ComponentOne

Studio for Windows Phone Basic Library

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor

Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: http://www.componentone.com

Sales

E-mail: sales@componentone.com

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

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Studio for Windows Phone Basic Library Overview

The future of Windows Phone development tools is here! **ComponentOne Studio for Windows Phone** features industrial strength Silverlight-based controls you cannot find anywhere else**. Studio for Windows Phone Basic Library** includes input controls and panels in the **C1.Phone.dll** assembly that are all optimized for touch interaction in the Windows Phone environment.

For a list of the latest features added to **ComponentOne Studio for Windows Phone**, visit What's New in Studio for Windows Phone.

Help with ComponentOne Studio for Windows Phone

Getting Started

For information on installing **ComponentOne Studio for Windows Phone**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for Windows Phone.

What's New

For a list of the latest features added to **ComponentOne Studio for Windows Phone**, visit What's New in Studio for Windows Phone.

ContextMenu

Add touch-sensitive menus to your apps with **ComponentOne™ ContextMenu for Windows Phone**. Provide additional commands while saving UI space with this simple, yet customizable menu control.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

ContextMenu for Windows Phone Key Features

ComponentOne ContextMenu for Windows Phone allows you to create customized, rich applications. Make the most of **ContextMenu for Windows Phone** by taking advantage of the following key features:

- **Data Bindable**

  Bind your menus to a data source. Just set the **ItemsSource** property to bind C1ContextMenu to any collection that implements **IEnumerable**.

- **Zoom Background**

  Give emphasis to the menu by zooming out the background. The **IsZoomEnabled** property indicates whether the background will zoom out when the C1ContextMenu is open. You can also style the background by just setting one brush property.

- **Separate Items**

  Add separators to aid in the visual grouping of items.

- **Commanding Support**

  C1ContextMenu items support commands and command parameters. Program against them as you would against any other buttons throughout your application.

- **Customizable Appearance**

  Customize the look of each menu item easily through XAML.

ContextMenu for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne ContextMenu for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add the **C1ContextMenu** control to your application, and customize the appearance and behavior of the control.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **ContextMenu for Windows Phone**. Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.

3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
8. Navigate to the Toolbox and double-click the **C1ContextMenu** icon to add the control to the grid. Create an opening and closing tag. The XAML markup should resemble the following:

```
<Grid x:Name="LayoutRoot">
      <c1:C1ContextMenu>
      </c1:C1ContextMenu>
 </Grid>
```

You have successfully created a Windows Phone application containing a **C1ContextMenu** control. In the next step, you will customize the control.

Step 2 of 4: Customizing the Control

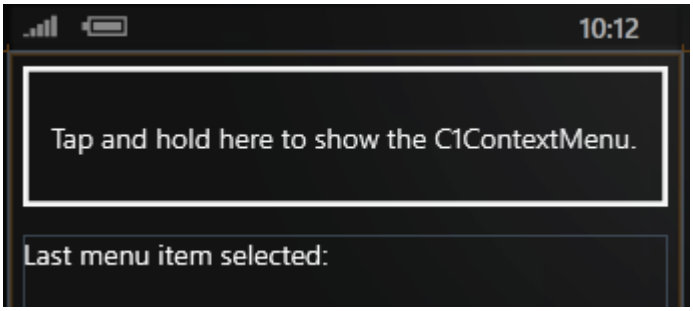
In the previous step you set up the application's user interface and added a C1ContextMenu control to your application. In this step you will add some **C1MenuItems** to the control.

1. First let's create a **TextBlock** with a border that can be tapped to access the context menu. We'll also add some markup to create a second **TextBlock** that will tell the user the last context menu item that was tapped. Add the following XAML to the `<Grid x:Name="LayoutRoot">`, like this:

```
<Grid x:Name="LayoutRoot">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Border Margin="10" Height="100"
               BorderBrush="WhiteSmoke" BorderThickness="4"
VerticalAlignment="Center" Padding="16">

        <TextBlock  Foreground="WhiteSmoke" Text="Tap and hold here to
show the C1ContextMenu." VerticalAlignment="Center"/>
        </Border>
<c1:C1ContextMenu>
```

```
</c1:C1ContextMenu>

        <StackPanel Orientation="Horizontal" Grid.Row="1" Margin="10">
            <TextBlock Text="Last menu item selected: "/>
            <TextBlock x:Name="MenuFeedback" Grid.Row="1"/>
        </StackPanel>
    </Grid>
```

The page should look similar to this:



2. After the closing `</Border>` tag, let's add opening and closing tags for the C1ContextMenuService. The C1ContextMenuService class exposes context menus as extender properties that can be attached to any **FrameworkElement** objects on the page. The C1ContextMenuService tags should contain the C1ContextMenu tags so that the XAML markup looks like this:

```
<c1:C1ContextMenuService.ContextMenu>
   <c1:C1ContextMenu>

   </c1:C1ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

3. Place your cursor in between the `<c1:C1ContextMenu></c1:C1ContextMenu>` tags and add three **C1MenuItems** so the markup resembles the following:

```
<c1:C1ContextMenuService.ContextMenu>
<c1:C1ContextMenu>
  <c1:C1MenuItem x:Name="FirstMenuItem" Header="first menu item"/>
  <c1:C1MenuItem x:Name="SecondMenuItem" Header="second menu item"/>
  <c1:C1Separator/>
  <c1:C1MenuItem x:Name="ThirdMenuItem" Header="third menu item"/>
</c1:C1ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

In this step, you added menu items to the C1ContextMenu control. In the next step, you will run the project and experience the functionality of the control.

Step 3 of 4: Adding Code

In the previous step, you customized the C1ContextMenu and add three menu items. In this step, we're going to add some code that returns the last item tapped so any user of the phone will be able to see the last menu item that was selected.

1. In the Visual Studio menu, select **View | Code**.
2. Add the following code:

```
private void C1ContextMenu_ItemTap(object sender, SourcedEventArgs e)
        {
```

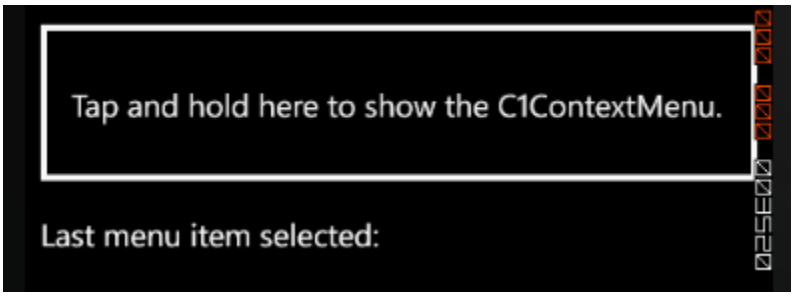
```
            MenuFeedback.Text = ((C1MenuItem)e.Source).Header + " tapped
!!";
        }
```

When the project runs and a menu item is tapped in the C1ContextMenu, the **TextBlock** named ***MenuFeedback*** will state the last menu item that was tapped.

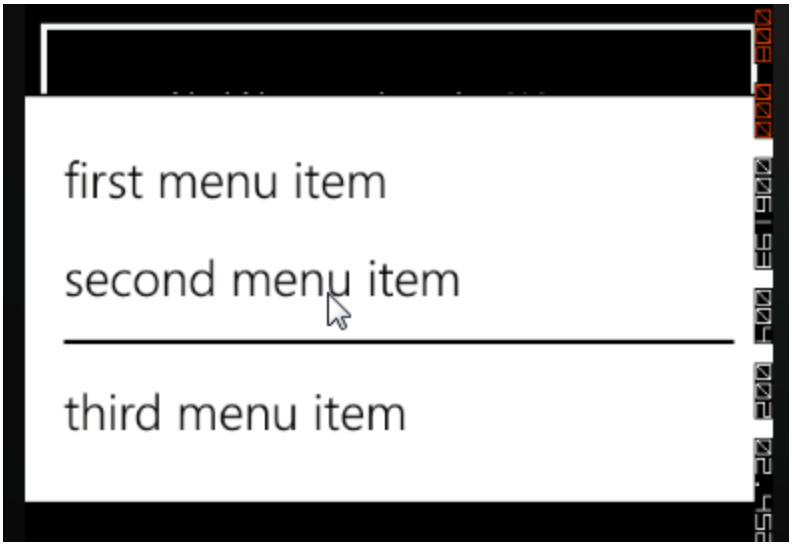
Step 4 of 4: Running the Application

Now that you've created a Windows Phone application and created a context menu with menu items, the only thing left to do is run your application. To run your application and observe **ContextMenu for Windows Phone**'s run-time behavior, complete the following steps:

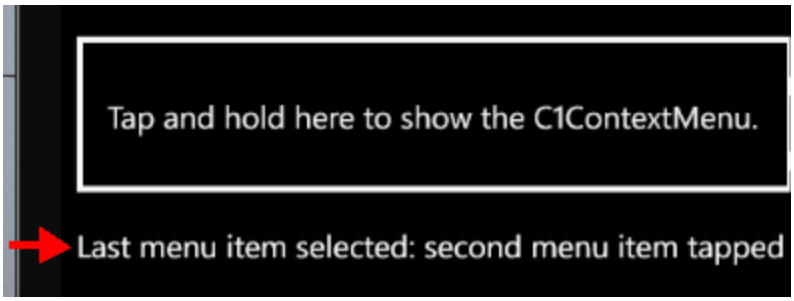
1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:



2. Tap and hold in the **TextBox** with the white border. The C1ContextMenu appears and the menu items are listed.



3. Tap one of the menu items. Notice that the menu item selected is now listed in the second **TextBox** where it says "Last menu item selected:".



Congratulations! You've completed the **ContextMenu for Windows Phone** quick start.

Working with the C1ContextMenu Control

The following topics introduce you to all of the elements and several features of the C1ContextMenu control.

C1ContextMenu Elements

The C1ContextMenu provides a pop-up menu that provides frequently used commands that are associated with the selected object. The C1ContextMenuService class exposes context menus as extender properties that can be attached to any **FrameworkElement** objects on the page. Once the menu is attached to an object, the users can right-click that object to open the menu.

DateTimePicker

Provide touch-driven date and time editing with **ComponentOne DateTimePicker™ for Windows Phone**. The C1DateTimePicker control provides a single, intuitive UI for selecting date and time values. Just set the desired format and you are ready to go!

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

DateTimePicker for Windows Phone Key Features

ComponentOne DateTimeEditors for Windows Phone allows you to create customized, rich applications. Make the most of **DateTimeEditors for Windows Phone** by taking advantage of the following key features:

- **Three Edit Modes**

  Set the **EditMode** property to specify whether you need to edit Date, Time or Date and Time values.

- **Date and Time Formats**

  Just set the **DateFormat** and **TimeFormat** properties to specify the desired format. C1DateTimePicker supports most standard .NET **Date and Time** format strings. For instance, set **DateFormat** to "d" for a short date pattern.

- **Metro UI Design**

  C1DateTimePicker supports the Metro UI design and interaction guidelines specified by Microsoft. It is similar to the toolkit **DateTimePicker** control in features and functionality.

- **Wide Range of Cultures**

  C1DateTimePicker supports a wide range of cultures specified at the application level.

DateTimePicker for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne DateTimeEditors for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add **C1DateTimePicker** control to your application, and customize the appearance and behavior of the control.

Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **DateTimePicker for Windows Phone**. Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.

5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
8. Navigate to the Toolbox and double-click the **C1DateTimePicker** icon to add the control to the grid. The XAML markup resembles the following:

```

<Grid x:Name="LayoutRoot">
<c1datetime:C1DateTimePicker></c1datetime:C1DateTimePicker>
</Grid>
```

You have successfully created a Windows Phone application containing a **C1DateTimePicker** control. In the next step, you will customize the control.

Step 2 of 3: Customizing the Control

In the previous step you set up the application's user interface and added a C1DateTimePicker control to your application. In this step you will modify the appearance of the control.

