ComponentOne

Xamarin.Android 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.Android Controls

ComponentOne Xamarin.Android is a collection of Android UI controls developed by GrapeCity. Xamarin.Android Edition has been optimized for Android development with outstanding built-in features and superior flexibility. It allows you to design views in XML and develop applications similar to the pre-built UI controls in Android.

For existing Xuni new users, the new architecture brings many new features:

- **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 User

New Package Names

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

Xuni.Android.Calendar now corresponds to **C1.Android.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.Android.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.Android.FlexGrid now corresponds to C1.Android.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 CollectionView and is automatically installed when any dependent package is installed.
C1.Android.CollectionView	This is the dependency package to use CollectionView with a native RecyclerView on Android.
C1.Android.Calendar	Installing this NuGet package adds all the references that enable you to use the Calendar control in your Xamarin.Android application.
C1.Android.Core	This is the dependency package for the control NuGet packages and is automatically installed when any dependent package is installed.
C1.Android.Chart	Installing this NuGet package adds all the references that enable you to use the FlexChart and FlexPie controls in your Xamarin.Android application.
C1.Android.Grid	Installing this NuGet package adds all the references that enable you to use the FlexGrid control in your Xamarin.Android application.
C1.Android.Gauge	Installing this NuGet package adds all the references that enable you to use the Gauge controls in your Xamarin.Android application.
C1.Android.Input	Installing this NuGet package adds all the references that enable you to use the Input controls in your Xamarin.Android application.

Redistributable Files

Xamarin.Android 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.Android.Calendar.dll
CollectionView	C1.CollectionView.dll, C1.Android.CollectionView.dll
Core	C1.Android.Core.dll
FlexChart	C1.Android.Chart.dll
FlexGrid	C1.Android.Grid.dll
Gauge	C1.Android.Gauge.dll
Input	C1.Android.Input.dll

System Requirements

ComponentOne Xamarin.Android can be used in applications written for the following mobile operating systems:

Requirements

- Xamarin Platform 2.3.3.193 and above
- Visual Studio 2015 Update 3
- Android 4.2.2 and above

Windows System Requirements

- Windows 8.1 and above

Mac System Requirements

- Xamarin Studio or Visual Studio for Mac
- MacOS 10.12
- Android 7 SDK (API 24) installed

Creating a New Xamarin.Android App

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

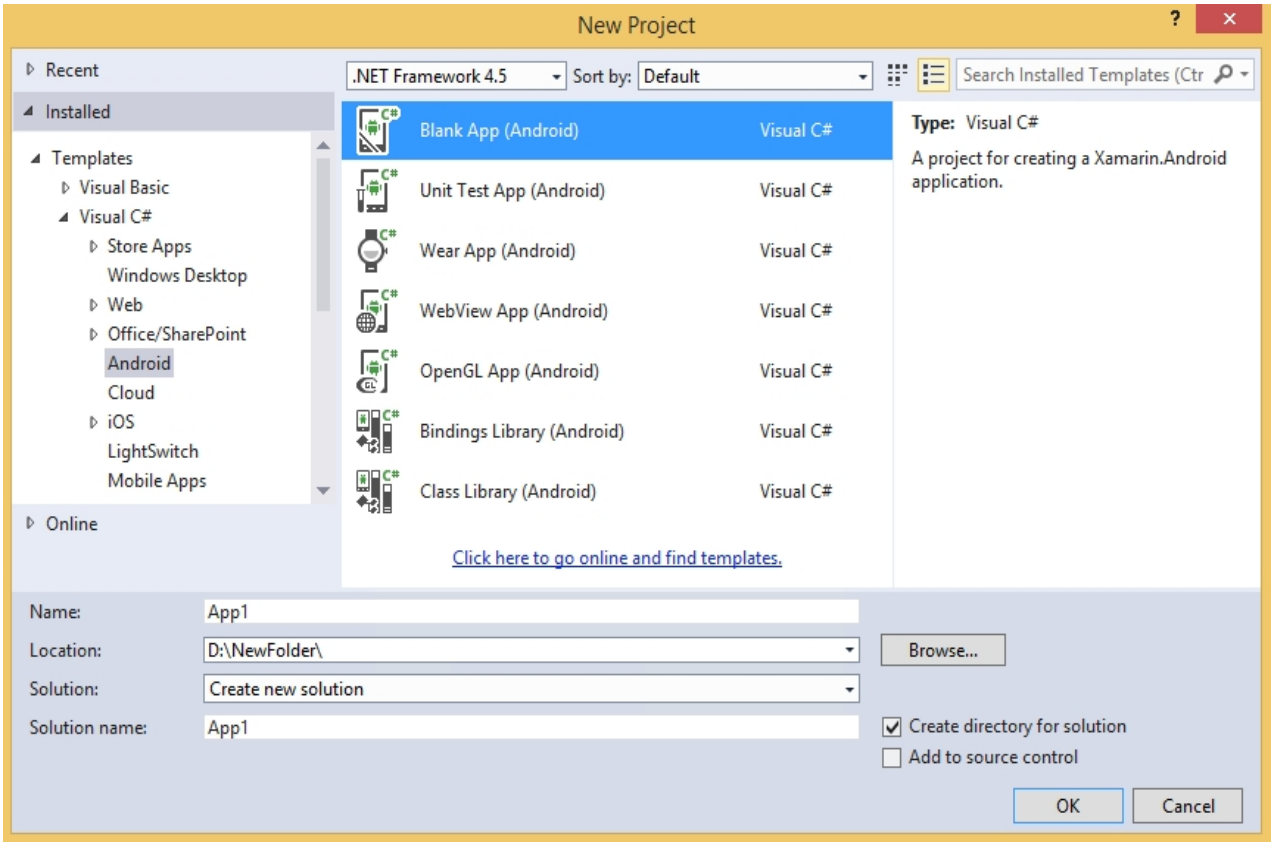
To know more about Xamarin.iOS, visit:

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

Complete the following steps to create a new Xamarin.Android app:

Visual Studio (Windows)

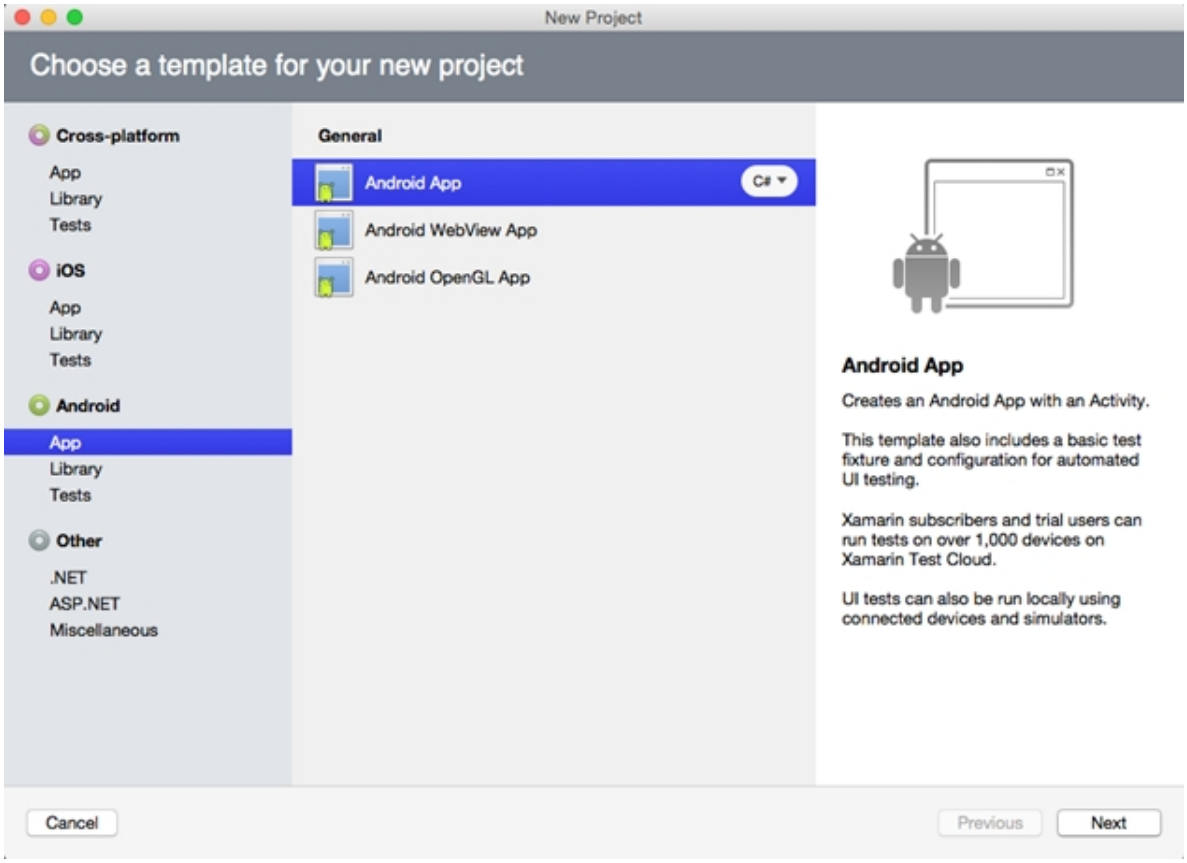
1. Select File | New | Project.
2. Under installed templates, select Visual C# | Android.
3. In the right pane, select Blank App.
4. Type a name for your app and select a location to save it.



5. Click OK.

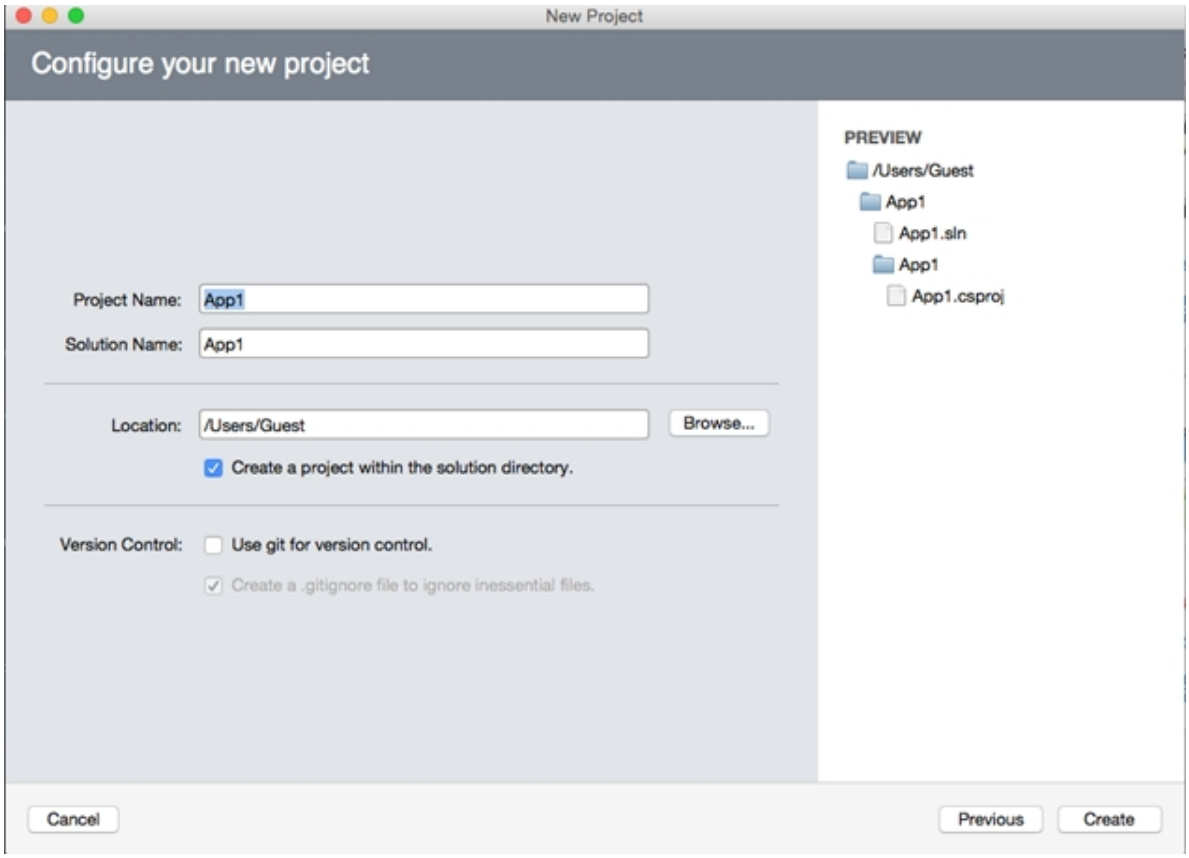
Visual Studio for Mac (macOS)

1. Select **File** | **New Solution**.
2. Select **Android** | **App**.
3. In the right pane, select **Android App**.
4. Type a name for your app and select a location to save it.
5. Click **OK**.



