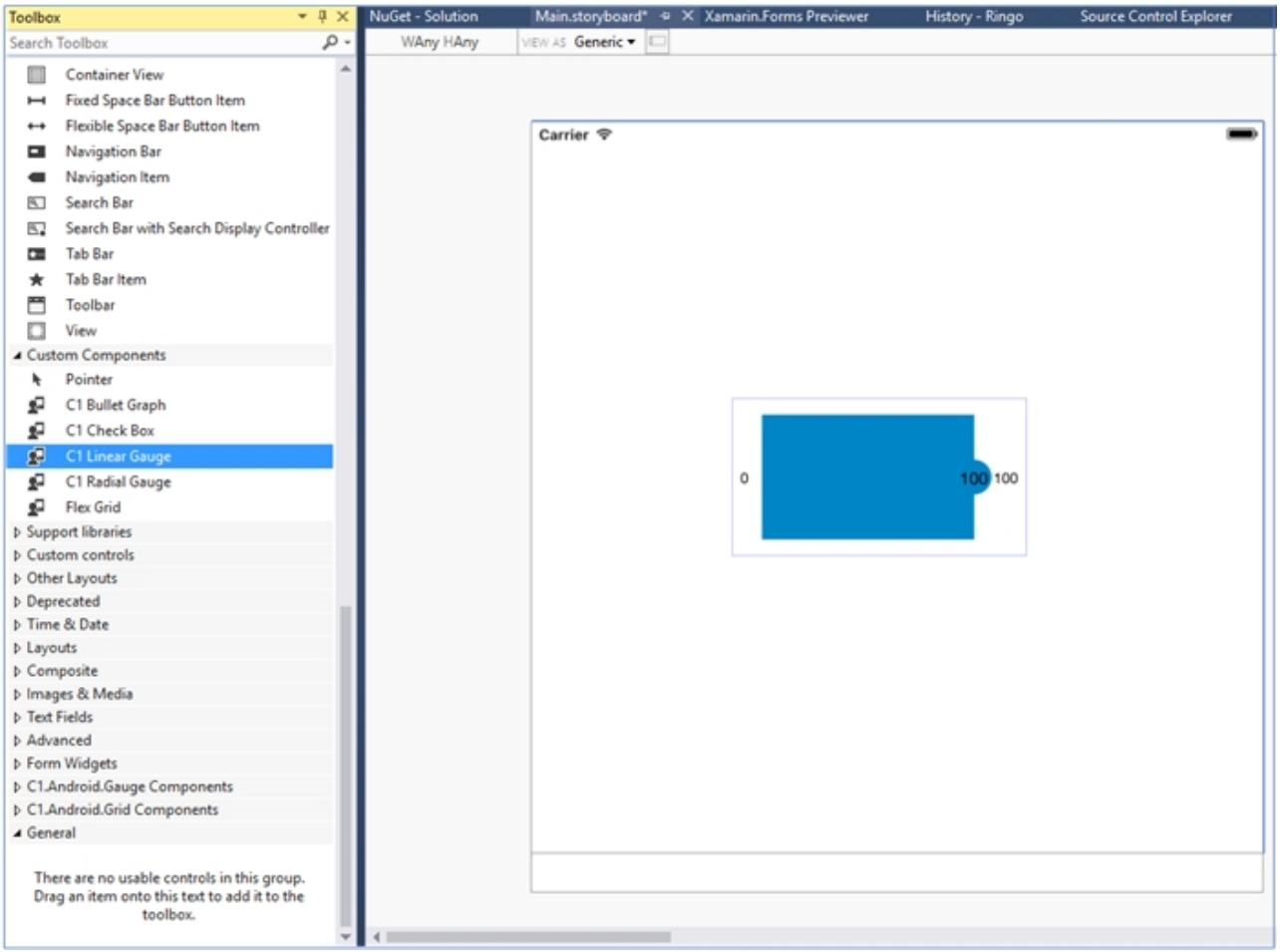
Radial Gauge: A radial gauge displays the value along a circular scale, using a curved pointer. The scale can be rotated as defined by the StartAngle and SweepAngle properties.		A radial gauge is commonly used to denote data such as volume, velocity, etc.
Bullet Graph: A bullet graph displays a single 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);