Complete the following steps:

1. Add `Height="30"` to the `<c1datetime:C1DateTimePicker>` tag to determine height of the control. The XAML markup appears as follows:

```
<c1:C1DateTimePicker Height="30">
```

2. Add `Width="300"` to the `<c1datetime:C1DateTimePicker>` tag to determine the width of the control. The XAML markup appears as follows:

```
<c1:C1DateTimePicker Height="50" Width="300">
```

3. Add `TimeFormat="ShortTime"` to the `<c1datetime:C1DateTimePicker>` tag to change the format of the time to a short format consisting of only hours and minute spaces. The XAML markup appears as follows:

```
<c1:C1DateTimePicker Height="50" Width="300" TimeFormat="ShortTime">
```

4. Add `DateFormat="Long"` to the `<c1datetime:C1DateTimePicker>` tag to change the format of the date to a longer format that includes the weekday. The XAML markup appears as follows:

```
<c1:C1DateTimePicker Height="50" Width="300" TimeFormat="ShortTime"
DateFormat="Long" Name="c1DateTimePicker1" >
```

5. Add `EditMode="DateTime"` to the `<c1datetime:C1DateTimePicker>` tag so that both the date and time appear in the C1DateTimePicker. The XAML markup appears as follows:

```
<c1:C1DateTimePicker Height="50" Width="300" TimeFormat="ShortTime"
DateFormat="Long" Name="c1DateTimePicker1" EditMode="DateTime">
```

6. You can add a **TextBlock** to the **C1DateTimePicker.Header** to put a label over the C1DateTimePicker:

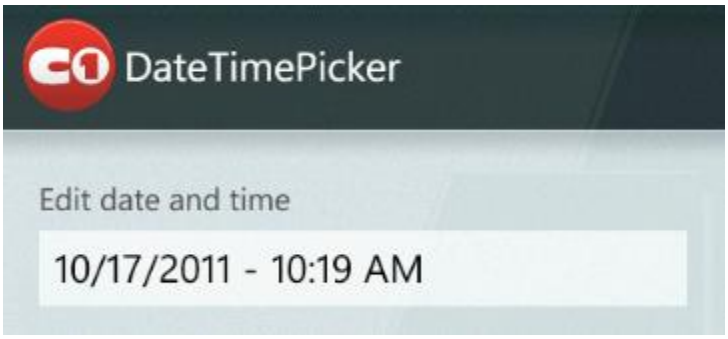
```
<c1:C1DateTimePicker.Header>
      <TextBlock Text="Edit date and time" Style="{StaticResource
PageSimpleTextBlock}"/>
</c1:C1DateTimePicker.Header>
```

In this step, you customized the appearance of the C1DateTimePicker control. In the next step, you will run the project and experience the functionality of the control.

Step 3 of 3: Running the Application

Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **DateTimePicker for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:



2. Tap the date and time text box. The **Choose Date** window opens.

3. Tap a month, date, and year. Then tap the checkmark.
4. Tap the date and time text box again. In the **Choose Date** window, tap the time to change it. Tap the checkmark when you are finished.

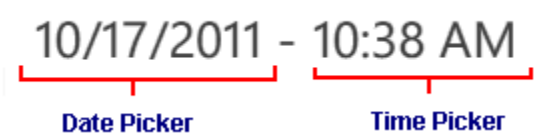
Congratulations! You've completed the **DateTimePicker for Windows Phone** quick start and created a **DateTimePicker for Windows Phone** application, customized the appearance and behavior of the control, and viewed some of the run-time capabilities of your application.

Working with the C1DateTimePicker Control

The following topics introduce you to all of the elements and several features of the C1DateTimePicker control.

C1DateTimePicker Elements

ComponentOne DateTimePicker for Windows Phone includes the C1DateTimePicker control, a simple control that provides, by default, both a date picker and a time picker. When you add the C1DateTimePicker control to a XAML window, it exists as a completely functional date and time picker. By default, the control's interface looks similar to the following image:



The C1DateTimePicker control consists of a date picker and time picker. When the pickers are tapped, you can choose a new date and/or time.

Edit Modes

By default, the C1DateTimePicker control will appear on the page with both the date picker and the time picker. You can change the pickers that are displayed by setting the EditMode property to **Date**, **Time**, or **DateTime**. You can set the EditMode property to **Date** to display only the date picker; you can set the EditMode property to **Time** to only display the time picker; and you can set the EditMode property to **DateTime** (default) to display both the time picker and date picker. The table below illustrates each editor mode.

Editor Mode	Result
Date	10/17/2011
Time	10:47 AM
DateTime (default)	10/17/2011 - 10:47 AM

Date and Time Format

You can use the DateFormat property to set the format that the date picker displays. You can use the TimeFormat property to set the format that the time displays. The Microsoft .NET standard date and time format strings can be used to determine how you want the C1DateTimePicker to look. The following table provides examples for some of the available formats.

Format	Description	Result
d	Short date pattern. (Default)	10/17/2011
D	Long date pattern	Monday, November 07, 2011
h:mm tt	Displays the hours, minutes, and AM/PM. (Default)	10:47 AM
T	Long	12:50:27 PM

DateTimePicker for Windows Phone Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1DateTimePicker control in general. If you are unfamiliar with the **ComponentOne DateTimePicker for Windows Phone** product, please see the **DateTimePicker for Windows Phone** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne DateTimePicker for Windows Phone** product.

Each task-based help topic also assumes that you have created a new Windows Phone project.

Allowing Null Values

By default, the C1DateTimePicker control doesn't allow users to enter null values, but you can force the control to accept a null value by setting the AllowNull property to **True**. In this topic, you will learn how to set the AllowNull property to **True** in XAML, from the Properties Window, and in code.

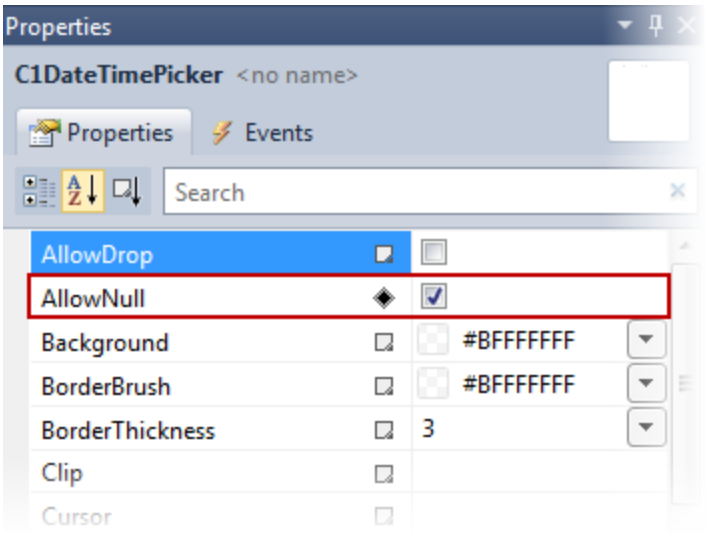
In XAML

To allow Null values, place `AllowNull="True"` to the `<c1: C1DateTimePicker>` tag so that the markup resembles the following:

```
<c1:C1DateTimePicker AllowNull="True"> </c1:C1DateTimePicker>
```

From the Properties Window

1. From the Visual Studio toolbar, select **View | Properties**.
2. Locate the AllowNull property and check the check box to allow null values as in the following image:



In Code

1. Add `x:Name="C1DateTimePicker1"` to the `<c1:C1DateTimePicker>` tag so that the control will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page by right-clicking on your project and selecting **View Code** from the menu.
3. Insert the following using statement at the top of the page.
   - Visual Basic
     ```
     Imports C1.Phone
     ```
   - C#
     ```
     using C1.Phone;
     ```
4. Add the following code after the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1DateTimePicker1.AllowNull = true
     ```
   - C#
     ```
     C1DateTimePicker1.AllowNull = true;
     ```

Selecting the Edit Mode

By default, the C1DateTimePicker control shows both the date and time pickers, but you may also choose to show only the date picker or only the time picker. In this topic, you will learn how to change the editor mode in XAML, from the Properties window, and in code.

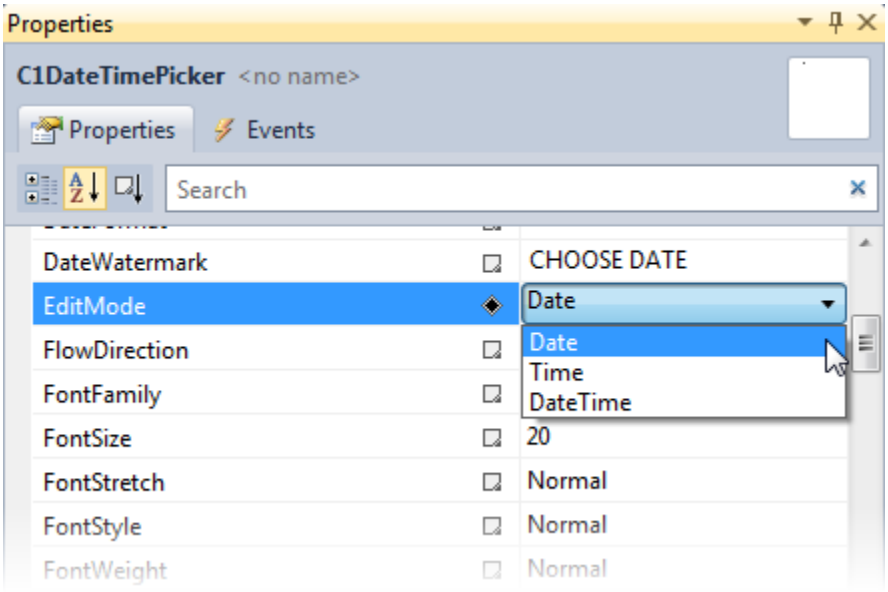
In XAML

To change the EditMode, place EditMode="Date" in the <c1:C1DateTimePicker> tag so the markup resembles the following:

```
<c1:C1DateTimePicker x:Name="C1DateTimePicker1" EditMode="Date">
```

From the Properties Window

Locate the EditMode property in your application's Properties window. Use the drop-down list to select "**Date**" as the EditMode, as in the following image:



In Code

1. Add `x:Name="C1DateTimePicker1"` to the `<c1:C1DateTimePicker>` tag so that the control will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page by right-clicking on your project and selecting **View Code** from the menu.
3. Insert the following using statement at the top of the page.
   - Visual Basic
     ```
     Imports C1.Phone
     ```
   - C#
     ```
     using C1.Phone;
     ```
4. Place the following code after the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1DateTimePicker1.EditMode = C1DateTimePickerEditMode.Date
     ```
   - C#

```
C1DateTimePicker1.EditMode = C1DateTimePickerEditMode.Date;
```

5. Press F5 to run the application.

This Topic Illustrates

Changing the Edit Mode of the C1DateTimePicker

Selecting the Time Format

By default, the C1DateTimePicker control's time picker displays the time in a long format that includes seconds, but it can also display time in a shorter format. In this topic, you will learn how to change the time format in XAML, from the Properties window, and in code.

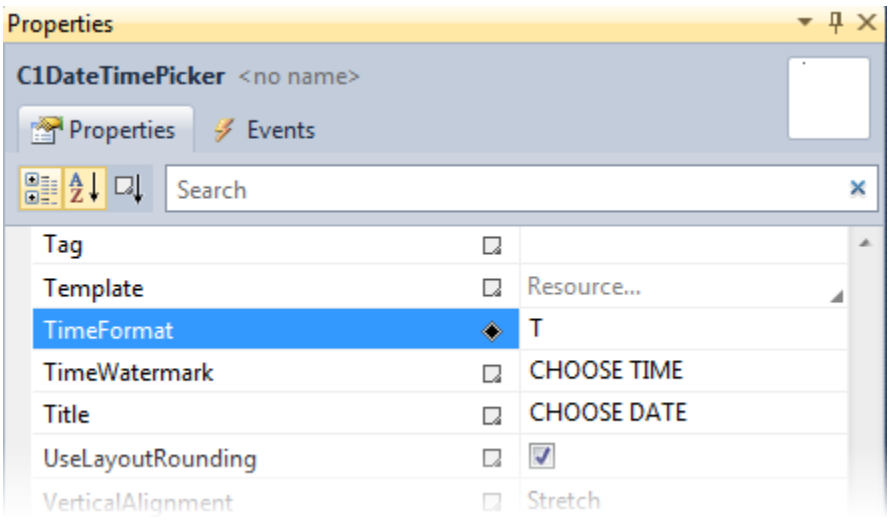
In XAML

To change the Date format, place `TimeFormat="T"` to the `<c1:C1DateTimePicker>` tag to resemble the following markup:

```
<c1:C1DateTimePicker TimeFormat="T">
```

From the Properties Window

Locate the TimeFormat property in your application's Properties window. Change the character in the TimeFormat property setting textbox to **T**. The time is now set to display in a long format, as in the following image:



This Topic Illustrates

Changing the **TimeFormat** property.