6. Add a name for your app and select a location to save it.

7. Click **Next**.



8. Click **Create**.

Adding NuGet Packages to your App

To install Installing 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 Android platform, related references need to be added to your project. Complete the following steps to add Xamarin references to your project.

Visual Studio (PC)

1. Open an existing or a new Xamarin.Android App.
2. In the **Project** menu, select **Manage NuGet Packages.**
3. In the **Manage NuGet Packages** dialog, click **Online** and then click **GrapeCity.**
4. Click **Install** next to C1.Android.ControlName (eg. C1.Android.Chart). This adds the references for Xamarin control.
5. Click **I Accept** to accept the license and then click **Close** in the Manage NuGet Packages dialog.

Visual Studio for Mac

1. Open an existing or a new Android app.
2. In the **Solution Explorer**, right-click the project and select **Add | Add NuGet Packages.** The **Add NuGet 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.Android.Control and click the Add Package button. This adds the references for the Xamarin control.

To manually create a Xamarin feed source

Visual Studio (PC)

1. In the Tools menu, select **NuGet Package Manager | Package Manager Settings**. The **Options** dialog appears.
2. In the left pane, select **Package Sources.**
3. Click the **Add (+)** button in top right corner to add a new source 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 Xamarin feed has now been added as another NuGet feed sources.

To install Xamarin using the new feed:

1. Open an existing or a new Android app.
2. Select **Project | Manage NuGet Packages.**
3. In the **Manage NuGet Packages** dialog, go to the **Online** drop down and select Xamarin. The available Xamarin packages get displayed in the right pane.
4. Click **Install** next to the NuGet package (for example, C1.Android.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.

Visual Studio for Mac

1. In the Projects menu, select **Add Packages.**
2. In the **Add Packages** dialog, open the drop-down menu in the top left corner and select **Configure Sources**. The **Preferences** dialog appears.
3. In the left pane, expand **Packages** and select **Sources.**
4. Click the **Add** button to open the **Add Package Source** dialog.
5. Set the Name of the new package source as Xamarin and the URL as http://nuget.grapecity.com/nuget/.
6. Click **Add Source** button to add the Xamarin feed as a new feed source.
7. Click **OK**.

To install Xamarin using the new feed:

1. Open an existing or a new Android app.
2. In the **Solution Explorer**, right-click the project and select **Add | Add Packages**. The **Add Packages** dialog appears.
3. In the **Add Packages** dialog, open the drop-down menu in the top left corner and select Xamarin. The available packages are displayed.
4. Select the package (for example C1.Android.Chart) 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 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 or Xamarin 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 **MainActivity.cs** and set the runtime license inside the **OnCreate()** method as shown below.

      C#

      ```
      C1.Android.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.

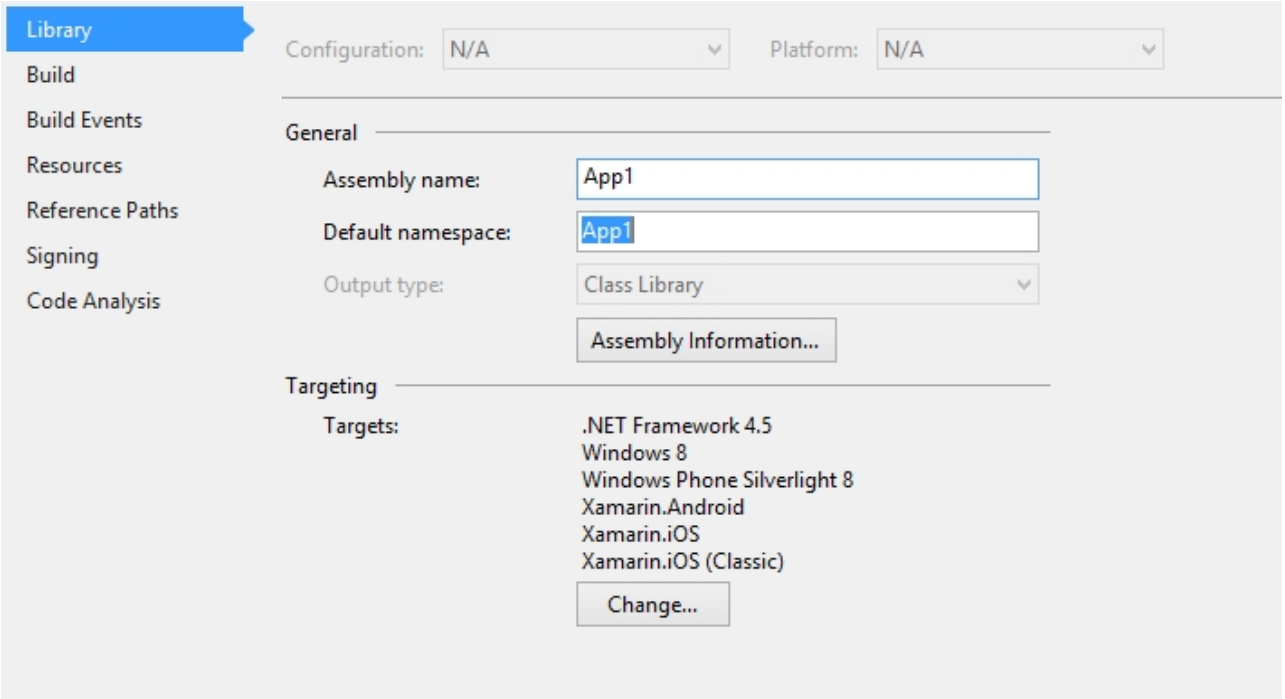
> The trial period is limited to 30 days, which begins when you generate your first runtime license. The controls will stop working after your 30-day trial period is over.

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

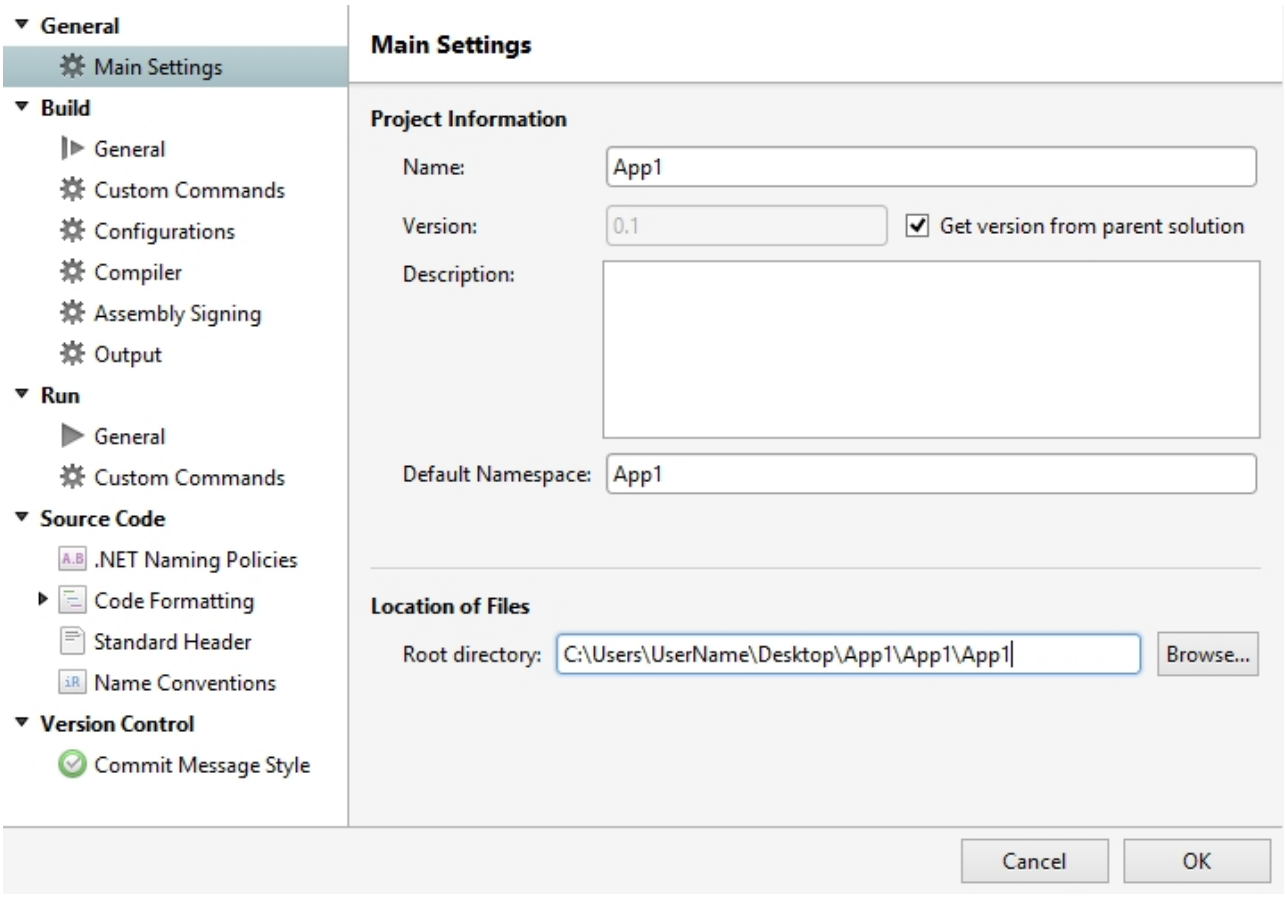
1. Open a pre-existing mobile application.
2. In the Solution Explorer, right-click the project **YourAppName** and select **Properties**.
3. Open the **Library** tab.
4. The application name is the same as the displayed **Default namespace**.



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 android application and select a date on the calendar at runtime. This topic comprises of two steps:

- **Step 1: Add a Calendar control**
- **Step 2: Run the project**

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

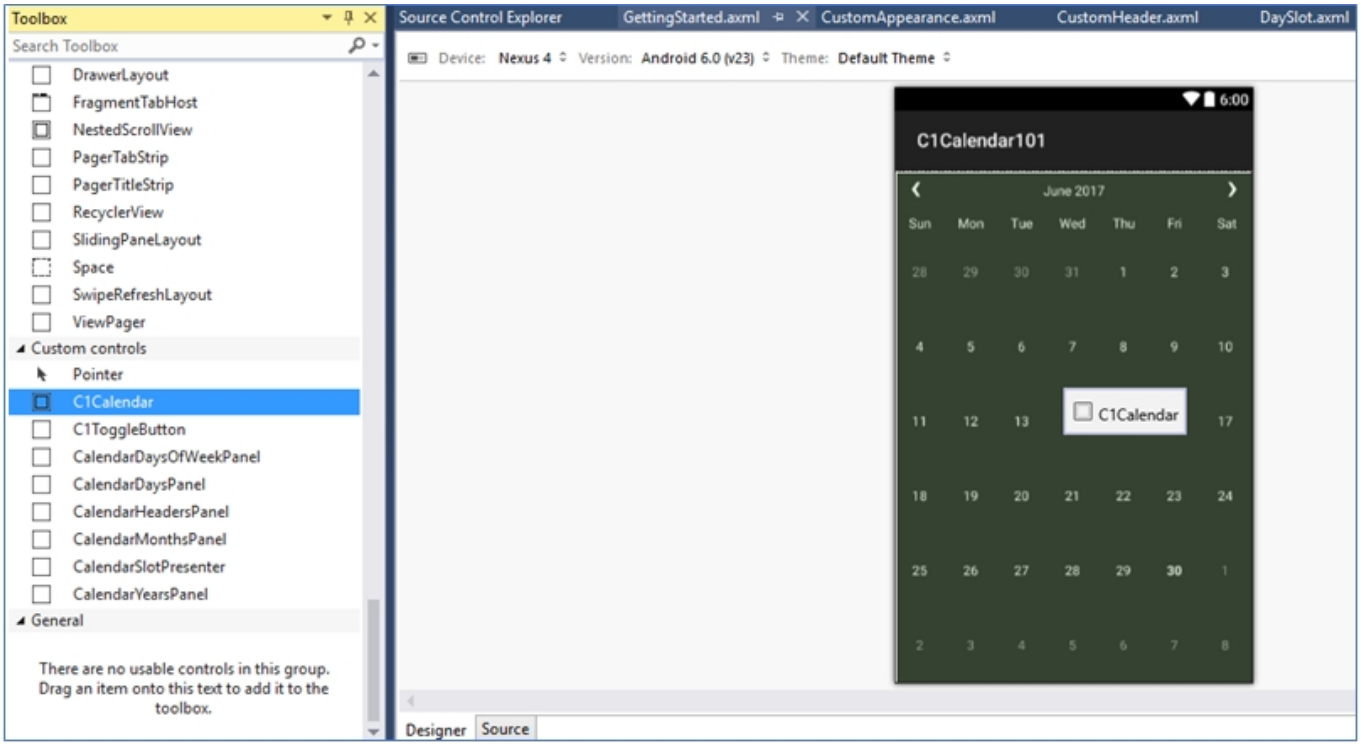
Step 1: Add a Calendar control

To add C1Calendar control to your layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with the code below.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<C1.Android.Calendar.C1Calendar
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/Calendar" />
```

Alternatively, you can drag a C1Calendar control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to the OnCreate method to initialize your layout.

C#