        }
```

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.

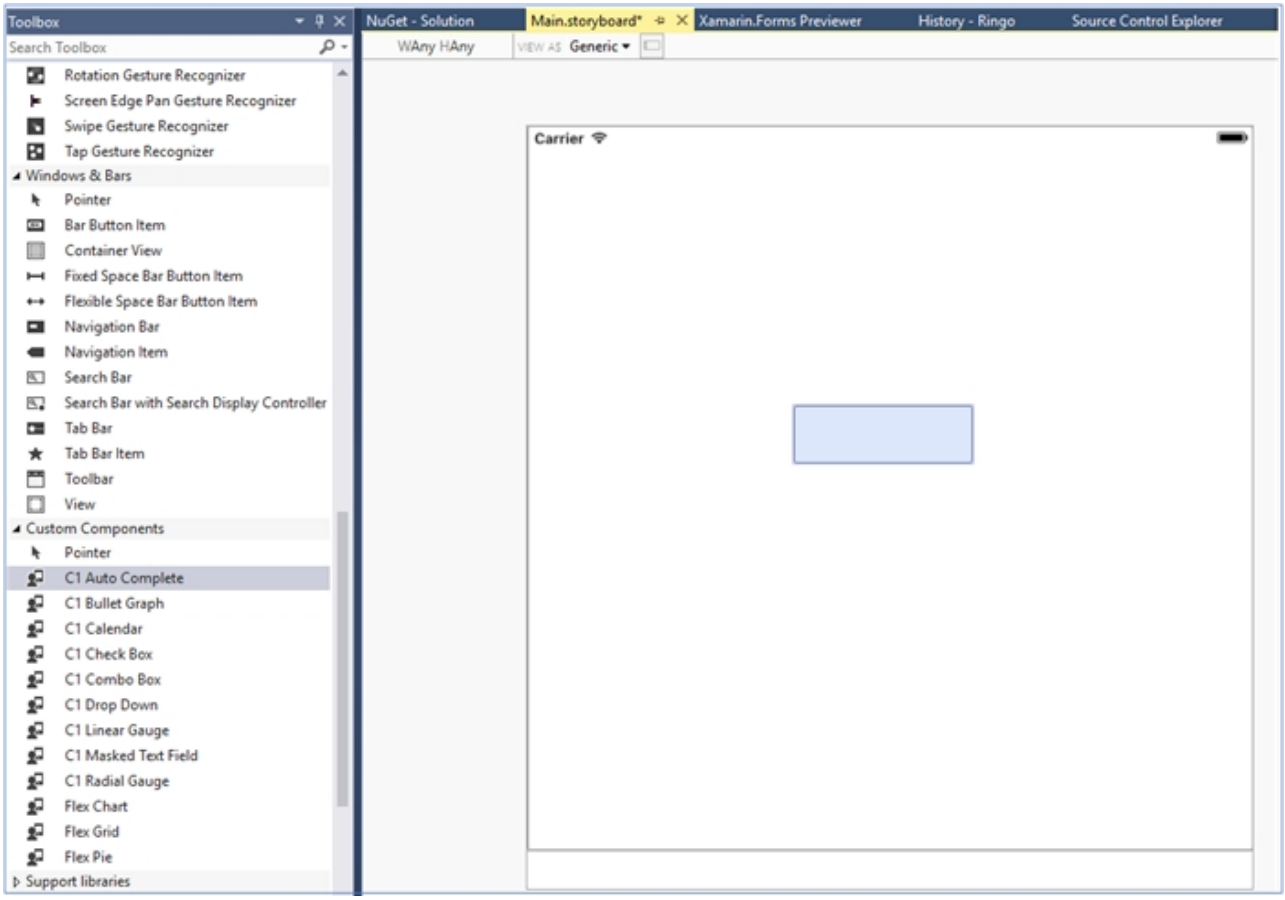
- **Step 1: Add a C1AutoComplete control to ViewController**
- **Step 2: Run the project**

The following image shows a C1AutoComplete control displaying input suggestions as the user types.

Step 1: Add a C1AutoComplete control to ViewController

Add C1AutoComplete control to StoryBoard

1. In the **Solution Explorer**, click MainStoryboard to open the storyboard editor.
2. Under the **OutlineDocument** , expand **View Controller** and click **View**.
3. In the Toolbox under the **Custom Components** tab, drag the C1AutoComplete control onto the ViewController.



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 = GetDemoDataList();
        }

        public static IEnumerable<object> GetDemoDataList()
        {
            List<object> array = new List<object>();

            //NSMutableArray array = new NSMutableArray();
            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>;
    }
```

Step 2: Run the Project

Press F5 to run your application

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: Add an item list to be displayed in the drop-down**
- **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: Add an item list to be displayed in the drop-down

Complete the following steps to add a list of items to be displayed in the drop-down list.

1. Create a new iOS application (Refer Creating a new Xamarin.iOS app for detailed instructions).
2. Add the following code in the ViewController to create an item list.