Selecting the Date Format

By default, the C1DateTimePicker control's date picker displays the date in a short format, but it can also display the date in a long format. In this topic, you will learn how to change the date format in XAML and from the Properties window.

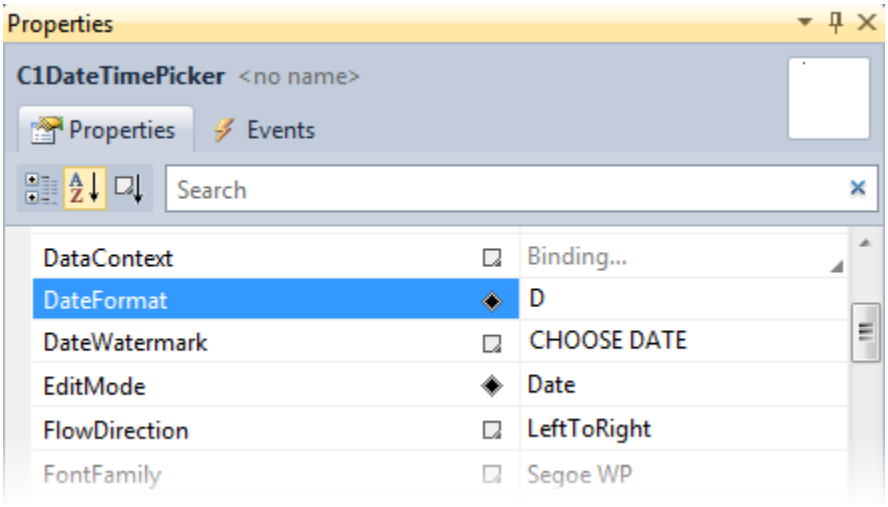
In XAML

To change the Date format, place `DateFormat="D"` to the `<c1:C1DateTimePicker>` tag to resemble the following markup:

```
<c1:C1DateTimePicker DateFormat="D">
```

From the Properties Window

Locate the DateFormat property in your application's Properties window. Change the character in the DateFormat property setting textbox to **D**. The date is now set to display in a long format as in the following image:



This Topic Illustrates

Changing the **DateFormat** property.

Selecting Minimum and Maximum Calendar Dates

You can change the dates that calendar spans by setting the **MinimumDate** and **MaximumDate** properties in XAML or from the Properites window.

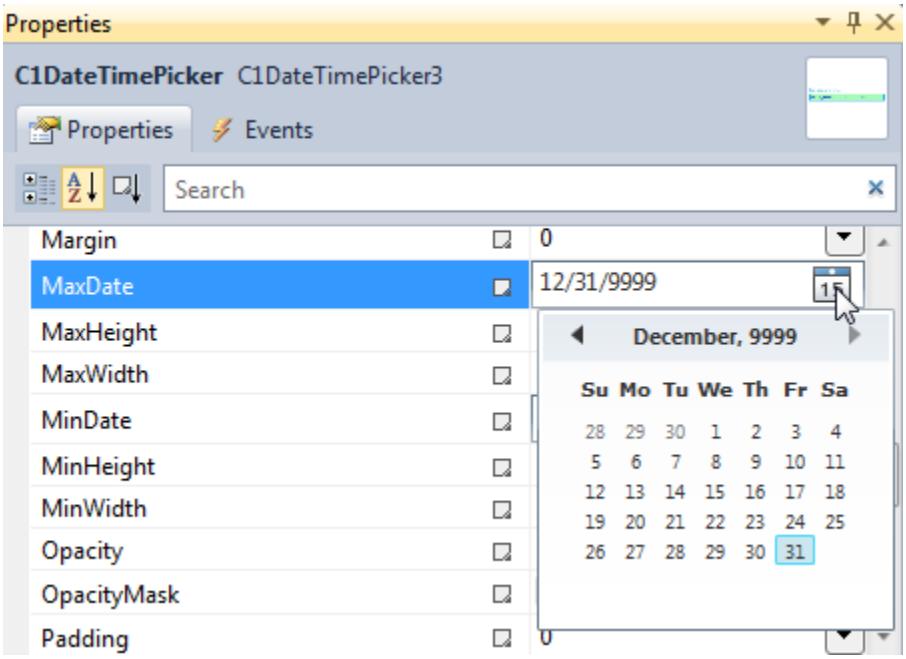
In XAML

Add `MaxDate="12/22/9999"` and `MinDate="1/15/0001"` to the `<c1:C1DateTimePicker>` tag so that the markup resembles the following:

```
<c1:C1DateTimePicker MaxDate="12/22/9999" MinDate="01/25/0001">
```

From the Properties window

Locate the MaxDate and MinDate properties in your application's Properties window. Either enter a date in the property settings field, or use the drop-down calendar to select a new minimum and maximum date.



Specifying a Date

You can specify the displayed date of a C1DateTimePicker control by setting the C1DateTimePicker.SelectedDate property in XAML, from the Properties window, or in code.

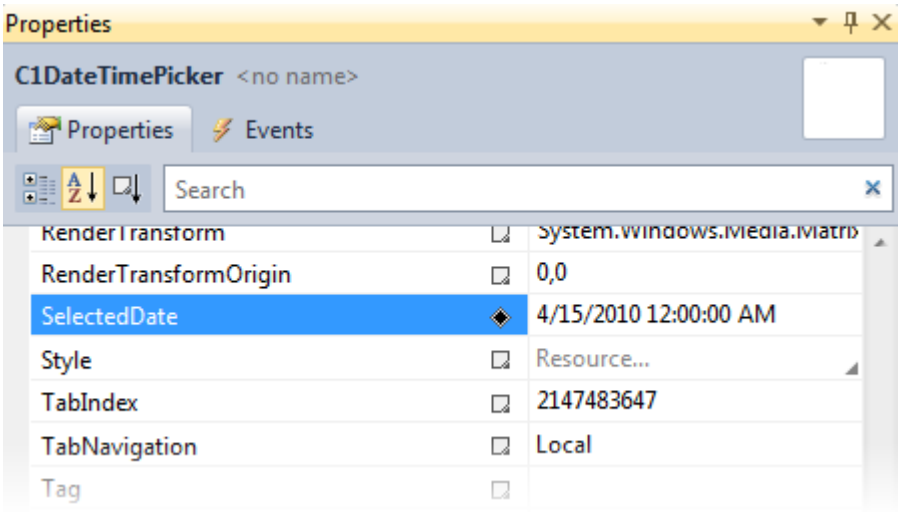
In XAML

To change the date displayed at run-time, add `SelectedDate="4/15/2010"` to the `<c1:C1DateTimePicker>` tag. The markup will resemble the following:

```
<c1:C1DateTimePicker SelectedDate="04/15/2010">
```

From the Properties Window

Locate the SelectedDate property in the application's Properties window. Enter a new date in the property setting textbox as in the following image:



In Code

1. Add `x:Name="C1DateTimePicker1"` to the `<c1:C1DateTimePicker>` tag so that the control will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page by right-clicking on your project and selecting **View Code** from the menu.
3. Insert the following using statement at the top of the page.
   - Visual Basic
     ```
     Imports C1.Phone
     ```
   - C#
     ```
     using C1.Phone;
     ```
4. Place the following code after the **InitializeComponent()** method:
   - Visual Studio
     ```
     C1DateTimePicker1.SelectedDate = New DateTime(2010, 4, 15)
     ```
   - C#
     ```
     C1DateTimePicker1.SelectedDate = new DateTime(2010, 4, 15);
     ```
5. Run your application. The date you chose will be displayed in the Date Picker as in the following image:

Changing the C1DateTimePicker's Appearance

You can change the appearance of a C1DateTimePicker control in XAML or from the Properties window. For this help, we will use an application that contains three instances of the C1DateTimePicker control: one set to `EditMode="Date"`, one set to `EditMode="Time"`, and one set to `EditMode="DateTime"`. The starting XAML markup for the entire application should resemble the following:

```
<!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" >
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

 <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="ComponentOne
DateTimePicker for Windows Phone" Style="{StaticResource
PhoneTextNormalStyle}" FontSize="18" />
            <TextBlock x:Name="PageTitle" Text="Date and Time"
Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"
FontSize="72" />
        </StackPanel>

 <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,30" >
            <StackPanel Margin="20 40" >
                <c1:C1DateTimePicker x:Name="C1DateTimePicker1"
EditMode="Date" UseLayoutRounding="False" TimeFormat="T"
DateFormat="D">
                    <c1:C1DateTimePicker.Header>
                        <TextBlock Text="Edit date" />
                    </c1:C1DateTimePicker.Header>
```

```
                </c1:C1DateTimePicker>
                <c1:C1DateTimePicker x:Name="C1DateTimePicker2"
EditMode="Time" TimeFormat="T">
                    <c1:C1DateTimePicker.Header>
                        <TextBlock Text="Edit time"/>
                    </c1:C1DateTimePicker.Header>
                </c1:C1DateTimePicker>
                <c1:C1DateTimePicker x:Name="C1DateTimePicker3"
AllowDrop="False" DateFormat="D" TimeFormat="T">
                    <c1:C1DateTimePicker.Header>
                        <TextBlock Text="Edit date and time" />
                    </c1:C1DateTimePicker.Header>
                </c1:C1DateTimePicker>
            </StackPanel>        </Grid>
    </Grid>
```

In XAML

Follow these steps:

1. In your application's MainPage.xaml page, locate the first set of `<c1:C1DateTimePicker>` tags. Insert `BorderThickness="5" Background="#FFA6FCB6" BorderBrush="#FF29ACEB" Foreground="#FF0C3EC4"` in the tag to customize the appearance of the control. The markup should resemble the following:

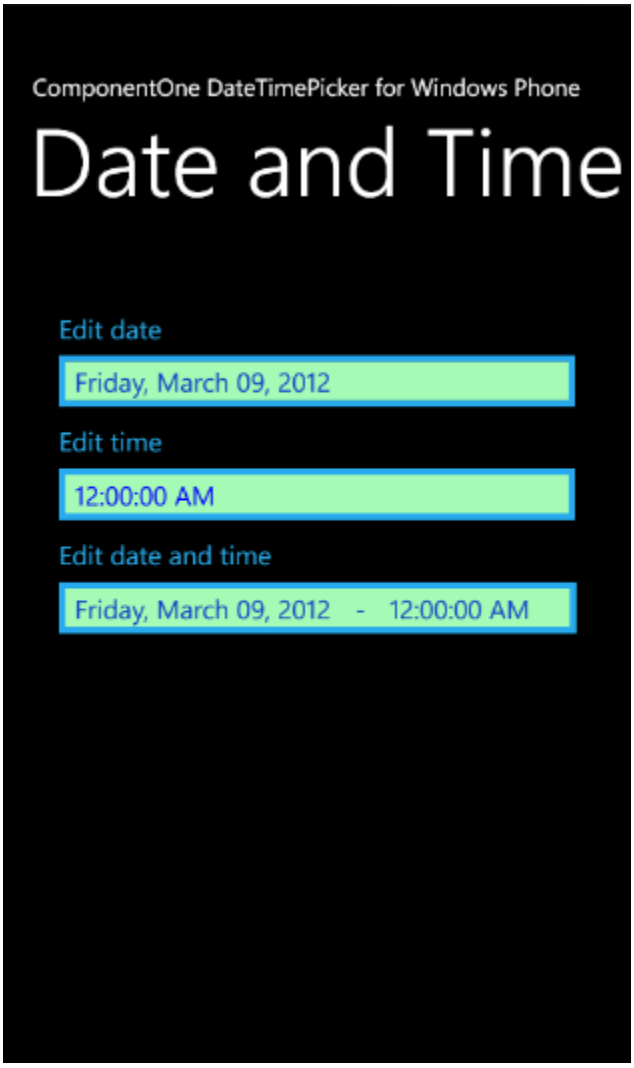
   ```
   <c1:C1DateTimePicker EditMode="Date" UseLayoutRounding="False"
   DateFormat="D" BorderThickness="5" Background="#FFA6FCB6"
   BorderBrush="#FF29ACEB" Foreground="#FF0C3EC4">
   ```

2. Locate the `<c1:C1DateTimePicker.Header>` tags and insert `Foreground="FF29AECB"` into the tag so that the markup resembles the following:

   ```
   <c1:C1DateTimePicker.Header>
    <TextBlock Text="Edit date" Foreground="#FF29ACEB" />
    </c1:C1DateTimePicker.Header>
   ```

3. Repeat the first two steps with the next two sets of `<c1:C1DateTimePicker>` and `<c1:C1DateTimePicker.Header>` tags so that all three controls will share the same color scheme.