```
protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            ActionBar.SetDisplayHomeAsUpEnabled(true);

            SetContentView(Resource.Layout.GettingStarted);
        }
```

Step 2: Run the project

Press **F5** to run the application.

CollectionView

C1CollectionView is a powerful data binding component that is designed to be used with data controls, such as FlexGrid. CollectionView 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 interfaces:

- 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 with a RecyclerView. It demonstrates how you can use a CollectionView for incremental loading within your app.

C#

```
public class SimpleOnDemandActivity : Activity
    {
        private SimpleOnDemandCollectionView _collectionView;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            SetContentView(Resource.Layout.SimpleOnDemand);

            SwipeRefresh = FindViewById<SwipeRefreshLayout>
(Resource.Id.SwipeRefresh);
            RecyclerView = FindViewById<RecyclerView>(Resource.Id.RecyclerView);


            _collectionView = new SimpleOnDemandCollectionView();
            RecyclerView.SetLayoutManager(new LinearLayoutManager(this));
            RecyclerView.SetAdapter(new SimpleOnDemandAdapter(_collectionView));

            SwipeRefresh.Refresh += OnRefresh;
        }

        public SwipeRefreshLayout SwipeRefresh { get; set; }
        public RecyclerView RecyclerView { get; set; }

        private async void OnRefresh(object sender, System.EventArgs e)
        {
            try
            {
                SwipeRefresh.Refreshing = true;
                await _collectionView.RefreshAsync();
            }
            finally
            {
                SwipeRefresh.Refreshing = false;
            }
        }
    }

    internal class SimpleOnDemandAdapter : C1RecyclerViewAdapter<MyDataItem>
```

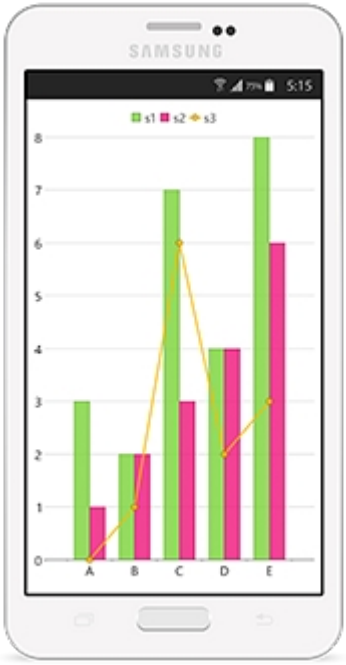
```
    {

        public SimpleOnDemandAdapter(ICollectionView<MyDataItem> collectionView)
            : base(collectionView)
        {
        }

        protected override RecyclerView.ViewHolder OnCreateItemViewHolder(ViewGroup
parent)
        {
            var view = LayoutInflater.From(parent.Context)
                                .Inflate(Resource.Layout.ListItem, null, false);
            return new SimpleOnDemandViewHolder(view);
        }

        protected override void OnBindItemViewHolder(RecyclerView.ViewHolder holder,
int position)
        {
            var h = holder as SimpleOnDemandViewHolder;
            var item = CollectionView[position];
            h.SetTitle(item.ItemName);
            h.SetSubtitle(item.ItemDateTime.ToLongTimeString());
        }
    }

    internal class SimpleOnDemandViewHolder : RecyclerView.ViewHolder
    {
        private TextView _title;
        private TextView _subTitle;

        public SimpleOnDemandViewHolder(View itemView)
            : base(itemView)
        {
            _title = itemView.FindViewById<TextView>(Resource.Id.Title);
            _subTitle = itemView.FindViewById<TextView>(Resource.Id.Subtitle);
            var icon = itemView.FindViewById<ImageView>(Resource.Id.Icon);
            icon.Visibility = ViewStates.Gone;
        }

        internal void SetTitle(string title)
        {
            _title.Text = title;
        }

        internal void SetSubtitle(string subTitle)
        {
            _subTitle.Text = subTitle;
        }
    }
```

```
    public class SimpleOnDemandCollectionView : C1CursorCollectionView<MyDataItem>
    {
        public SimpleOnDemandCollectionView()
        {
            PageSize = 20;
        }

        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 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 Android 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.

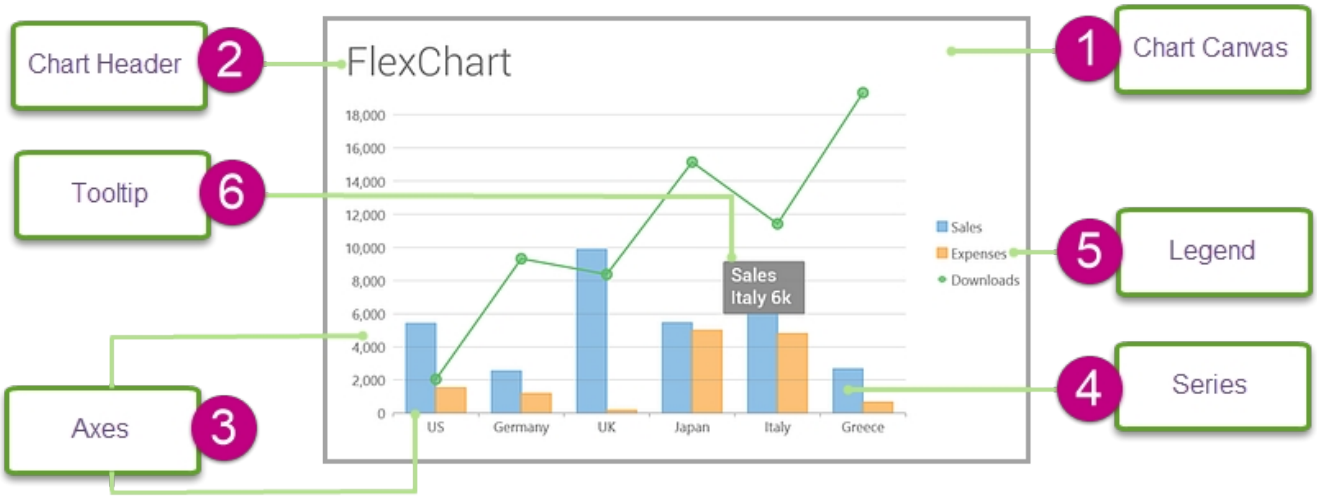


Key Features

- **Chart Type**: Change a line chart to a bar chart or any other chart type by setting a single property. FlexChart supports over ten 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

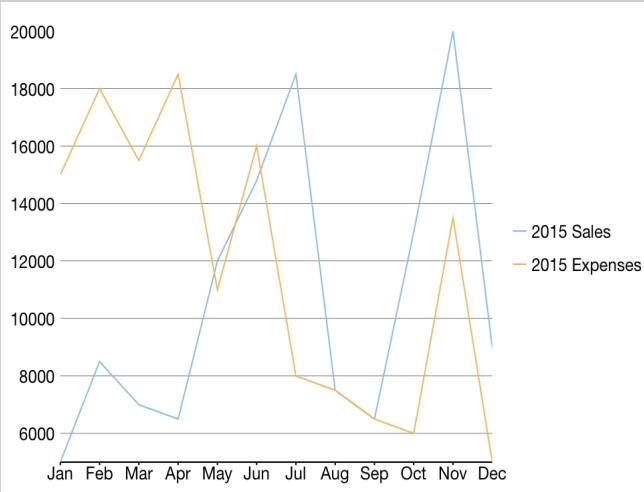
C#

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

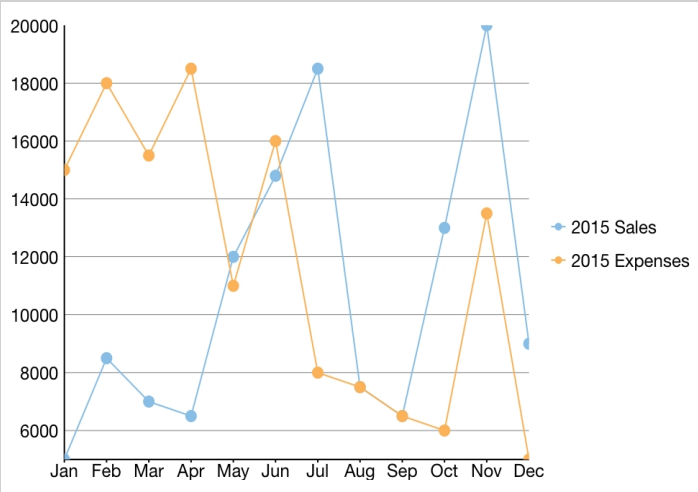
Line and LineSymbol chart

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



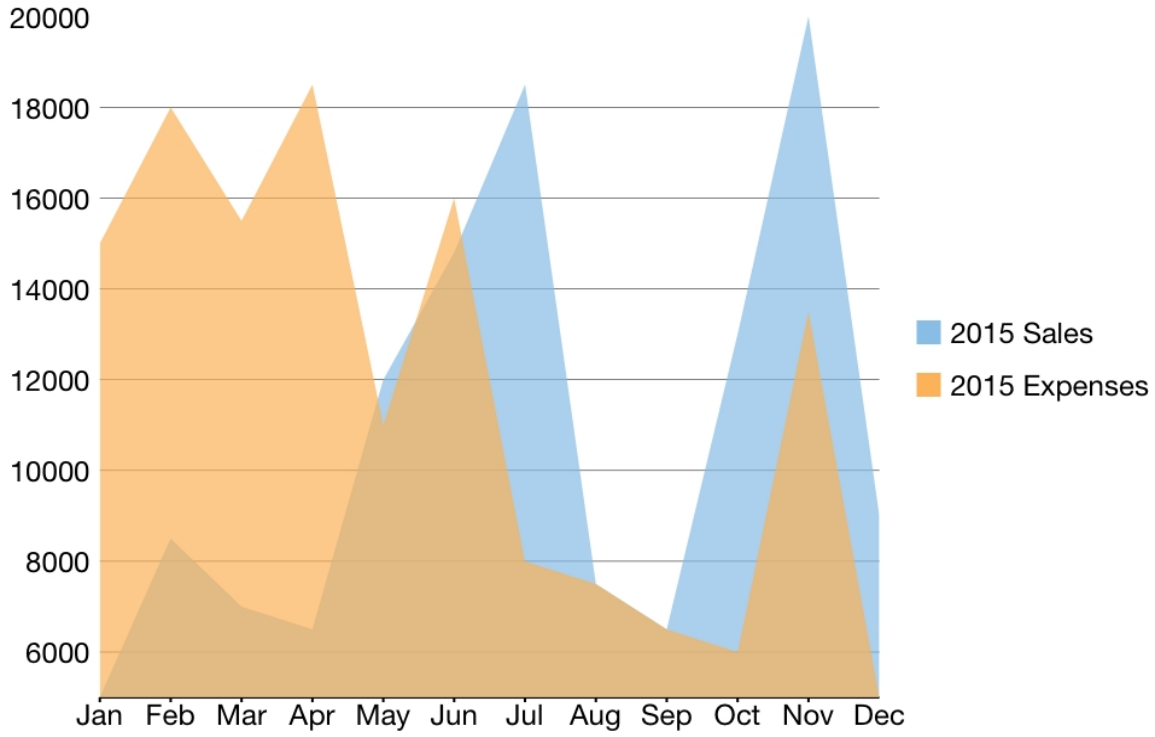
LineSymbol Chart

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.

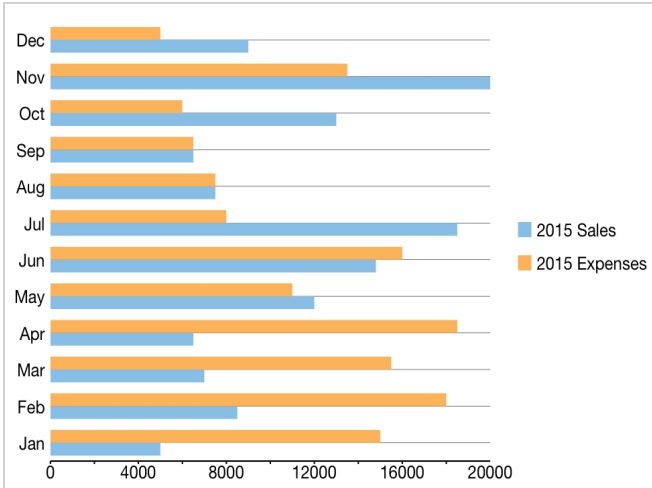
These charts are commonly used to show trends between associated attributes over time.