C#

```
public static IEnumerable<object> GetDemoDataList()
        {
            List<object> array = new List<object>();

            //NSMutableArray array = new NSMutableArray();
            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>;
        }
```

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...";
            ComboBoxEdit.ItemsSource = GetDemoDataList();
        }

        public static IEnumerable<object> GetDemoDataList()
        {
            List<object> array = new List<object>();

            //NSMutableArray array = new NSMutableArray();
            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>;
```

```
        }
```

Step 3: Run the Project

Press **F5** to run the application.

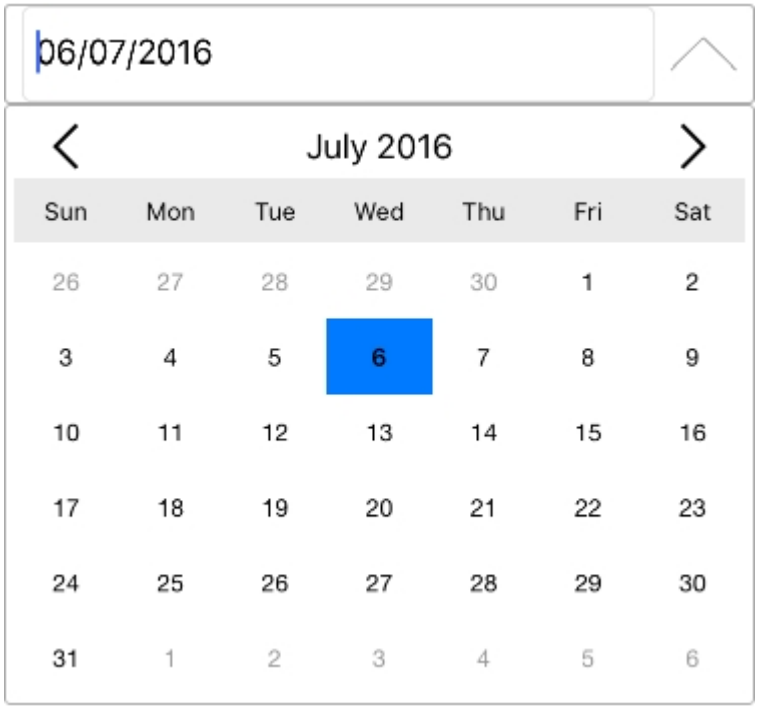
DropDown

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 = new C1DropDown();

        DropDown.DropDownHeight = 300;
        DropDown.DropDownWidth = DropDown.Frame.Size.Width;
        DropDown.DropDownMode = DropDownMode.ForceBelow;
        DropDown.IsAnimated = true;
        d = DropDown;

        maskedField = new C1MaskedTextField();
        maskedField.Mask = "00/00/0000";
        maskedField.BackgroundColor = UIColor.Clear;
        maskedField.BorderStyle = UITextBorderStyle.None;
        DropDown.Header = maskedField;

        calendar = new C1Calendar();
        calendar.SelectionChanged += (object sender,
CalendarSelectionChangedEventArgs e) =>
                                  {
                DateTime dateTime = calendar.SelectedDates[0];
                string strDate = dateTime.ToString("MM-dd-yyyy");
                maskedField.Text = strDate;
                d.IsDropDownOpen = false;
                   };
        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.

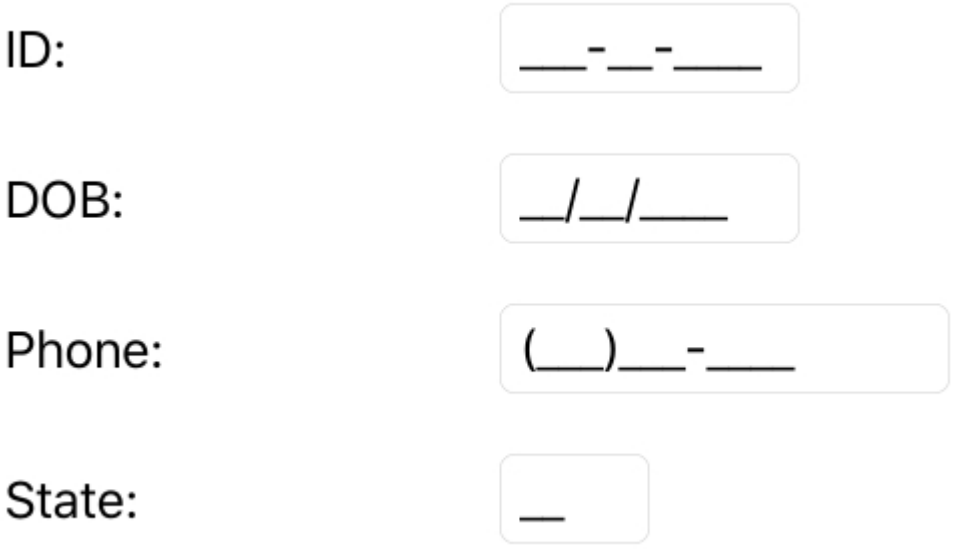
The following table enlists mask symbols supported by the C1MaskedTextField control.

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();
        C1MaskedTextField MaskedID = new C1MaskedTextField();
        C1MaskedTextField MaskedDOB = new C1MaskedTextField();
        C1MaskedTextField MaskedPhone = new C1MaskedTextField()
        C1MaskedTextField MaskedState = new C1MaskedTextField();

        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);
      }
```