4. Run your application. The application should resemble the following image:

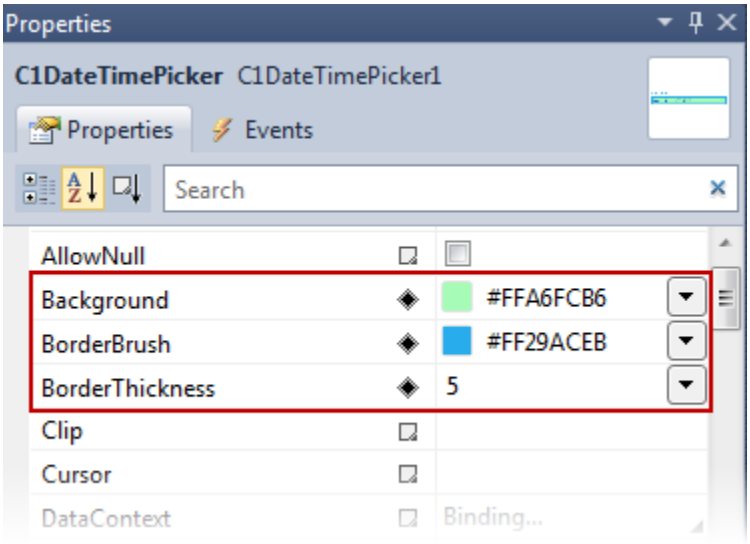


From the Properties Window

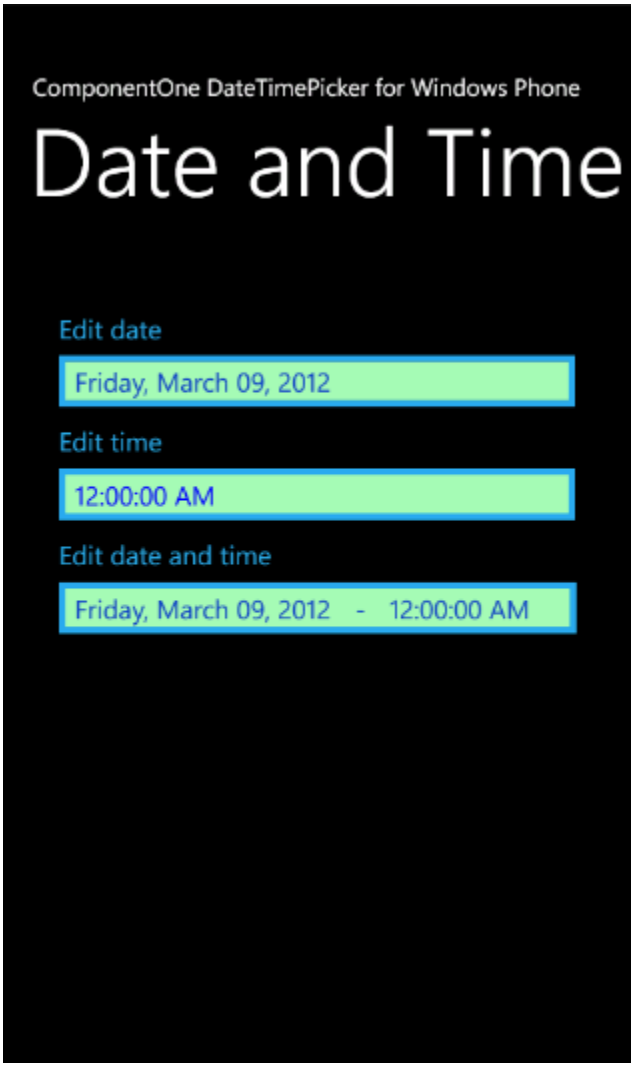
Follow these steps:

1. Click on the first **C1DateTimePicker** control in your application to view the Properties for that control. The Properties window should display the name of the control: C1DateTimePicker1.
2. Locate the **Background**, **BorderBrush**, and **BorderThickness** properties and set them as follows:
   - **Background**: #FFA6FCB6
   - **BorderBrush**: #FF29ACEB
   - **BorderThickness**: 5

   The Properties window should resemble the following image:



3. Locate the **Foreground** property and set it to "#FF0C3EC4". This property sets the color of the text.
4. Repeat these steps with the next two instances of the **C1DateTimePicker** control. All three controls should have the same color scheme when you run your application, as in the following image:


ComponentOne DateTimePicker for Windows Phone
Date and Time
Edit date
Friday, March 09, 2012
Edit time
12:00:00 AM
Edit date and time
Friday, March 09, 2012 - 12:00:00 AM

Layout Panels

Control the flow and positioning of the content on your Windows Phone application with **ComponentOne Layout Panels™ for Windows Phone**. Wrap content vertically or horizontally using C1WrapPanel. Dock content along the edges of the panel with C1DockPanel. Display content in a grid using C1UniformGrid.

Layout Panels for Windows Phone Features

Wrap Panel

Create flowing layouts that wrap content vertically or horizontally using **ComponentOne WrapPanel™ for Windows Phone**. This control is similar to the **WrapPanel** class in WPF. Features include:

- **Wrapping Layouts**

  C1WrapPanel positions child elements in sequential position from left to right, breaking content to the next line at the edge of the containing box. This is useful when you need wrapping content to break at different positions depending on the orientation of the phone.

- **Horizontal or Vertical Flow Layout**

  Switch the flow to vertical and child elements will arrange from top to bottom. Just set the **Orientation** property.

Dock Panel

ComponentOne DockPanel™ for Windows Phone enables you to dock elements to the top, bottom, left and right edges of a parent container. Add this classic Windows Forms functionality into your Phone development toolkit to create complex layouts with ease. Features include:

- **Filled Content**

  If you set the **LastChildFill** property to **True**, which is the default setting, the last child element of a **C1DockPanel** always fills the remaining (central) space.

- **Create Complex Layouts**

  You can nest **C1DockPanel**s to create complex layouts you are familiar with in Windows Forms. Set the attached **Dock** property of nested child elements to position along any edge of its parent container. As the **C1DockPanel** resizes during orientation changes, nested elements remain docked to their respective edges.

Uniform Grid

Neatly display child elements in rows and columns with **ComponentOne UniformGrid™ for Windows Phone**. Each child element is displayed at an equal width and height. Features include:

- **Rows and Columns**

  Specify the number of rows and columns by just setting two properties.

- **Hide Columns and Rows**

  Using **C1UniformGrid**, you can show or hide an entire column or row with one line of code.

- **Span Columns and Rows**

  By setting the attached **ColumnSpan** property you can span columns. Or, span rows by setting the attached **RowSpan** property. This functionality resembles the **Grid** control.

- **Set the First Column**

Use the **FirstColumn** property to set the number of leading blank cells in the first row of the **C1UniformGrid**.

Layout Panels for Windows Phone Quick Starts

A quick start is included for each layout panel in **Studio for Windows Phone**. In each quick start, you'll begin by creating a Windows Phone application and then you will add styles for the controls. For **WrapPanel**, you will wrap several **HyperlinkButton**s. For **DockPanel**, you will create the panel with several elements inside, demonstrating the four docking options for C1DockPanel. For **UniformGrid**, you will create a grid with three columns and two empty cells in the first row.

Choose one of the following quick starts to get started:

- WrapPanel for Windows Phone Quick Start
- DockPanel for Windows Phone Quick Start
- UniformGrid for Windows Phone Quick Start

WrapPanel for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **WrapPanel for Windows Phone**. In this quick start, you'll create a new project in Visual Studio, add styled **HyperlinkButton**s that can be wrapped, and change the orientation for the buttons.

Step 1 of 3: Creating a Windows Phone Application

In this step you'll create a Windows Phone application in Visual Studio 2010 using **ComponentOne Layout Panels for Windows Phone**.

To set up your project, complete the following steps:

1. In Visual Studio 2010, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="WrapPanel" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

In the next step, you'll add, style, and wrap several **HyperlinkButton**s.

Step 2 of 3: Adding a C1WrapPanel

We're going to use simple **HyperlinkButton**s to show how content can be wrapped vertically or horizontally. This is the typical scenario to create a TagCloud view; very commonly used in Web applications.

1. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
2. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a C1WrapPanel:

```
<c1:C1WrapPanel x:Name="c1wrappanel1"></c1:C1WrapPanel>
```

3. Add the following markup between the `<c1:C1WrapPanel ></c1:C1WrapPanel>` tags to add several **TextBlock** controls to the C1WrapPanel:

```
<TextBlock Text="Example Text " FontSize="30" />
<TextBlock Text="You can even wrap longer sentences." />
<TextBlock Text="Let's insert a break. " FontSize="24"/>
<TextBlock c1:C1WrapPanel.BreakLine="After" Text="Break After " />
<TextBlock Text="C1WrapPanel " />
<TextBlock Text="Wrap Vertically " />
<TextBlock Text="Wrap Horizontally " FontSize="20"  />
<TextBlock c1:C1WrapPanel.BreakLine="Before" Text="Break Before " />
<TextBlock Text="Silverlight-based " FontSize="8" />
<TextBlock Text="Windows Phone " FontSize="40"/>
<TextBlock Text="Controls " FontSize="18" />
<TextBlock c1:C1WrapPanel.BreakLine="AfterAndBefore" Text="Break After and
Before " />
<TextBlock Text="Create flow type layouts that wrap content. " />
<TextBlock Text="Small font size is not recommended. " FontSize="6" />
<TextBlock Text="The End" FontSize="24" />
```

The **ContentPanel** markup will now appear similar to the following:

```
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <c1:C1WrapPanel x:Name="c1wrappanel1">
        <TextBlock Text="Example Text " FontSize="30" />
        <TextBlock Text="You can even wrap longer sentences." />
        <TextBlock Text="Let's insert a break. " FontSize="24"/>
        <TextBlock c1:C1WrapPanel.BreakLine="After" Text="Break After " />
        <TextBlock Text="C1WrapPanel " />
        <TextBlock Text="Wrap Vertically " />
        <TextBlock Text="Wrap Horizontally " FontSize="20"  />
        <TextBlock c1:C1WrapPanel.BreakLine="Before" Text="Break Before "
/>
        <TextBlock Text="Silverlight-based " FontSize="8" />
        <TextBlock Text="Windows Phone " FontSize="40"/>
        <TextBlock Text="Controls " FontSize="18" />
```

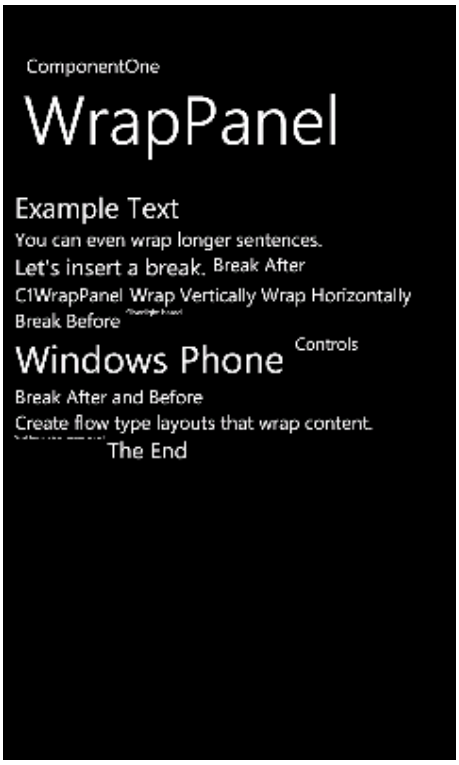
```
        <TextBlock c1:C1WrapPanel.BreakLine="AfterAndBefore" Text="Break
After and Before " />
        <TextBlock Text="Create flow type layouts that wrap content. " />
        <TextBlock Text="Small font size is not recommended. "
FontSize="6" />
        <TextBlock Text="The End" FontSize="24" />
    </c1:C1WrapPanel>
</Grid>
```

In the next step, you'll run the application.