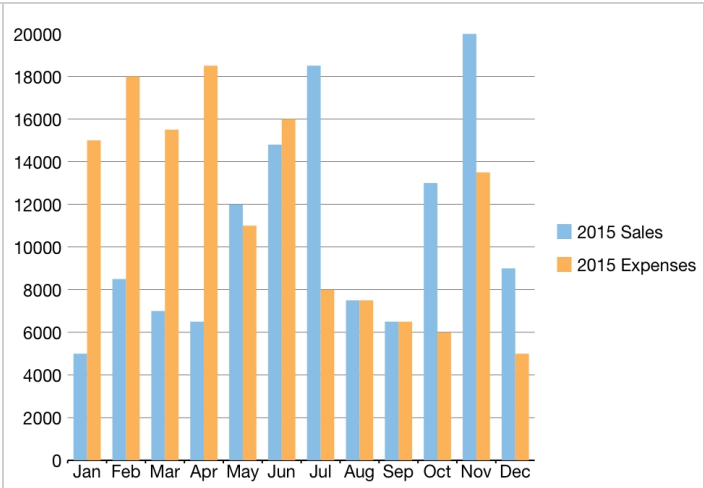
Bar and Column chart

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

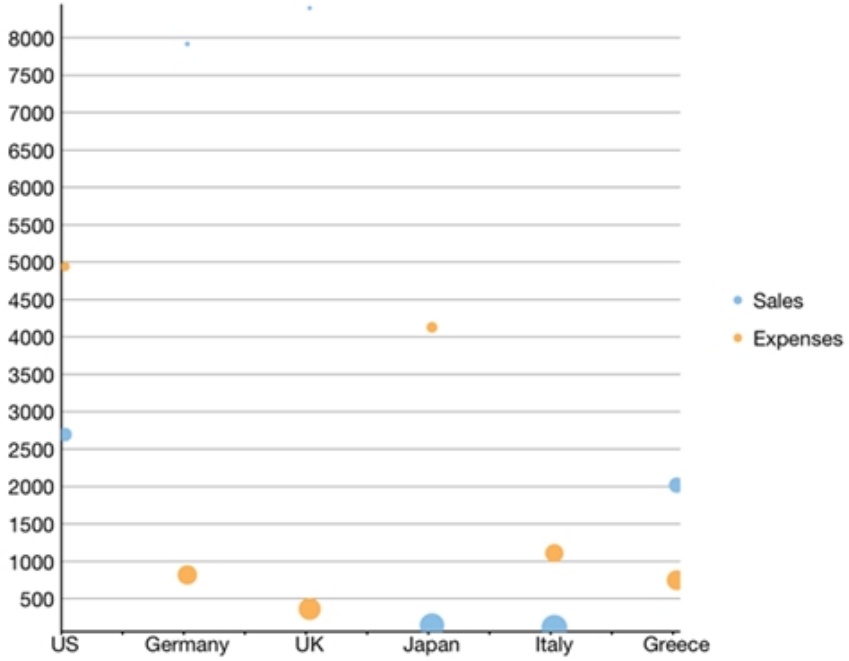


Column Chart

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.

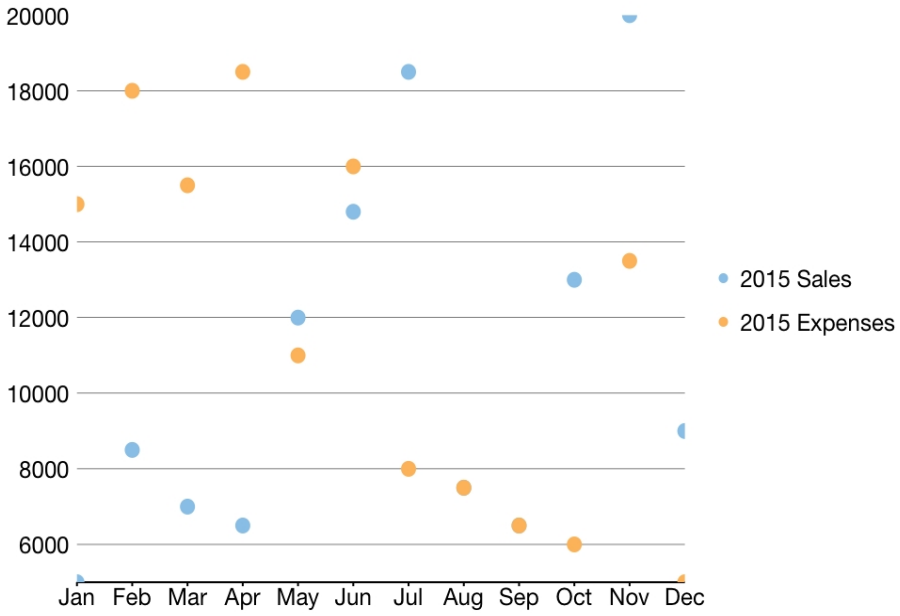
These charts are used to compare entities based on their relative positions on the axis as well as their size.



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. 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.

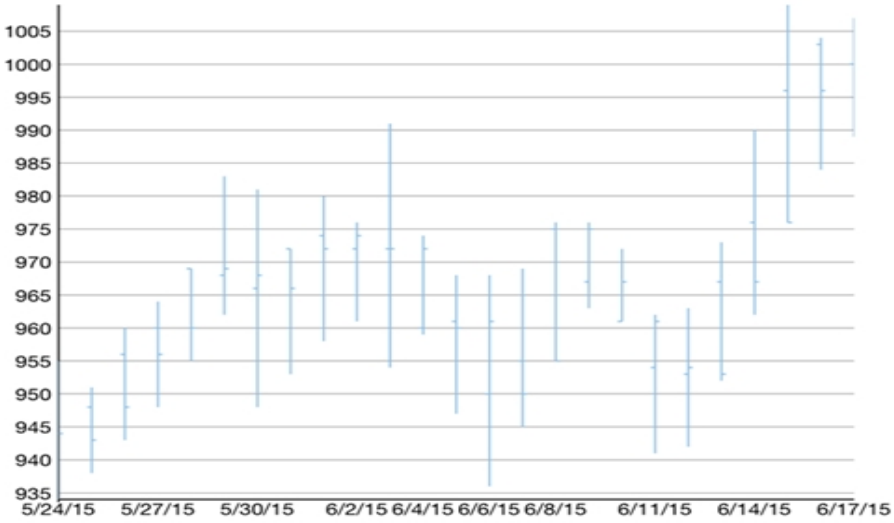
In a Candlestick there are five values for each data point in the series.

- **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

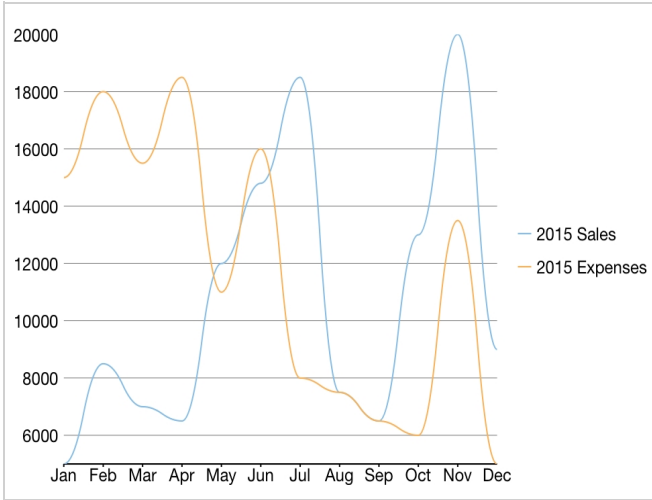
HiLoOpenClose are financial charts that combine four independent values to supply high, low, open and close data for a point in a series. In addition to showing the high and low value of a stock, the Y2 and Y3 array elements represent the stock's opening and closing price respectively.



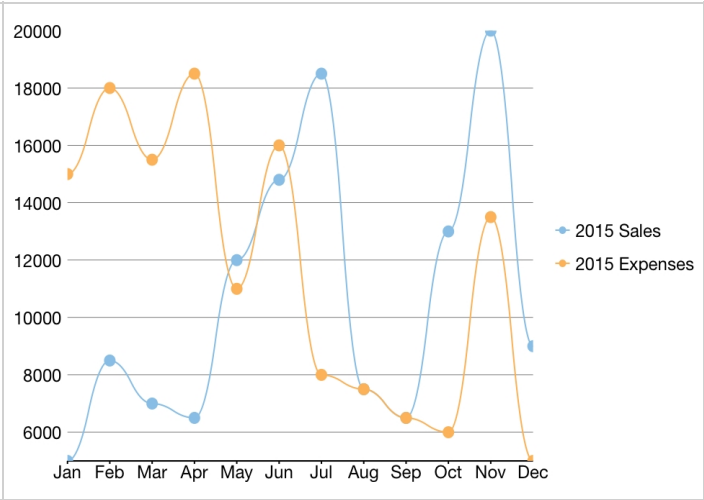
Spline and SplineSymbol chart

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.



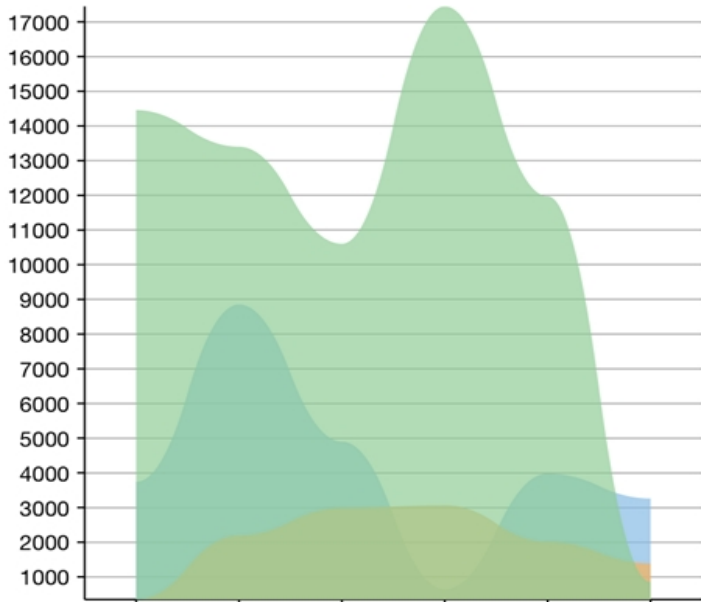
Spline Chart



SplineSymbol Chart

SplineArea chart

SplineArea charts are spline charts that display the area below the spline filled with color. SplineArea chart is similar to Area chart as both the charts show area, except that SplineArea chart uses splines and Area chart uses lines to connect data points.



SplineArea Chart

Quick Start: Add Data to FlexChart

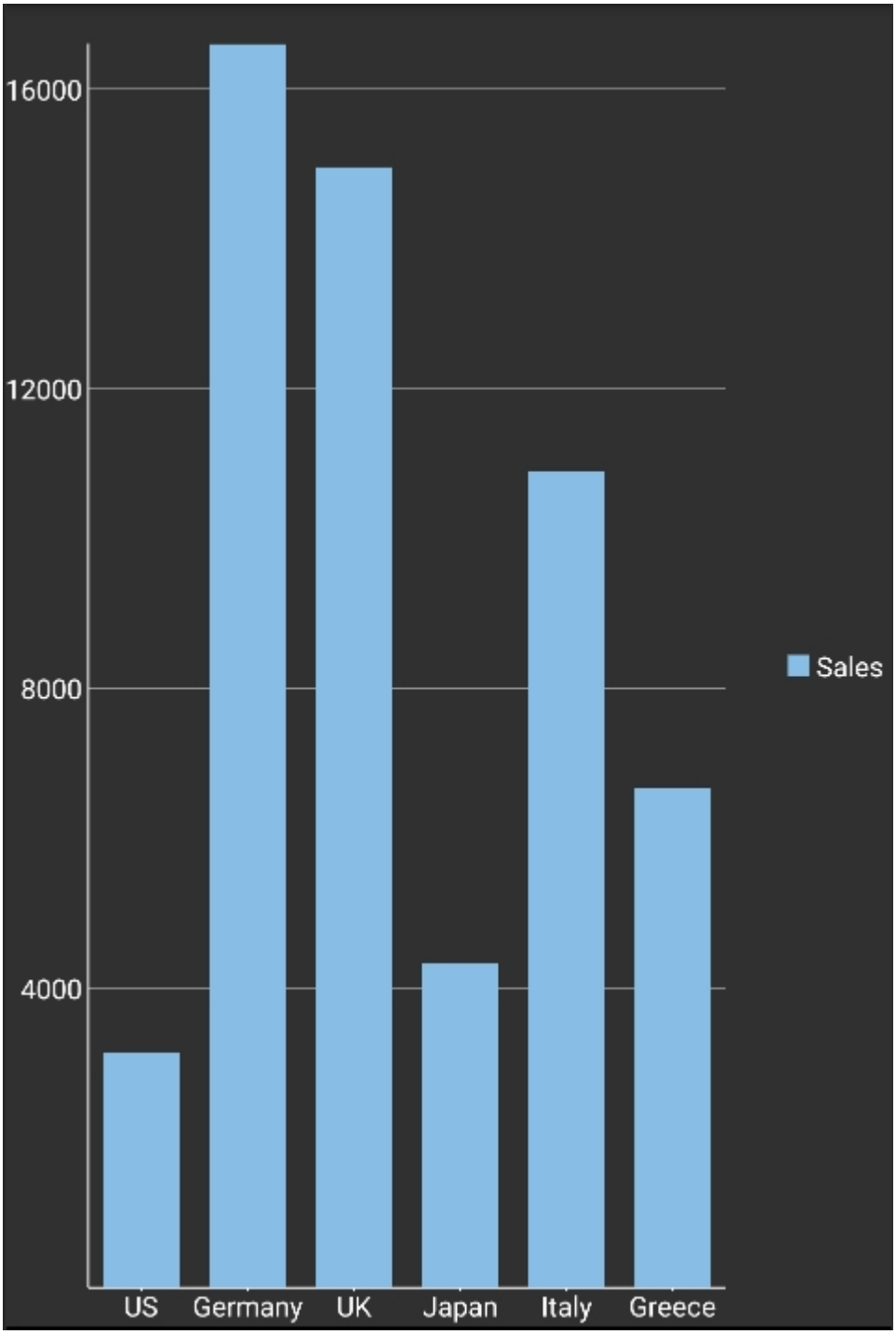
This section describes how to add a FlexChart control to your android application and add data to it.