Step 3 of 3: Customizing the Application

Now you're ready to run the application. Complete the following steps:

1. From the **Debug** menu, select **Start Debugging**. Your application will open in the Windows Phone emulator and will appear similar to the following:



2. Click the **Rotate Right** or **Rotate Left** button in the menu to view how the application would appear in landscape mode. Notice that the application does not rotate. In the next step you'll customize the application so that it rotates when the phone is turned.
3. Click the **Stop Debugging** button to close the application.
4. Edit the `<phone:PhoneApplicationPage>` tag to set the **SupportedOrientations** property to **PortraitOrLandscape**. It will appear similar to the following:

   - XAML Markup

     ```
     <phone:PhoneApplicationPage x:Class="C1WP7.MainPage"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     ```

```
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

By default when the phone is rotated, the application's interface does not rotate. By setting the **SupportedOrientations** property to **PortraitOrLandscape** the application will now rotate when the phone is rotated. In the next step you'll customize the appearance of the application when the phone is rotated to landscape mode.

5. In XAML view, click once on the `<phone:PhoneApplicationPage>` tag to select it, navigate to the Properties window, and select the **Event** tab to view the available events.
6. In the Properties window, locate and double-click the space next to the **OrientationChanged** event. This will create the **OrientationChanged** event handler and switch to Code view.
7. Add code to the **OrientationChanged** event handler so it appears similar to the following:

   - Visual Basic

```
Private Sub PhoneApplicationPage_OrientationChanged(ByVal sender As
System.Object, ByVal e As
Microsoft.Phone.Controls.OrientationChangedEventArgs) Handles
MyBase.OrientationChanged
    If (e.Orientation) = (PageOrientation.Landscape) Then
        Me.c1wrappanel1.Orientation =
System.Windows.Controls.Orientation.Horizontal
    Else If ((e.Orientation) = (PageOrientation.Portrait))
        Me.c1wrappanel1.Orientation =
System.Windows.Controls.Orientation.Vertical
    End If
End Sub
```

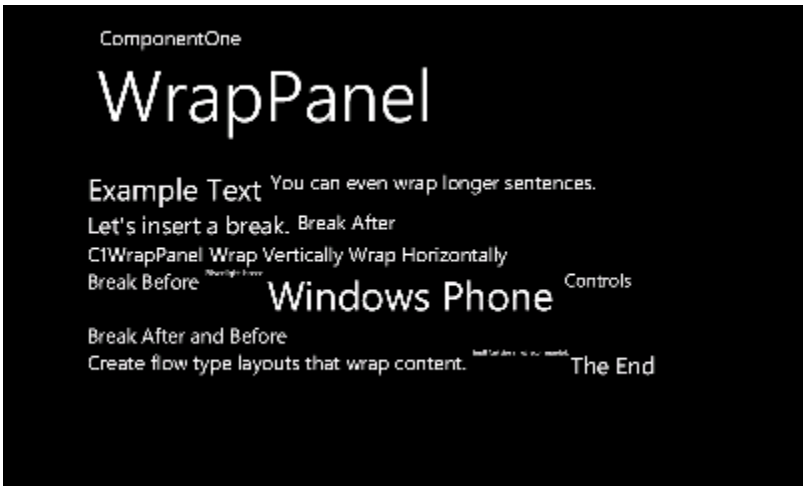
   - C#

```
private void PhoneApplicationPage_OrientationChanged(object sender,
OrientationChangedEventArgs e)
{
    if ((e.Orientation) == (PageOrientation.Landscape))
    {
        this.c1wrappanel1.Orientation =
System.Windows.Controls.Orientation.Horizontal;
    }
    else if ((e.Orientation) == (PageOrientation.Portrait))
    {
        this.c1wrappanel1.Orientation =
System.Windows.Controls.Orientation.Vertical;
    }
}
```

The Orientation property determines if items in the panel are displayed horizontally or vertically. By default Orientation is set to **Vertical** and the panel displays content vertically; setting Orientation property to **Horizontal** will display content horizontally. In the above code, the Orientation property is set in the

OrientationChanged event handler so that when the phone is in portrait mode the **C1WrapPanel** content will be displayed vertically and when the phone is in landscape mode the content will be displayed horizontally

8. Click **Start Debugging** again in the **Debug** menu. When the application is displayed in landscape mode it will now appear like the following:



Congratulations! You have successfully completed the **WrapPanel for Windows Phone** quick start.

DockPanel for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **DockPanel for Windows Phone**. In this quick start, you'll create a new project in Visual Studio, and add elements docked on the top, bottom, left, right, or even fill the C1DockPanel.

Step 1 of 3: Creating a Windows Phone Application

In this step you'll create a Windows Phone application in Visual Studio 2010 using **ComponentOne DockPanel for Windows Phone**.

To set up your project, complete the following steps:

1. In Visual Studio 2010, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
```

```
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="DockPanel" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

In the next step, you'll add and style **C1DockPanel**s.

Step 2 of 3: Adding a C1DockPanel

In this step you'll add and style several **C1DockPanel**s.

1. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
2. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a C1DockPanel:

```
<c1:C1DockPanel x:Name="c1dockpanel1"></c1:C1DockPanel>
```

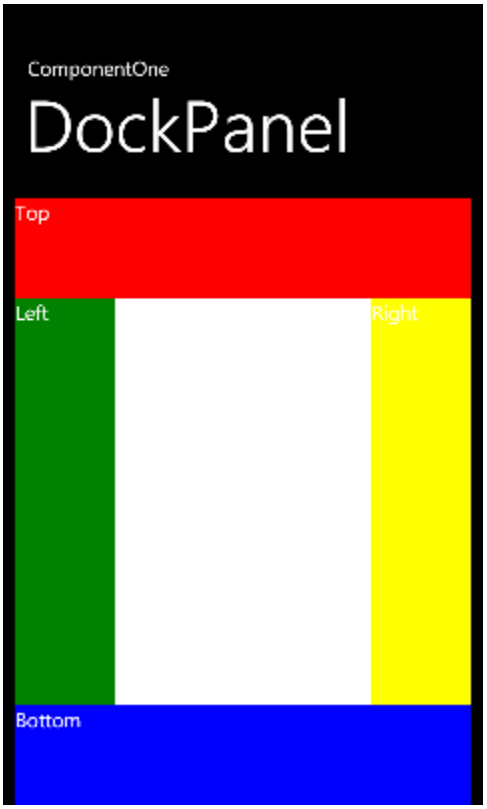
3. Place your cursor in between the `<c1:C1DockPanel>` tags and enter the following XAML to add **C1DockPanels** on the left, right, top, and bottom and to fill the center of the screen:

```
   <Border c1:C1DockPanel.Dock="Top" Height="100" Background="Red">
      <TextBlock Text="Top" />
   </Border>
   <Border c1:C1DockPanel.Dock="Bottom" Height="100" Background="Blue">
      <TextBlock Text="Bottom" />
   </Border>
   <Border c1:C1DockPanel.Dock="Right" Width="100" Background="Yellow">
      <TextBlock Text="Right" />
   </Border>
   <Border c1:C1DockPanel.Dock="Left" Background="Green" Width="100" >
      <TextBlock Text="Left" />
   </Border>
   <Border Background="White" >
      <TextBlock Text="Fill" />
   </Border>
```

In the next step, you'll run the application.

Step 3 of 3: Running the Application

Now you're ready to run the application. From the **Debug** menu, select **Start Debugging**. Your application will appear in the Windows Phone emulator and will look similar to the following, with four **C1DockPanels** on the top, right, left, and bottom and the center block filled:



Congratulations! You have successfully completed the **DockPanel for Windows Phone** quick start.

UniformGrid for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **UniformGrid for Windows Phone**. In this quick start, you'll create a new project in Visual Studio, add a C1UniformGrid to your application, set the Columns, FirstColumn, and **Width** properties, and then run the application.

Step 1 of 4: Creating a Windows Phone Application

In this step you'll create a Windows Phone application in Visual Studio 2010 using **ComponentOne UniformGrid for Windows Phone**.

To set up your project, complete the following steps:

1. In Visual Studio 2010, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="UniformGrid" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

In the next step, you'll add, style, and wrap several **HyperlinkButton**s.

Step 2 of 4: Adding a C1UniformGrid

Next we are going to add a C1UniformGrid.

1. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
2. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a C1DockPanel:

```
<c1:C1UniformGrid x:Name="c1uniformgrid1"></c1:C1UniformGrid>
```

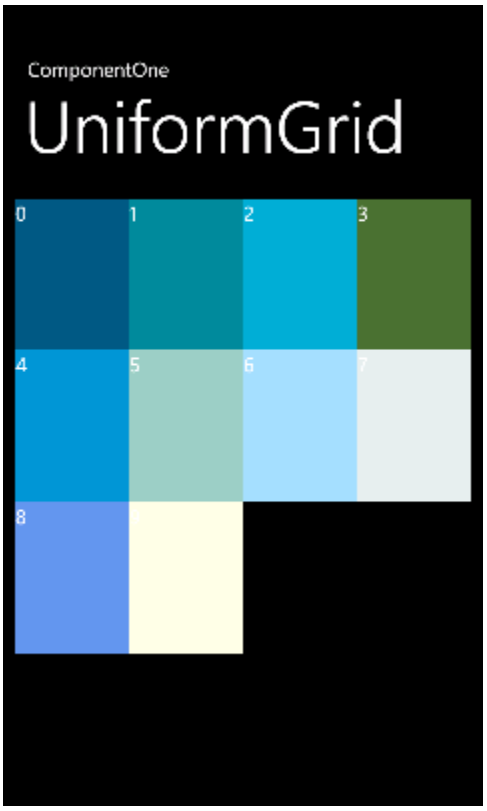
3. Place your cursor in between the `<c1:C1UniformGrid>` tags and enter the following XAML to add the grid's child elements, or numbered cells in this case:

- XAML Markup

```
<Border Background="#FF005B84" >
    <TextBlock Text="0" />
</Border>
<Border Background="#FF008B9C" >
    <TextBlock Text="1" />
</Border>
<Border Background="#FF00ADD6" >
    <TextBlock Text="2" />
</Border>
<Border Background="#FF497331" >
    <TextBlock Text="3" />
</Border>
<Border Background="#FF0094D6" >
    <TextBlock Text="4" />
</Border>
<Border Background="#FF9DCFC3" >
    <TextBlock Text="5" />
</Border>
```