This topic comprises of three steps:

- **Step 1: Create a data source for FlexChart**

- **Step 2: Add a FlexChart control**
- **Step 3: Run the project**

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 FlexChart control.

C#

```
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

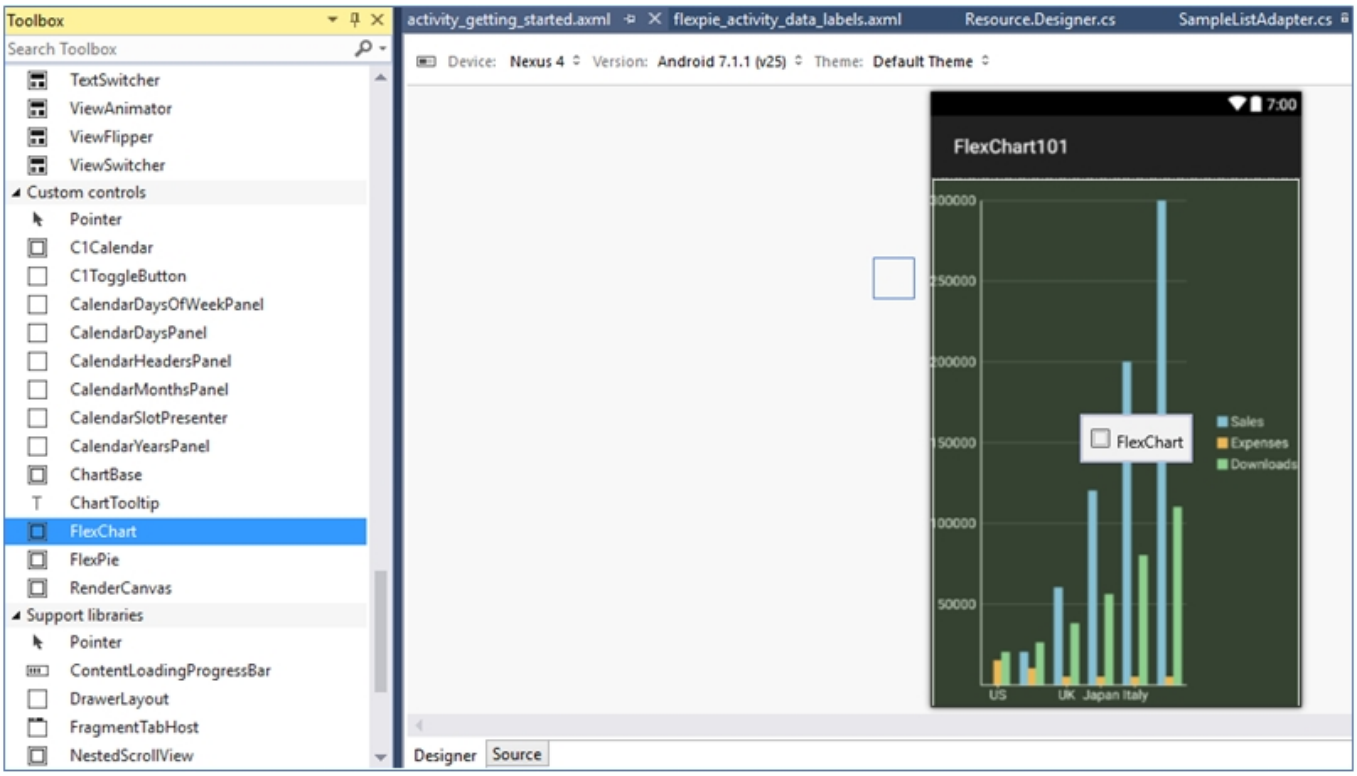
Step 2: Add a FlexChart control in code

To add FlexChart control to your layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with the following code.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <C1.Android.Chart.FlexChart
        android:id="@+id/flexchart"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Alternatively, you can drag a FlexChart control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to the OnCreate method to initialize your layout.

C#

```
public class GettingStartedActivity : Activity
{
    private FlexChart mChart;

    protected override void OnCreate(Bundle savedInstanceState)
    {

        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.activity_getting_started);

        // initializing widget
        mChart = this.FindViewById<FlexChart>(Resource.Id.flexchart);
        // set the binding for X-axis of FlexChart
        mChart.BindingX = "Name";

        // initialize series elements and set the binding to variables of
        // ChartPoint
        ChartSeries seriesSales = new ChartSeries();
        seriesSales.Chart = mChart;
        seriesSales.SeriesName = "Sales";
        seriesSales.Binding = "Sales,Sales";
        mChart.Series.Add(seriesSales);
        mChart.ItemsSource = ChartPoint.GetList();
    }

}
```

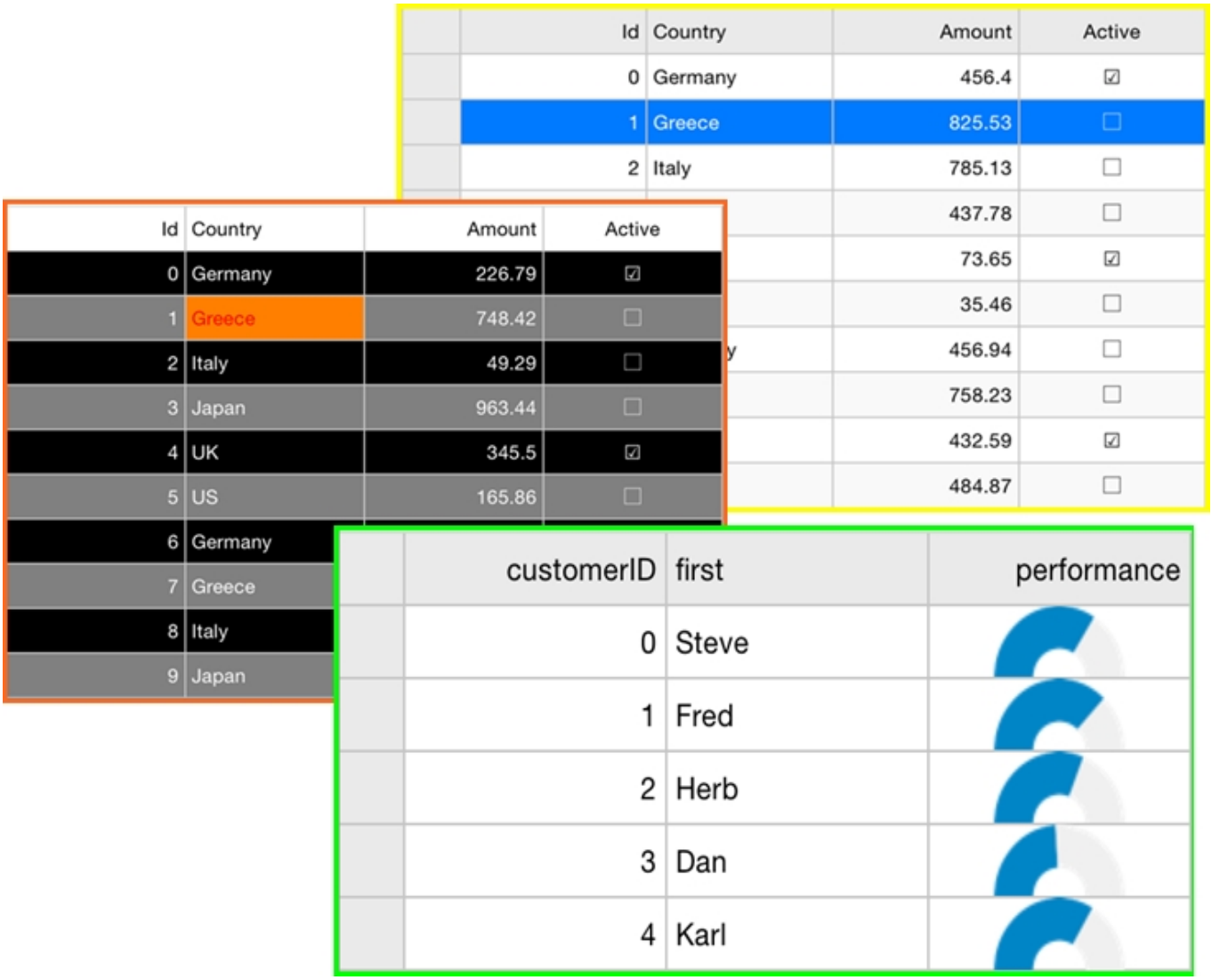
Step 3: Run the project

Press **F5** to run your application.

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 Android 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 via the 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 Android 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 project**

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

	Id	First Name	Last Name	Addres
	0	Dan	Orsted	512 Pa
	1	Jack	Lehman	813 Pa
	2	Ted	Neiman	259 Br
	3	Rich	Quaid	118 Gr
	4	Ted	Richards	995 Gr
	5	Andy	Ambers	434 Ma
	6	Ted	Bishop	511 Ma
	7	Ulrich	Evers	366 Gr
	8	Ted	Cole	991 Go
	9	Larry	Trask	301 Gr
	10	Xavier	Lehman	169 Br
	11	Vic	Krause	508 Gr

Step 1: Create a data source for FlexGrid

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

Customer.cs

```
public class Customer
    {
```

```
        int _id, _countryID;
        string _first, _last;
        bool _active;
        double _weight;
        DateTime _hired;
        static Random _rnd = new Random();
        static string[] _firstNames =
"Andy|Ben|Charlie|Dan|Ed|Fred|Gil|Herb|Jack|Rich|Ted|Ulrich|Vic|Xavier".Split('|');
        static string[] _lastNames =
"Ambers|Bishop|Cole|Evers|Lehman|Neiman|Orsted|Quaid|Richards|Trask".Split('|');
        static string[] _countries = "China|India|United
States|Indonesia|Brazil|Pakistan|Bangladesh|Nigeria|Russia|Japan".Split('|');

        public Customer()
            : this(_rnd.Next(10000))
        {
        }
        public Customer(int id)
        {
            ID = id;
            First = GetString(_firstNames);
            Last = GetString(_lastNames);
            CountryID = _rnd.Next() % _countries.Length;
            Active = _rnd.NextDouble() >= .5;
            Hired = DateTime.Today.AddDays(-_rnd.Next(1, 365));
            Weight = 50 + _rnd.NextDouble() * 50;
        }
        public int ID
        {
            get { return _id; }
            set { _id = value;}
        }
        public string Name
        {
            get { return string.Format("{0} {1}", First, Last); }
        }
        public string Country
        {
            get { return _countries[_countryID]; }
        }
        public int CountryID
        {
            get { return _countryID; }
            set {_countryID = value; }
        }
        public bool Active
        {
            get { return _active; }
            set { _active = value;}
        }
        public string First
```

```
        {
            get { return _first; }
            set {_first = value; }
        }
        public string Last
        {
            get { return _last; }
            set {_last = value; }
        }
        public DateTime Hired
        {
            get { return _hired; }
            set { _hired = value;}
        }
        public double Weight
        {
            get { return _weight; }
            set {_weight = value; }
        }
        static string GetString(string[] arr)
        {
            return arr[_rnd.Next(arr.Length)];
        }
        static string GetName()
        {
            return string.Format("{0} {1}", GetString(_firstNames),
GetString(_lastNames));
        }
        // 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;
        }
        //Provide static value members.
        public static string[] GetCountries() { return _countries; }
        public static string[] GetFirstNames() { return _firstNames; }
        public static string[] GetLastNames() { return _lastNames; }
    }
```

Back to Top

Step 2: Add a FlexGrid control

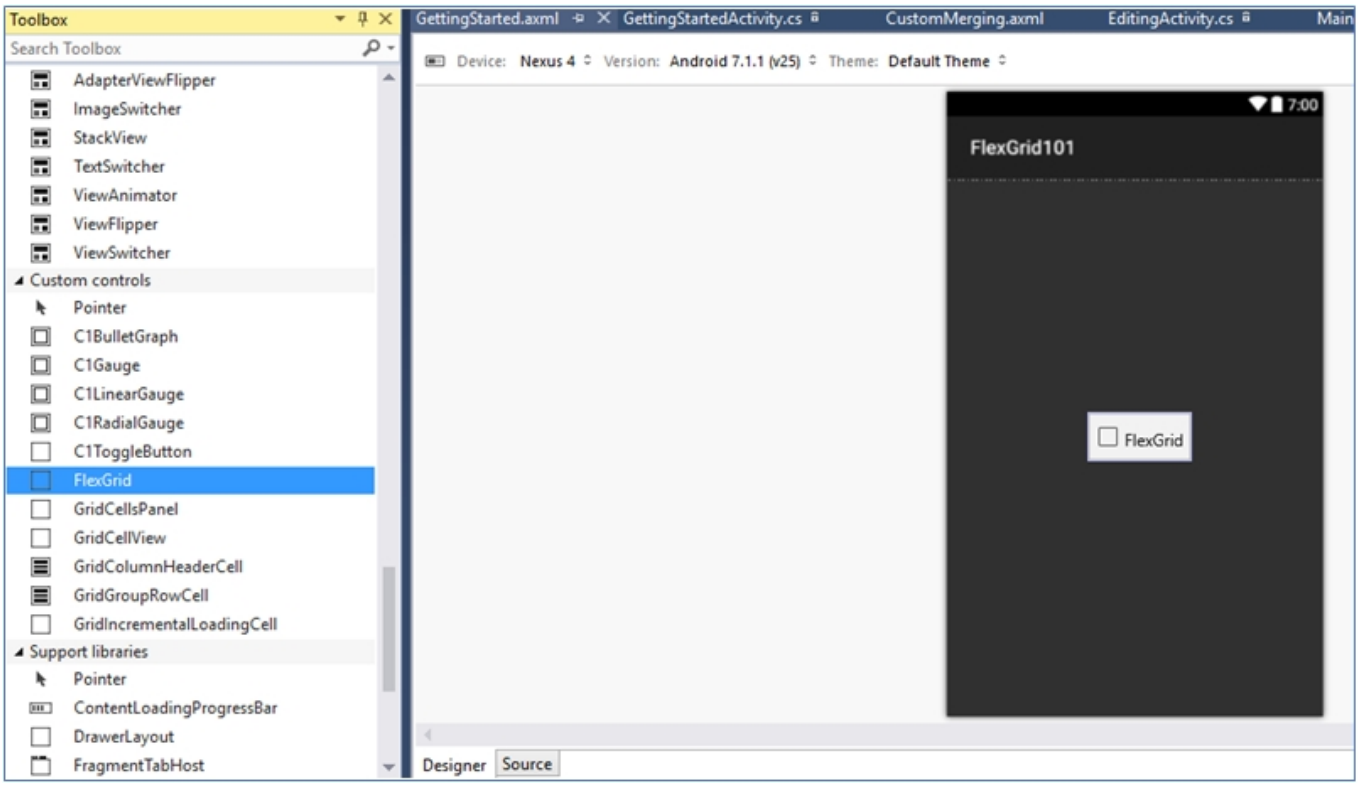
Initialize FlexGrid control in code

To add the FlexGrid control to your layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with the code below.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<C1.Android.Grid.FlexGrid xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/Grid" />
```

Alternatively, you can drag a FlexGrid control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to the OnCreate method to initialize your layout.

XML

```
protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            SetContentView(Resource.Layout.GettingStarted);

            var grid = FindViewById<FlexGrid>(Resource.Id.Grid);
            grid.ItemsSource = Customer.GetCustomerList(100);
        }
```

Back to Top

Step 3: Run the project

Press **F5** to your 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

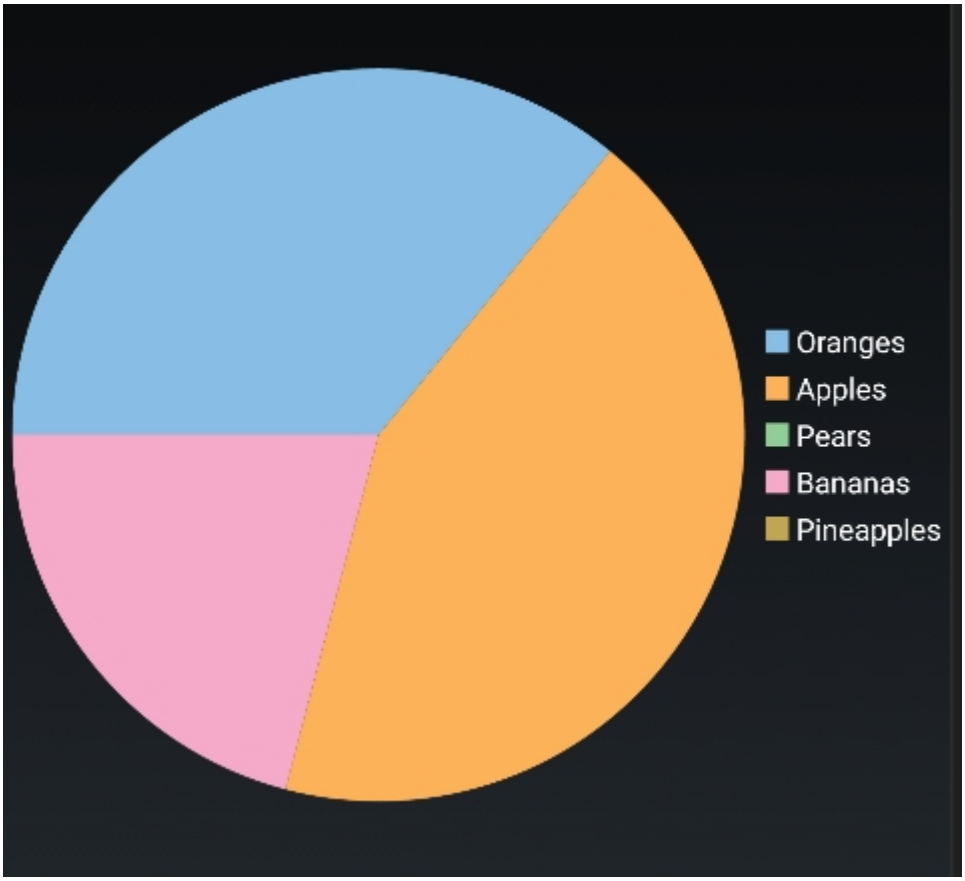
- **Touch Based Labels**: Display values using touch based labels.
- **Exploding and Donut Pie Charts**: Use simple FlexPie properties to convert it into an exploding pie chart or a donut pie chart.

Quick Start: Add Data to FlexPie

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

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

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 control.

PieChartData

```
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;
                            }
           }
```

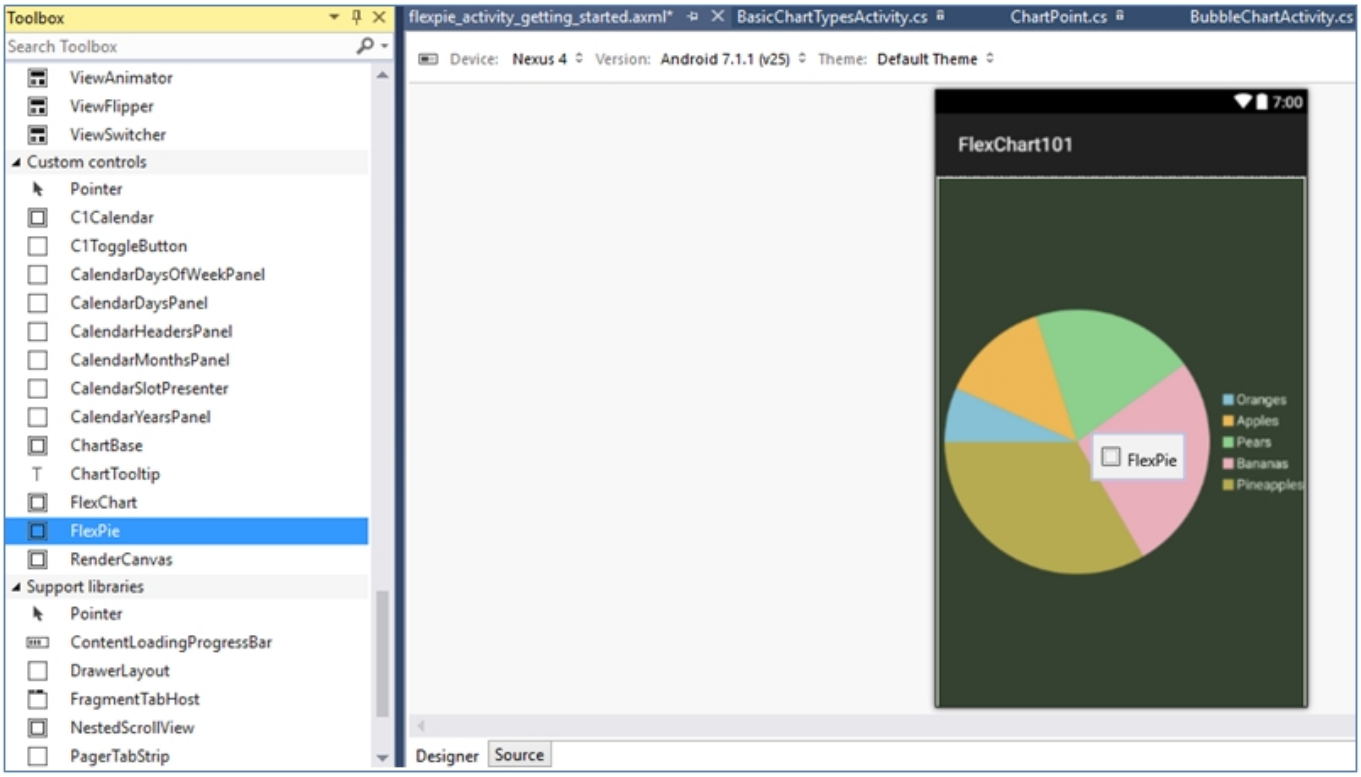
Step 2: Add a FlexPie control

To add a FlexPie control to your layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with the code below.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<C1.Android.Chart.FlexPie xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/flexPie"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"/>
```

Alternatively, you can drag a FlexPie control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to the OnCreate method to initialize your layout.

XML

```
public class GettingStartedActivity : Activity
    {
        private FlexPie mFlexPie;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            // setting the dark theme
            // FlexPie automatically adjusts to the current theme
            SetTheme(Android.Resource.Style.ThemeHolo);

            base.OnCreate(savedInstanceState);
            SetContentView(Resource.Layout.flexpie_activity_getting_started);
```

```
            // initializing widgets
            mFlexPie = (FlexPie)FindViewById(Resource.Id.flexPie);


            mFlexPie.BindingName = "Name";
            mFlexPie.Binding = "Value";