```
<Border Background="#FFA5DDFE" >
    <TextBlock Text="6" />
</Border>
<Border Background="#FFE0EEEF" >
    <TextBlock Text="7" />
</Border>
<Border Background="CornflowerBlue" >
    <TextBlock Text="8" />
</Border>
<Border Background="LightYellow" >
    <TextBlock Text="9" />
</Border>
```

If you run the application at this point, it looks like the following image:



Step 3 of 4: Setting C1UniformGrid Properties

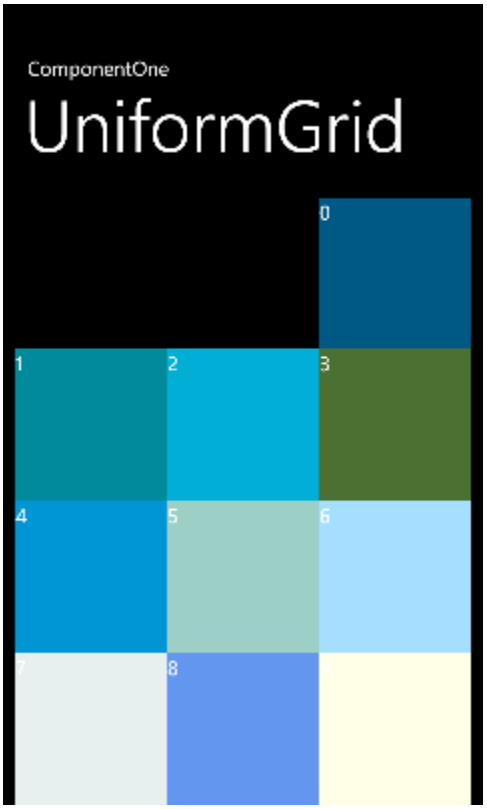
In this step, we'll set the set the Columns, FirstColumn, and **Width** properties. The Columns will set the number of columns in the grid, the **Width** property will set the width, in pixels, and the FirstColumn property will determine how many empty cells will appear in the first row.

1. Place your cursor within the opening `<c1:C1UniformGrid>` grid tag.
2. Set the Columns, FirstColumn, and **Width** properties using the following XAML:
   ```
   <c1:C1UniformGrid Columns="3" FirstColumn="2">
   ```
   This will give the grid three columns. There will be two empty cells in the first row of the grid, as specified with the FirstColumn property.

In the next step, you will run the application to see the results.

Step 4 of 4: Running the Application

Now you're ready to run the application. From the **Debug** menu, select **Start Debugging**. Your application will look similar to the following:



Notice the two empty cells in the first row.

Congratulations! You have successfully completed the **UniformGrid for Windows Phone** quick start.

Layout Panels for Windows Phone Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the Layout Panels in general. If you are unfamiliar with the **ComponentOne Layout Panels for Windows Phone** product, please see the Layout Panels for Windows Phone Quick Starts first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Layout Panels for Windows Phone** product. Each task-based help topic also assumes that you have created a new Windows Phone project.

Wrapping Items with C1WrapPanel

You can wrap items using the **C1WrapPanel.BreakLine** attached property. In this example, **HyperlinkButton**s are used. Complete the following steps:

1. Right-click the project in the Solution Explorer and select **Add Reference**.
2. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.

3. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

4. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="WrapPanel" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

5. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.

6. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a C1WrapPanel:

```
<c1:C1WrapPanel x:Name="c1wrappanel1"></c1:C1WrapPanel>
```

7. Place your cursor in between the `<c1:C1WrapPanel>` tags and press ENTER.

8. Add the following XAML to wrap **HyperlinkButton**s:

```
<Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
    <HyperlinkButton Foreground="White"  Content="Example Text"
FontSize="25" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
    <HyperlinkButton Foreground="White" Content="Break After" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
    <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
    <HyperlinkButton Foreground="White" Content="Wrap Vertically" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
    <HyperlinkButton Foreground="White" Content="Wrap Horizontally"
FontSize="20"  />
```

```
</Border>
```

Notice the **C1WrapPanel.BreakLine** property is set to **After** for the second **HyperlinkButton**. This will add a break after the button.

9. Run your project. The C1WrapPanel will resemble the following image:

Notice there is a break after the second **HyperlinkButton**.

Wrapping Items Vertically with C1WrapPanel

By default, items are wrapped horizontally. However, in some cases you may need them to wrap vertically. You can set the Orientation property to specify vertical wrapping. In this example, **HyperlinkButton**s are used. Complete the following steps:

1. Right-click the project in the Solution Explorer and select **Add Reference**.
2. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
3. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

4. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="WrapPanel" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

5. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.

6. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a C1WrapPanel:

```
<c1:C1WrapPanel x:Name="c1wrappanel1"></c1:C1WrapPanel>
```

7. In the `<c1:C1WrapPanel>` tag, set the **Orientation** property to **Vertical**; the XAML will look like the following:

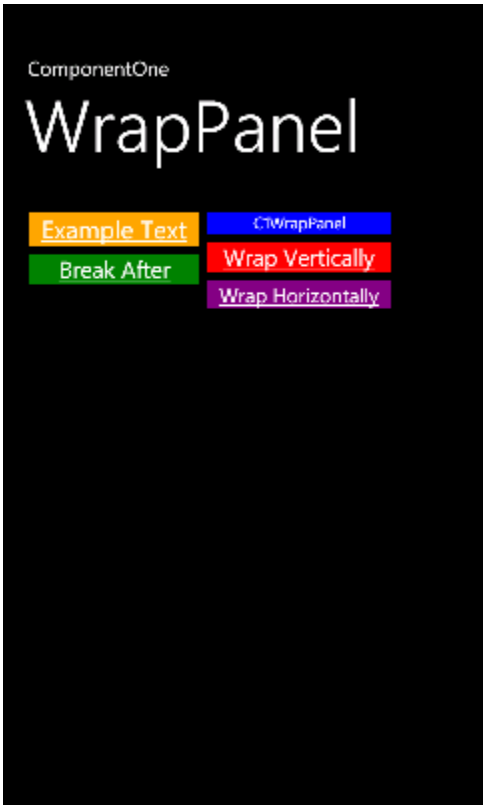
```
    <c1:C1WrapPanel x:Name="c1wrappanel1" Orientation="Vertical">
```

8. Place your cursor in between the `<c1:C1WrapPanel>` tags and press ENTER.

9. Add the following XAML to wrap **HyperlinkButton**s:

```
<Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
    <HyperlinkButton Foreground="White"  Content="Example Text"
FontSize="25" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
    <HyperlinkButton Foreground="White" Content="Break After" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
    <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
    <HyperlinkButton Foreground="White" Content="Wrap Vertically" />
</Border>
<Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
    <HyperlinkButton Foreground="White" Content="Silverlight"
FontSize="20"  />
</Border>
```

10. Run your project. The C1WrapPanel will resemble the following image:


ComponentOne
WrapPanel
Example Text
Break After
C1WrapPanel
Wrap Vertically
Wrap Horizontally

LayoutTransformer

Rotate, skew, translate and scale any elements of your UI with ease. **ComponentOne LayoutTransformer™ for Windows Phone** is similar to the **LayoutTransform** in WPF.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

LayoutTransformer for Windows Phone Key Features

ComponentOne LayoutTransformer for Windows Phone allows you to create customized, rich applications. Make the most of **LayoutTransformer for Windows Phone** by taking advantage of the following key features:

- **Rotate Any Element**

  Use a **RotateTransform** to rotate an object clockwise about a specified point in a 2-D x-y coordinate system.

- **Skew Any Element**

  You can distort an object by a specified angle from an axis using the **SkewTransform**. A skew transformation is useful for creating the illusion of 3D depth in a 2-D object.

- **Scale Any Element**

  Use a **ScaleTransform** to stretch or shrink an object horizontally or vertically.

- **Custom Transformations**

  Use the **MatrixTransform** class to create custom transformations that are not provided by the **RotateTransform**, **SkewTransform**, **ScaleTransform**, or **TranslateTransform** classes.

LayoutTransformer for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **LayoutTransformer for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add a **LayoutTransformer for Windows Phone** panel to your application, add controls within C1LayoutTransformer, and customize the appearance and behavior of the panel.

You will create a simple application using a C1LayoutTransformer panel that contains several items. You'll then customize the C1LayoutTransformer panel's appearance and behavior settings to explore the possibilities of using **LayoutTransformer for Windows Phone**.

Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **LayoutTransformer for Windows Phone**..

To set up your project, complete the following steps:

1. In Visual Studio 2010, select **File | New | Project** to open the **New Project** dialog box.

2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage  x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="LayoutTransformer" FontSize="56"
Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

You've successfully created a Windows Phone application.. In the next step you'll add controls to and customize C1LayoutTransformer transforms.

Step 2 of 3: Adding Transforms

In the previous step you created a new Windows Phone project and added a C1LayoutTransformer panel to the application. In this step you'll continue by adding controls with transforms.

Complete the following steps:

1. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
2. Add the following markup between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a **C1LayoutTransformer** control to the application:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <RotateTransform Angle="90"></RotateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="DarkMagenta" Content="Rotate" Height="80"
Width="180" HorizontalAlignment="Right" VerticalAlignment="Top"/>
</c1:C1LayoutTransformer>
```

Note that the **C1LayoutTransformer** includes a **RotateTransform** that changes the apperance of the **Button** control that the **C1LayoutTransformer** contains. Note that the **RotateTransform** rotates the **Button** by 90 degrees.

3. Add the following markup below the previous **C1LayoutTransfomer** markup and between the `<Grid x:Name="ContentPanel"></Grid>` tags to add an additional **C1LayoutTransformer** control to the application:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <ScaleTransform ScaleX="2" ScaleY="4"></ScaleTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumVioletRed" Content="Scale" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
</c1:C1LayoutTransformer>
```

Note that the **C1LayoutTransformer** includes a **ScaleTransform** that changes the appearance of the **Button** control that the **C1LayoutTransformer** contains. Note that the **ScaleTransform** scales the button to twice its initial width and four times its initial height.

4. Add the following markup below the previous **C1LayoutTransfomer** markup and between the `<Grid x:Name="ContentPanel"></Grid>` tags to add an additional **C1LayoutTransformer** control to the application:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <SkewTransform AngleX="140" AngleY="140"></SkewTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="HotPink" Content="Skew" Height="80" Width="180"
VerticalAlignment="Top" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>
```

Note that the **C1LayoutTransformer** includes a **SkewTransform** that changes the appearance of the **Button** control that the **C1LayoutTransformer** contains. Note that the **SkewTransform** skews the button to 140 degree angle both horizontally and vertically.

5. Add the following markup below the previous **C1LayoutTransfomer** markup and between the `<Grid x:Name="ContentPanel"></Grid>` tags to add an additional **C1LayoutTransformer** control to the application:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <TranslateTransform X="50" Y="50"></TranslateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="DarkOrchid" Content="Translate" Height="80"
Width="180" VerticalAlignment="Center" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>
```

Note that the **C1LayoutTransformer** includes a **TranslateTransform** that changes the appearance of the **Button** control that the **C1LayoutTransformer** contains. Note that the **TranslateTransform** changes the location of the control along the X and Y axes.

6. Add the following markup below the previous **C1LayoutTransfomer** markup and between the `<Grid x:Name="ContentPanel"></Grid>` tags to add an additional **C1LayoutTransformer** control to the application:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <MatrixTransform >
            <MatrixTransform.Matrix >
            <Matrix M11="-1" M12="1" M21="-1" M22="-2"/>
            </MatrixTransform.Matrix>
        </MatrixTransform>
```

```
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumPurple" Content="Matrix" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
</c1:C1LayoutTransformer>
```

Note that the **C1LayoutTransformer** includes a **MaxtrixTransform** that changes the appearance of the **Button** control that the **C1LayoutTransformer** contains. Note that the **MatrixTransform** represents a 3x3 affine transformation matrix used for transformations in 2-D space and so changes, stretches, and skews the appearance of the **Button**.