            // setting the source of data/items and default values in FlexPie
            mFlexPie.ItemsSource = PieChartData.DemoData();
        }
    }
```

Back to Top

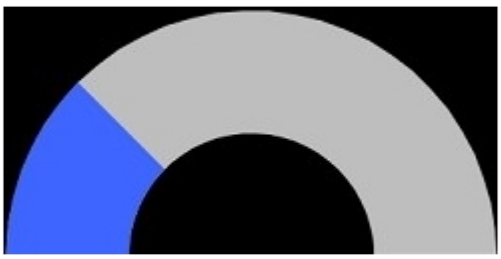
Step 3: Run the Project

Press **F5** to run the application.

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

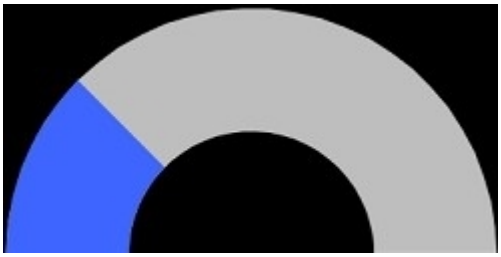
Bullet Graph

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 C1LinearGauge and C1BulletGraph 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: LinearGauge, RadialGauge, and BulletGraph.

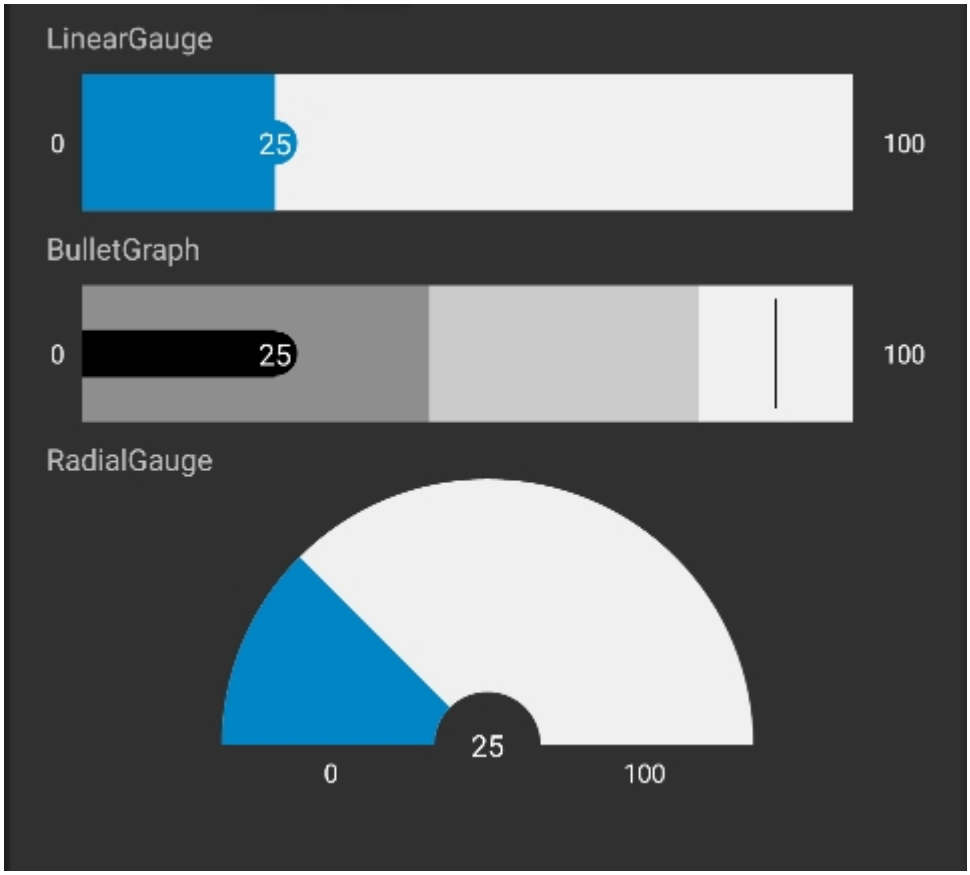
Type	Image	Usage
LinearGauge: A linear gauge displays the value along a linear scale, using a linear pointer. The linear scale can be either horizontal or vertical, which can be set using the **direction** property.		A linear gauge is commonly used to denote data as a scale value such as length, temperature, etc.
RadialGauge: 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.
BulletGraph: 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 control to your Android app. This topic comprises two steps:

- **Step 1: Add Gauge Control**s
- **Step 2: Run the Project**

The following image shows how the C1Gauge appears after completing the steps above:



Step 1: Add Gauge Controls

Initialize Gauge controls in code

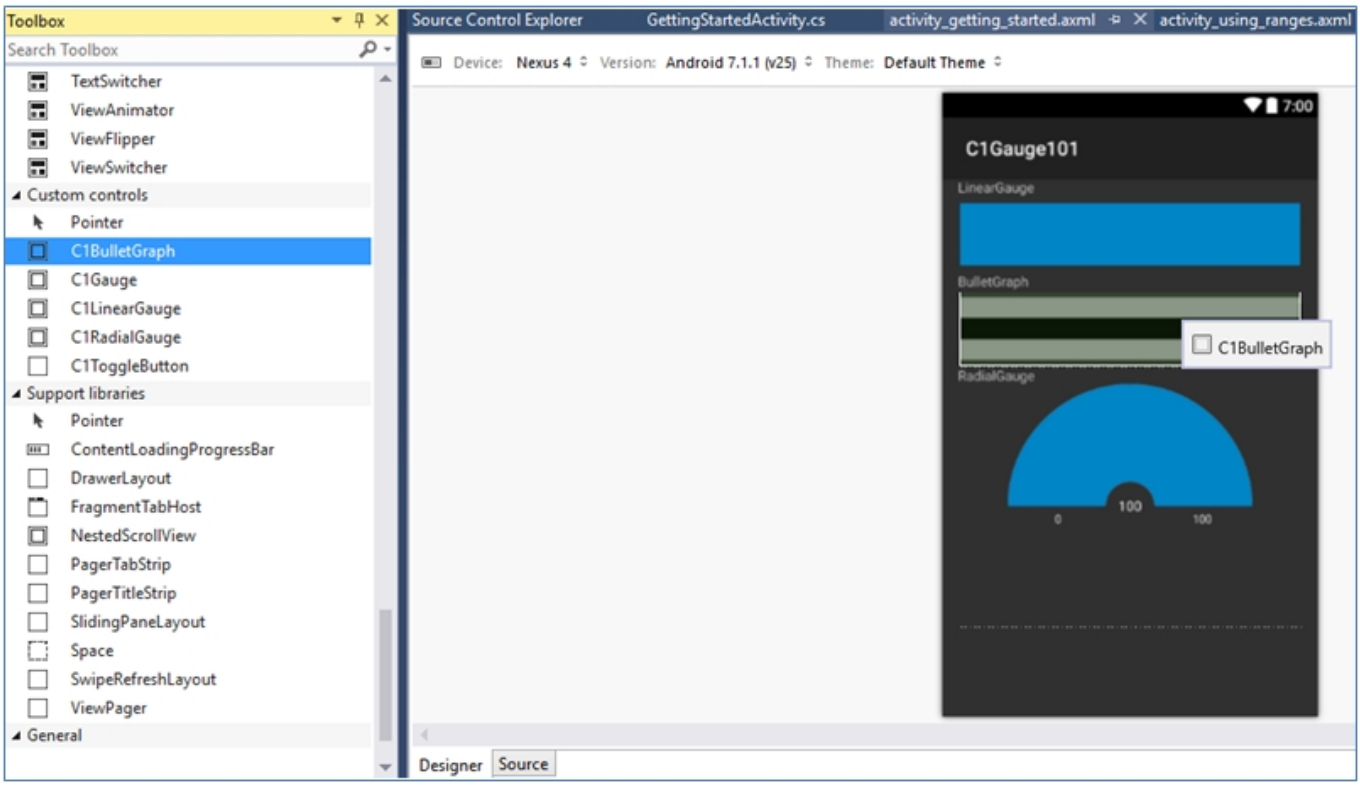
To add C1Gauge controls to your layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with following code.

XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lineargauge" />
    <c1.android.gauge.C1LinearGauge
        android:id="@+id/linearGauge1"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bulletgraph" />
    <c1.android.gauge.C1BulletGraph
        android:id="@+id/bulletGraph1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/radialgauge" />
    <c1.android.gauge.C1RadialGauge
        android:id="@+id/radialGauge1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Alternatively, you can drag the C1LinearGauge, C1RadialGauge, or C1BulletGraph control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to the OnCreate method to initialize your layout.

C#

```
public class GettingStartedActivity : Activity
    {
        private C1LinearGauge mLinearGauge;
        private C1RadialGauge mRadialGauge;
```

```
        private C1BulletGraph mBulletGraph;
        private TextView mValueText;
        private int mValue = 25;
        private int mMin = 0;
        private int mMax = 100;

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView(Resource.Layout.activity_getting_started);

            // initializing widgets
            mRadialGauge = (C1RadialGauge)FindViewById(Resource.Id.radialGauge1);
            mLinearGauge = (C1LinearGauge)FindViewById(Resource.Id.linearGauge1);
            mBulletGraph = (C1BulletGraph)FindViewById(Resource.Id.bulletGraph1);

            // setting dafault values
            mBulletGraph.Enabled = true;
            mBulletGraph.Value = mValue;
            mBulletGraph.Bad = 45;
            mBulletGraph.Good = 80;
            mBulletGraph.Min = mMin;
            mBulletGraph.Max = mMax;
            mBulletGraph.Target = 90;
            mBulletGraph.ShowText = GaugeShowText.All;
            mBulletGraph.Step = 1;
            mBulletGraph.IsReadOnly = false;
            mBulletGraph.IsAnimated = true;

            mLinearGauge.Enabled = true;
            mLinearGauge.Value = mValue;
            mLinearGauge.Min = mMin;
            mLinearGauge.Max = mMax;
            mLinearGauge.Step = 1;
            mLinearGauge.ShowText = GaugeShowText.All;
            mLinearGauge.IsReadOnly = false;
            mLinearGauge.IsAnimated = true;

            mRadialGauge.Enabled = true;
            mRadialGauge.Value = mValue;
            mRadialGauge.Min = mMin;
            mRadialGauge.Max = mMax;
            mRadialGauge.Step = 1;
            mRadialGauge.ShowText = GaugeShowText.All;
            mRadialGauge.IsReadOnly = false;
            mRadialGauge.IsAnimated = true;
        }

    }
```

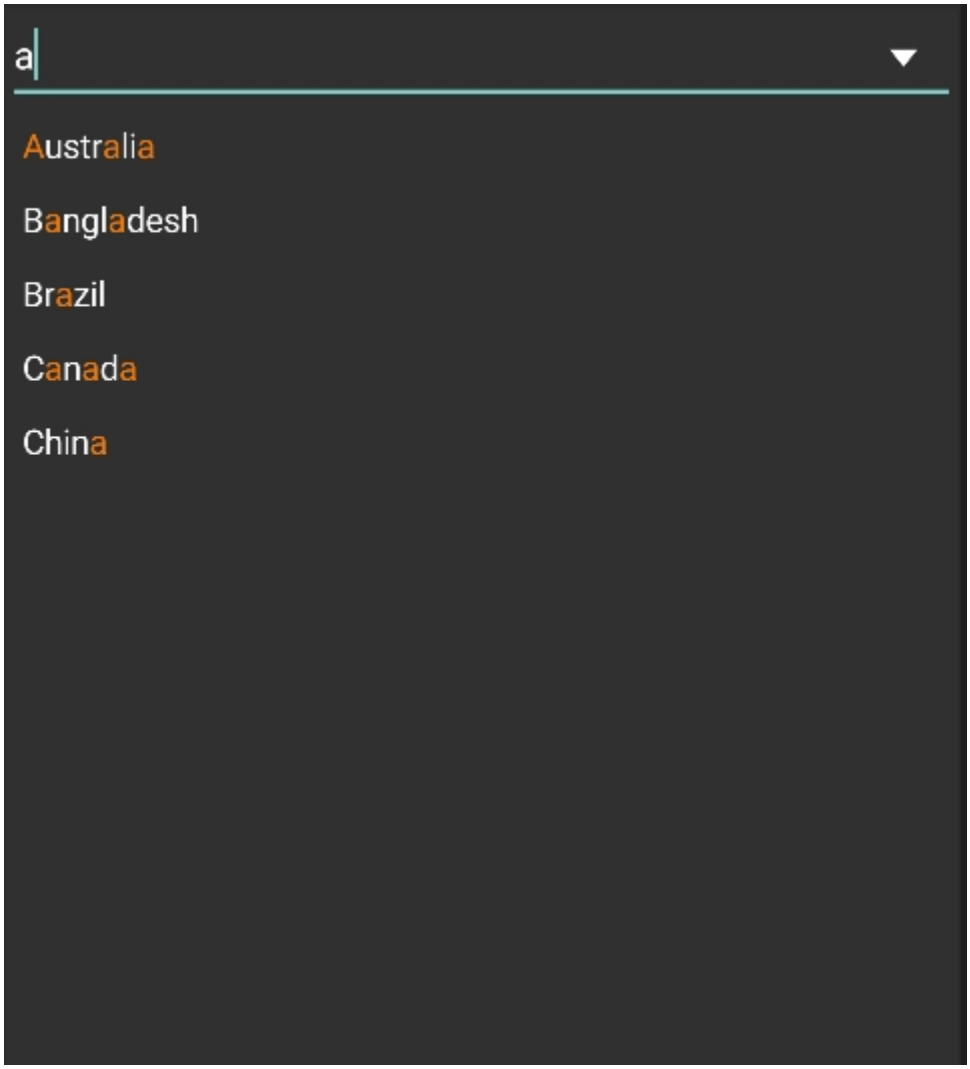
Step 2: Run the Project

Press **F5** to run your application.

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 match the input. The suggestions that match the user input appear instantly in a drop-down list as shown in the image.



Key Features

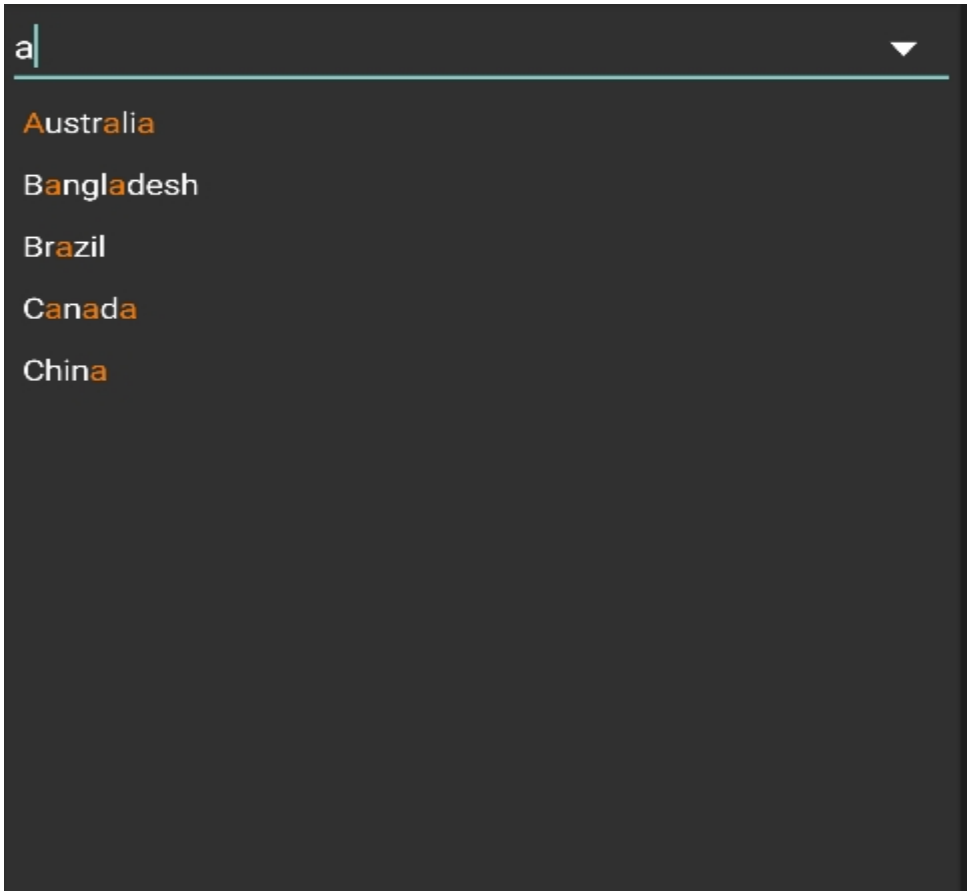
- **Customize Appearance** - Use basic appearance properties to customize the appearance of the drop-down list.
- **Delay** - Use delay feature to provide some time gap (in milliseconds) between user input and suggestion.
- **Highlight Matches** - Highlight the input text with matching string in the suggestions.

Quick Start: Populating C1AutoComplete with data

This section describes how to add a C1AutoComplete control to your Android application, and populating it with data. The data is shown This topic comprises of three steps:

- **Step 1: Add C1AutoComplete Control**
- **Step 3: Run the Project**

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



Step 1: Add C1AutoComplete Control

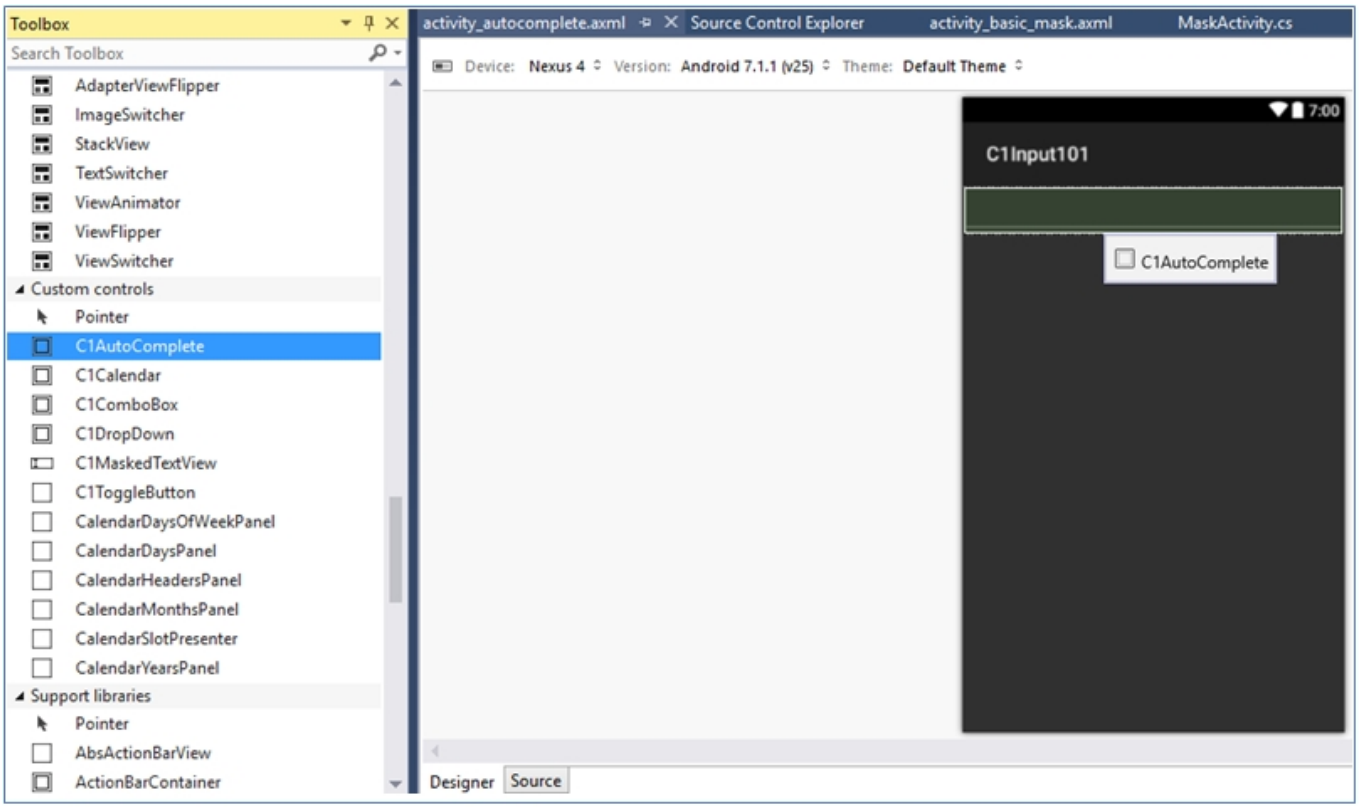
Initialize C1AutoComplete control in code

To add C1AutoComplete control to your layout, open the .axml file in your layout folder from the Solution Explorer and replace the code with following code.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<C1.Android.Input.C1AutoComplete
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_highlight"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Alternatively, you can drag C1AutoComplete control from the Toolbox within the custom control tab onto your layout surface in designer mode.



Then, inside your activity, add the following code to initialize your layout and add data for C1AutoComplete control.

C#

```
public class AutoCompleteActivity : Activity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            SetContentView(Resource.Layout.activity_autocomplete);

            var highLightAutoComplete =
(C1AutoComplete)this.FindViewById(Resource.Id.autocomplete_highlight);
            highLightAutoComplete.ItemsSource = Countries.GetDemoDataList();
            highLightAutoComplete.DisplayMemberPath = "Name";
        }
    }
    public class Countries : object
    {
        public string Name { get; set; }
        public Countries()
        {
            this.Name = string.Empty;
        }
        public Countries(string name, double sales, double salesgoal, double
download, double downloadgoal, double expense, double expensegoal, string fruits)
        {
            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>;
        }
    }
```

Step 2: Run the Project

Press **F5** to run your application.

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, C1ComboBox control is non-editable. However, you can make it editable so that users can modify their input as well.

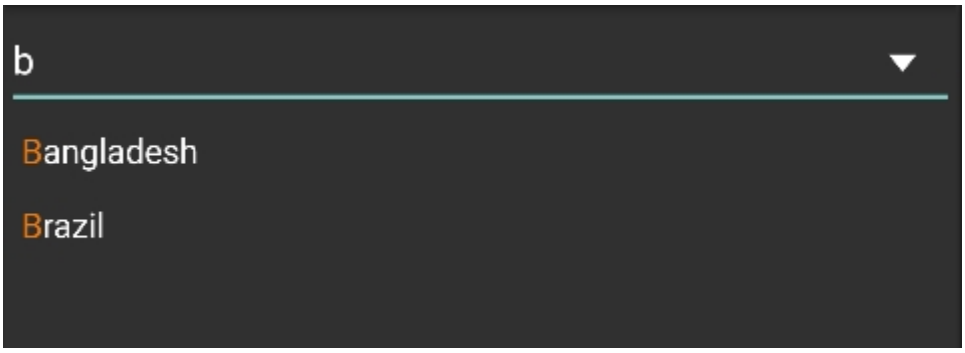
Quick Start: Display a C1ComboBox Control

This section describes adding a C1ComboBox control to your Android application and displaying a list of items in the drop-down as suggestions for users.

Complete the following steps to display a C1ComboBox control.

- **Step 1: Add C1ComboBox Control**
- **Step 2: Run the Project**

The following image shows a C1ComboBox displaying input suggestions as the user types.



Step 1: Add C1ComboBox Control

Initialize C1ComboBox Control

To add the C1ComboBox C1control to you layout, open the .axml file in your layout folder from the Solution Explorer and replace its code with the code below.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<C1.Android.Input.C1AutoComplete
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_highlight"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Alternatively, you can drag a C1ComboBox control from the Toolbox within the custom control tab onto your layout surface in designer mode. Then, inside your activity, add the following code to initialize C1ComboBox

C#

```
public class ComboBoxActivity : Activity
    {
        protected override void OnCreate(Bundle savedInstanceState)
            {
```

```
            base.OnCreate(savedInstanceState);
            LinearLayout layout = new LinearLayout(this);
            layout.Orientation = Orientation.Vertical;


            C1ComboBox comboBox = new C1ComboBox(this);
            comboBox.DisplayMemberPath = "Name";
            comboBox.ItemsSource = Countries.GetDemoDataList();
            comboBox.SelectedBackgroundColor = Color.Green;
            layout.AddView(comboBox);
            comboBox.SelectedValue = new System.Object { };
            Space emptySpace = new Space(this);
            layout.AddView(emptySpace);

            this.SetContentView(layout);
        }
    }
```

Step 2: Run the Project

Press **F5** to run your 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 drop-down 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 Android 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 the ViewController file to display the control.

C#

```
public class DropDownActivity : Activity
   {
        C1DropDown dropdown;
        C1MaskedTextView header;
        C1Calendar calendar;

     protected override void OnCreate(Bundle savedInstanceState)
      {
        base.OnCreate(savedInstanceState);
        dropdown = new C1DropDown(this);
        header = new C1MaskedTextView(this);
        header.Mask = Resources.GetString(Resource.String.date_mask_string);

        calendar = new C1Calendar(this);
        dropdown.Header = header;
        dropdown.DropDown = calendar;
        dropdown.DropDownHeight = 800;
        dropdown.IsAnimated = true;

        calendar.SelectionChanged += (object sender,
CalendarSelectionChangedEventArgs e) =>
```

```
                        {
         dropdown.IsDropDownOpen = false;
         System.DateTime  dateTime = calendar.SelectedDates[0];
         string strDate =
dateTime.ToString(Resources.GetString(Resource.String.date_mask_format));
         header.Value = strDate;
        };

        LinearLayout layout = new LinearLayout(this);
        LinearLayout.LayoutParams parameters = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MatchParent,
LinearLayout.LayoutParams.WrapContent);
        layout.AddView(dropdown, parameters);
        SetContentView(layout);
         }
       }
```

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 input field. For this, all you need to do is use the mask property and specify the data format.

For example, setting the mask property for a C1MaskedTextField control to "90/90/0000" lets users enter date in international 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 Android 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 after completing the above steps.

Add the following code to initialize four input fields using C1MaskedTextField controls in you .axml file.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:columnCount="2"
    android:paddingBottom="20dp"
    android:paddingLeft="20dp"
    android:paddingRight="20dp"
```

```
    android:paddingTop="20dp"
    android:rowCount="4">
    <C1.Android.Input.C1MaskedTextView
        android:id="@+id/idMask"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        app:c1_mask="999 99-0000"
        app:c1_promptChar="_" />
    <C1.Android.Input.C1MaskedTextView
        android:id="@+id/dateMask"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        app:c1_mask="90/90/0000"
        app:c1_promptChar="_" />
    <C1.Android.Input.C1MaskedTextView
        android:id="@+id/phoneMask"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        app:c1_mask="(999)000-0000"
        app:c1_promptChar="_" />
    <C1.Android.Input.C1MaskedTextView
        android:id="@+id/stateMask"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        app:c1_mask="LL"
        app:c1_promptChar="_" />
</GridLayout>
```