Your complete markup will appear like the following:

- XAML Markup

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <RotateTransform Angle="90"></RotateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="DarkMagenta" Content="Rotate" Height="80"
Width="180" HorizontalAlignment="Right" VerticalAlignment="Top"/>
</c1:C1LayoutTransformer>

<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <ScaleTransform ScaleX="2" ScaleY="4"></ScaleTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumVioletRed" Content="Scale" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
</c1:C1LayoutTransformer>

<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <SkewTransform AngleX="140" AngleY="140"></SkewTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="HotPink" Content="Skew" Height="80" Width="180"
VerticalAlignment="Top" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>

<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <TranslateTransform X="50" Y="50"></TranslateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="DarkOrchid" Content="Translate" Height="80"
Width="180" VerticalAlignment="Center" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>

<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <MatrixTransform >
            <MatrixTransform.Matrix >
                <!-- OffsetX and OffsetY specify the position of the
button,
                     M11 stretches it, and M12 skews it. -->
            <Matrix M11="-1" M12="1" M21="-1" M22="-2"/>
            </MatrixTransform.Matrix>
        </MatrixTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumPurple" Content="Matrix" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
```

```
</c1:C1LayoutTransformer>
```

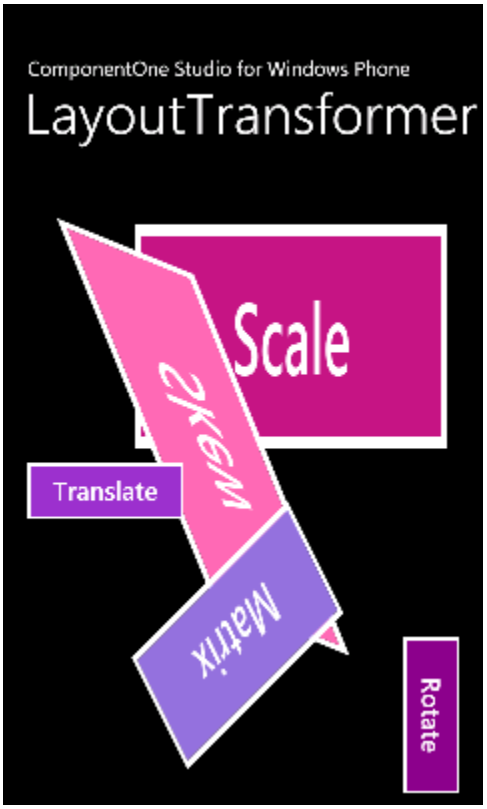
You've successfully created a Windows Phone application and added C1LayoutTransformer and controls to the application. In the next step you'll view C1LayoutTransformer at run time.

Step 3 of 3: Running the Application

In the previous steps you created a new Windows Phone project and added C1LayoutTransformers and several controls to the application. In this step you'll run the application.

Complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view you application in the Windows Phone emulator. It will appear similar to the following:



2. Stop debugging the application and return to the XAML view of the page.
3. Edit the `<phone:PhoneApplicationPage>` tag to set the **SupportedOrientations** property to **PortraitOrLandscape**. It will appear similar to the following:

- XAML Markup

```
<phone:PhoneApplicationPage x:Class="C1WP7.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

By default when the phone is rotated, the application's interface does not rotate. By setting the **SupportedOrientations** property to **PortraitOrLandscape** the application will now rotate when the phone is rotated.

4. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Notice that the C1LayoutTransformer control still appears vertical.
5. In the menu, click the **Rotate Left** or **Rotate Right** button to rotate the emulator to landscape mode. Notice that the application now rotates.

Congratulations! You've completed the **LayoutTransformer for Windows Phone** quick start and created a **ComponentOne LayoutTransformer for Windows Phone** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Working with LayoutTransformer for Windows Phone

ComponentOne LayoutTransformer for Windows Phone includes the C1LayoutTransformer control. Similar to the **LayoutTransform** in WPF, **C1LayoutTransformer** allows you to rotate, skew, translate and scale any elements of your UI with ease. When you add the C1LayoutTransformer to a XAML window, it exists as an empty control where you can add controls and content similarly as you would to any other panel such as the **Grid** or **Canvas**.

Basic Transforms

ComponentOne LayoutTransformer for Windows Phone supports several transforms that allow you to change the appearance of items in the control. The following transforms let you customize the C1LayoutTransformer control:

Property	Description
MatrixTransform	Creates an arbitrary affine matrix transformation that is used to manipulate objects or coordinate systems in a two-dimensional plane.
RotateTransform	Rotates an object clockwise about a specified point in a two-dimensional x-y coordinate system.
ScaleTransform	Scales an object in the two-dimensional x-y coordinate system.
SkewTransform	Represents a two-dimensional skew.
TransformGroup	Represents a composite **Transform** composed of other **Transform** objects.
TranslateTransform	Translates (moves) an object in the two-dimensional x-y coordinate system.

MatrixTransform

The **MatrixTransform** creates an arbitrary affine matrix transformation that is used to manipulate objects or coordinate systems in a two-dimensional plane. Use a **MatrixTransform** to create custom transformations that are not provided by the **RotateTransform**, **ScaleTransform**, **SkewTransform**, and **TranslateTransform** classes.

A two-dimensional x-y plane uses a 3 x 3 matrix for transformations. You can multiply affine matrix transformations to form linear transformations, such as rotation and skew (shear) transformations that are followed by translation.

An affine matrix transformation has its final column equal to (0, 0, 1); therefore, you only have to specify the members in the first two columns.

The members in the last row, **OffsetX** and **OffsetY**, represent translation values.

Methods and properties usually specify the transformation matrix as a vector that has only six members; the members are as follows: (M11, M12, M21, M22, OffsetX, OffsetY).

Values include:

- **m11**: The value at position (1, 1) of the transformation **Matrix**.
- **m12**: The value at position (1, 2) of the transformation **Matrix**.
- **m21**: The value at position (2, 1) of the transformation **Matrix**.
- **m22**: The value at position (2, 2) of the transformation **Matrix**.
- **offsetX**: The value at position (3, 1) of the transformation **Matrix**.
- **offsetY**: The value at position (3, 2) of the transformation **Matrix**.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <MatrixTransform >
            <MatrixTransform.Matrix >
            <Matrix M11="-1" M12="1" M21="-1" M22="-2"/>
            </MatrixTransform.Matrix>
        </MatrixTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumPurple" Content="Matrix" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
</c1:C1LayoutTransformer>
```

Multiple transforms can be applied with a **TransformGroup**.

RotateTransform

The **RotateTransform** rotates an object clockwise about a specified point in a two-dimensional x-y coordinate system. A **RotateTransform** is defined by the following properties: Angle rotates an object by a specified angle about the point **CenterX**, **CenterY**.

When you use a **RotateTransform**, the transformation rotates the coordinate system for a particular object about the origin point for its frame of reference. Therefore, depending on the position of the object, it might not rotate in place (around its center). For example, if an object is positioned 200 units from 0 along the x-axis, a rotation of 30 degrees can swing the object 30 degrees along a circle that has a radius of 200, which is drawn around the origin. To rotate an object in place, set the **CenterX** and **CenterY** of the **RotateTransform** to the center of the object to rotate.

Multiple transforms can be applied with a **TransformGroup**.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <RotateTransform Angle="90"></RotateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
```

```
    <Button Background="DarkMagenta" Content="Rotate" Height="80"
Width="180" HorizontalAlignment="Right" VerticalAlignment="Top"/>
</c1:C1LayoutTransformer>
```

ScaleTransform

The **ScaleTransform** scales an object in the two-dimensional x-y coordinate system. Use a **ScaleTransform** to stretch or shrink an object horizontally or vertically. The **ScaleX** property specifies the amount to stretch or shrink an object along the x-axis, and the **ScaleY** property specifies the amount to stretch or shrink an object along the y-axis. Scale operations are centered on the point specified by the **CenterX** and **CenterY** properties.

Multiple transforms can be applied with a **TransformGroup**. Custom transforms can be created with a **MatrixTransform**.

Transforms can alter the display of text in your application to create a decorative effect.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <ScaleTransform ScaleX="2" ScaleY="4"></ScaleTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="MediumVioletRed" Content="Scale" Height="80"
Width="180" VerticalAlignment="Top" HorizontalAlignment="Right"/>
</c1:C1LayoutTransformer>
```

SkewTransform

The **SkewTransform** represents a two-dimensional skew. A **SkewTransform** is useful for creating the illusion of three-dimensional depth in a two-dimensional object.

Multiple transforms can be applied with a **TransformGroup**. Custom transforms can be created with a **MatrixTransform**. Transforms can alter the display of text in your application to create a decorative effect.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <SkewTransform AngleX="140" AngleY="140"></SkewTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="HotPink" Content="Skew" Height="80" Width="180"
VerticalAlignment="Top" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>
```

TransformGroup

The **TransformGroup** represents a composite **Transform** composed of other **Transform** objects. Typically these are the Silverlight defined classes **RotateTransform**, **ScaleTransform**, **SkewTransform**, **TranslateTransform**, **MatrixTransform**, or **TransformGroup**. Object elements defined here become members of the **TransformCollection** collection when code accesses the **Children** property at run time.

Use a **TransformGroup** when you want to apply multiple **Transform** operations to a single object.

In a composite transformation, the order of individual transformations is important. For example, if you first rotate, then scale, then translate, you get a different result than if you first translate, then rotate, then scale. One reason order is significant is that transformations like rotation and scaling are done with respect to the origin of the coordinate system. Scaling an object that is centered at the origin produces a different result than scaling an object that has been moved away from the origin. Similarly, rotating an object that is centered at the origin produces a different result than rotating an object that has been moved away from the origin.

In XAML usages, **TransformGroup** uses **Children** as its content property and supports implicit collection usage. Therefore, to declare transforms that will be in a **TransformGroup** in XAML, you declare one or more transforms

as object elements, placing them in order as the child elements of the **TransformGroup**. Nesting more than one **TransformGroup** is permitted.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <TransformGroup>
            <ScaleTransform ScaleX="0.8" ScaleY="0.8" />
            <SkewTransform AngleX="30" />
            <RotateTransform Angle="45" />
        </TransformGroup>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="OrangeRed" Content="Group" Height="80"
Width="180" VerticalAlignment="Bottom" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>
```

TranslateTransform

The **TranslateTransform** translates (moves) an object in the two-dimensional x-y coordinate system. Multiple transforms can be applied with a **TransformGroup**. Custom transforms can be created with a **MatrixTransform**.

TranslateTransform defines an axis-aligned translation along the x and y axes. Transforms can alter the display of text in your application to create a decorative effect.

For example:

```
<c1:C1LayoutTransformer >
    <c1:C1LayoutTransformer.LayoutTransform>
        <TranslateTransform X="50" Y="50"></TranslateTransform>
    </c1:C1LayoutTransformer.LayoutTransform>
    <Button Background="DarkOrchid" Content="Translate" Height="80"
Width="180" VerticalAlignment="Center" HorizontalAlignment="Left"/>
</c1:C1LayoutTransformer>
```

ListBox

Get two high performance controls for displaying lists of bound data with **ComponentOne ListBox™ for Windows Phone**. Display lists with tile layouts or with optical zoom using the **C1ListBox** and **C1TileListBox** controls. These controls support UI virtualization so they are blazing-fast while able to display thousands of items with little-to-no loss of performance.

Getting Started

Get started with the following topics:

- Key Features
- C1ListBox Quick Start
- C1TileListBox Quick Start

ListBox for Windows Phone Key Features

ListBox for Windows Phone's key features include the following:

- **Horizontal or Vertical Orientation**

  The **ComponentOne ListBox** controls support both horizontal and vertical orientation, allowing for more layout scenarios.

- **Display Items as Tiles**

  The **C1TileListBox** can arrange its items in both rows and columns creating tile displays. Specify the size and template of each item and select the desired orientation.

- **Optical Zoom**

  The **C1ListBox** control supports optical zoom functionality so users can manipulate the size of the items intuitively through pinch gestures. The zooming transformation is smooth and fluid so the performance of your application is not sacrificed.

- **UI Virtualization**

  The **ComponentOne ListBox** controls support UI virtualization so they are blazing-fast while able to display thousands of items with virtually no loss of performance. You can determine how many items are rendered in each layout pass by setting the **ViewportGap** and **ViewportPreviewGap** properties. These properties can be adjusted depending on the scenario.

- **Preview State**

  In order to have the highest performance imaginable, the **ListBox** controls can render items outside the viewport in a preview state. Like the standard **ItemTemplate**, the **Preview** template defines the appearance of items when they are in a preview state, such as zoomed out or during fast scroll. The controls will then switch to the full item template when the items have stopped scrolling or zooming.

C1ListBox Quick Start

The following quick start guide is intended to get you up and running with the **C1ListBox** control. In this quick start you'll start in Visual Studio and create a new project, add **C1ListBox** to your application, and customize the appearance and behavior of the control.

Step 1 of 3: Creating an Application with a C1ListBox Control

In this step, you'll create a Windows Phone application in Visual Studio using **ListBox for Windows Phone**.

Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="ListBox" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

8. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
9. Add the following markup between the `<Grid>` and `</Grid>` tags to add a **StackPanel** containing a **TextBlock** and **ProgressBar**:

```
<StackPanel x:Name="loading" VerticalAlignment="Center">
    <TextBlock Text="Retrieving data from Flickr..."
TextAlignment="Center" />
    <ProgressBar IsIndeterminate="True" Height="4" />
</StackPanel>
<Button x:Name="retry" HorizontalAlignment="Center"
VerticalAlignment="Center" Content="Retry" Visibility="Collapsed"
Click="Retry_Click"/>
```

   The **TextBlock** and **ProgressBar** will indicate that the **C1ListBox** is loading.
10. Add the following markup to add the **C1ListBox** control and customize the control:

```
<c1:C1ListBox x:Name="listBox" ItemsSource="{Binding}"
Background="Transparent" Visibility="Collapsed" ItemWidth="800"
ItemHeight="600" Zoom="Fill" ViewportGap="0"
RefreshWhileScrolling="False"></c1:C1ListBox>
```

This gives the control a name and customizes the binding, background, visibility, size, and refreshing ability of the control.

11. Add the following markup between the `<c1:C1ListBox>` and `</c1:C1ListBox>` tags:

```
<c1:C1ListBox.PreviewItemTemplate>
                <DataTemplate>
                    <Grid Background="Gray">
                        <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill"/>
                    </Grid>
                </DataTemplate>
            </c1:C1ListBox.PreviewItemTemplate>
            <c1:C1ListBox.ItemTemplate>
                <DataTemplate>
                    <Grid>
                        <Image Source="{Binding Content}"
Stretch="UniformToFill"/>
                        <TextBlock Text="{Binding Title}"
Foreground="White" Margin="4 0 0 4" VerticalAlignment="Bottom"/>
                    </Grid>
                </DataTemplate>
            </c1:C1ListBox.ItemTemplate>
```

12. This markup adds data templates for the **C1ListBox** control's content. Note that you'll complete binding the control in code.

What You've Accomplished

You've successfully created a Windows Phone application containing a **C1ListBox** control. In the next step, Step 2 of 3: Adding Data to the ListBox, you will add the data for **C1ListBox**.

Step 2 of 3: Adding Data to the ListBox

In the last step, you added the **C1ListBox** control to the application. In this step, you will add code to display images from a photo stream.

Complete the following steps to add data to the control programmatically:

1. Select **View | Code** to switch to Code view.
2. Add the following imports statements to the top of the page:
   - Visual Basic

```
Imports C1.Phone
Imports System.Xml.Linq
```

   - C#

```
using C1.Phone;
using System.Xml.Linq;
```

   Note that you may need to add a reference to System.Xaml.Linq as well.
3. Add the following code inside the initial event handler:
   - Visual Basic

```
DataContext = Enumerable.Range(0, 100)
AddHandler Loaded, AddressOf ListBoxSample_Loaded
```

- C#

```
DataContext = Enumerable.Range(0, 100);
Loaded += new System.Windows.RoutedEventHandler(ListBoxSample_Loaded);
```

4. Add the following code below within the **MainPage** class:

- Visual Basic

```
Private Sub ListBoxSample_Loaded(sender As Object, e As
RoutedEventArgs)
        LoadPhotos()
    End Sub

    Private Sub LoadPhotos()
        Dim flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne"
        Dim AtomNS = "http://www.w3.org/2005/Atom"
        loading.Visibility = Visibility.Visible
        retry.Visibility = Visibility.Collapsed

        Dim photos = New List(Of Photo)()
        Dim client = New WebClient()
        AddHandler client.OpenReadCompleted, Function(s, e)
                                                 Try
                                                     '#Region "** parse
flickr data"
                                                     Dim doc =
XDocument.Load(e.Result)
                                                     For Each entry In
doc.Descendants(XName.[Get]("entry", AtomNS))
                                                         Dim title =
entry.Element(XName.[Get]("title", AtomNS)).Value
                                                         Dim enclosure
= entry.Elements(XName.[Get]("link", AtomNS)).Where(Function(elem)
elem.Attribute("rel").Value = "enclosure").FirstOrDefault()
                                                         Dim contentUri
= enclosure.Attribute("href").Value
                                                         photos.Add(New
Photo() With { _
                                                           .Title =
title, _
                                                           .Content =
contentUri, _
                                                           .Thumbnail =
contentUri.Replace("_b", "_m") _
                                                         })
                                                     Next
                                                     '#End Region


listBox.ItemsSource = photos
                                                     loading.Visibility
= Visibility.Collapsed
                                                     listBox.Visibility
= Visibility.Visible
                                                 Catch
```

```
MessageBox.Show("There was an error when attempting to download data
from Flickr.")
                                                        loading.Visibility
= Visibility.Collapsed
                                                        retry.Visibility =
Visibility.Visible
                                                    End Try

                                                End Function
        client.OpenReadAsync(New Uri(flickrUrl))
    End Sub

    Private Sub Retry_Click(sender As Object, e As RoutedEventArgs)
        LoadPhotos()
    End Sub

#Region "** public properties"

    Public Property Orientation() As Orientation
        Get
            Return listBox.Orientation
        End Get
        Set(value As Orientation)
            listBox.Orientation = value
        End Set
    End Property

    Public Property ItemWidth() As Double
        Get
            Return listBox.ItemWidth
        End Get
        Set(value As Double)
            listBox.ItemWidth = value
        End Set
    End Property

    Public Property ItemHeight() As Double
        Get
            Return listBox.ItemHeight
        End Get
        Set(value As Double)
            listBox.ItemHeight = value
        End Set
    End Property

    Public Property ZoomMode() As ZoomMode
        Get
            Return listBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            listBox.ZoomMode = value
        End Set
    End Property
#End Region
```

- C#

```
void ListBoxSample_Loaded(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }

        private void LoadPhotos()
        {
            var flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne";
            var AtomNS = "http://www.w3.org/2005/Atom";
            loading.Visibility = Visibility.Visible;
            retry.Visibility = Visibility.Collapsed;

            var photos = new List<Photo>();
            var client = new WebClient();
            client.OpenReadCompleted += (s, e) =>
            {
                try
                {
                    #region ** parse flickr data
                    var doc = XDocument.Load(e.Result);
                    foreach (var entry in
doc.Descendants(XName.Get("entry", AtomNS)))
                    {
                        var title = entry.Element(XName.Get("title",
AtomNS)).Value;
                        var enclosure =
entry.Elements(XName.Get("link", AtomNS)).Where(elem =>
elem.Attribute("rel").Value == "enclosure").FirstOrDefault();
                        var contentUri =
enclosure.Attribute("href").Value;
                        photos.Add(new Photo() { Title = title, Content
= contentUri, Thumbnail = contentUri.Replace("_b", "_m") });
                    }
                    #endregion

                    listBox.ItemsSource = photos;
                    loading.Visibility = Visibility.Collapsed;
                    listBox.Visibility = Visibility.Visible;
                }
                catch
                {
                    MessageBox.Show("There was an error when attempting
to download data from Flickr.");
                    loading.Visibility = Visibility.Collapsed;
                    retry.Visibility = Visibility.Visible;
                }
            };
            client.OpenReadAsync(new Uri(flickrUrl));
        }

        private void Retry_Click(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }

        #region ** public properties
```

```
        public Orientation Orientation
        {
            get
            {
                return listBox.Orientation;
            }
            set
            {
                listBox.Orientation = value;
            }
        }

        public double ItemWidth
        {
            get
            {
                return listBox.ItemWidth;
            }
            set
            {
                listBox.ItemWidth = value;
            }
        }

        public double ItemHeight
        {
            get
            {
                return listBox.ItemHeight;
            }
            set
            {
                listBox.ItemHeight = value;
            }
        }

        public ZoomMode ZoomMode
        {
            get
            {
                return listBox.ZoomMode;
            }
            set
            {
                listBox.ZoomMode = value;
            }
        }
        #endregion
```

5. The code above pulls images from Flickr's public photo stream and binds the **C1ListBox** to the list of images.
6. Add the following code just below the **MainPage** class:

   - Visual Basic

```
Public Class Photo
    Public Property Title() As String
```

```
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
    Public Property Content() As String
        Get
            Return m_Content
        End Get
        Set(value As String)
            m_Content = Value
        End Set
    End Property
    Private m_Content As String
End Class
```

- C#

```
public class Photo
    {
        public string Title { get; set; }
        public string Thumbnail { get; set; }
        public string Content { get; set; }
    }
```

What You've Accomplished

You have successfully added data to **C1TileListBox**. In the next step, Step 3 of 3: Running the ListBox Application, you'll examine the **ListBox for Windows Phone** features.

Step 3 of 3: Running the ListBox Application

Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **ListBox for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging**.

   The application will appear in the Windows Phone emulator.

2. Use the scroll bar on the right of the control, to scroll through the image stream.
3. If you have touch capabilities, try pinching to zoom an image.

What You've Accomplished

Congratulations! You've completed the **ListBox for Windows Phone** quick start and created an application using the **C1ListBox** control and viewed some of the run-time capabilities of your application.

C1TileListBox Quick Start

The following quick start guide is intended to get you up and running with the **C1TileListBox** control. In this quick start you'll start in Visual Studio and create a new project, add **C1TileListBox** to your application, and customize the appearance and behavior of the control.

Step 1 of 3: Creating an Application with a C1TileListBox Control

In this step, you'll create a Windows Phone application in Visual Studio using **TileListBox for Windows Phone**.

Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="TileListBox" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

8. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
9. Add the following `<c1:C1TileListBox>` tag to add and the control:

```
<c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}"
ItemWidth="130" ItemHeight="130"></c1:C1TileListBox>
```

This gives the control a name, sets the **ItemsSource** property (you'll customize this in code in a later step), and sets the size of the control.

10. Add markup between the `<c1:C1TileListBox>` and `</c1:C1TileListBox>` tags so it looks like the following:

```
<c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}"
ItemWidth="130" ItemHeight="130">
    <c1:C1TileListBox.PreviewItemTemplate>
        <DataTemplate>
            <Grid Background="Gray" />
                </DataTemplate>
                </c1:C1TileListBox.PreviewItemTemplate>
                <c1:C1TileListBox.ItemTemplate>
                <DataTemplate>
                <Grid Background="LightBlue">
                    <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill" />
                    <TextBlock Text="{Binding Title}" Margin="4 0 0 4"
VerticalAlignment="Bottom" />
                </Grid>
        </DataTemplate>
    </c1:C1TileListBox.ItemTemplate>
</c1:C1TileListBox>
```

This markup adds data templates for the **C1TileListBox** control's content. Note that you'll complete binding the control in code.

What You've Accomplished

You've successfully created a Windows Phone application containing a **C1TileListBox** control. In the next step, Step 2 of 3: Adding Data to the TileListBox, you will add data for **C1TileListBox**.

Step 2 of 3: Adding Data to the TileListBox

In the last step, you added the **C1TileListBox** control to the application. In this step, you will add code to bind the control to data.

Complete the following steps to add data to the control programmatically:

1. Right-click the page and select **View Code** to open the Code Editor.
2. Add the following imports statements to the top of the page:
   - Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq
Imports System.Xml.Linq
Imports System.Net
Imports C1.Phone
```

   - C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using C1.Phone;
```

3. Add the following code inside the initial event handler within the **MainPage** class:

- Visual Basic

```
DataContext = Enumerable.Range(0, 100).[Select](Function(i) New Item()
With {.Title = i.ToString()})
```

- C#

```
DataContext = Enumerable.Range(0, 100).Select(i => new Item { Title =
i.ToString() });
```

4. Add the following code below the initial event handler but within the **MainPage** class:

- Visual Basic

```
#Region "** public properties"

    Public Property Orientation() As Orientation
        Get
            Return tileListBox.Orientation
        End Get
        Set(value As Orientation)
            tileListBox.Orientation = value
        End Set
    End Property

    Public Property ItemWidth() As Double
        Get
            Return tileListBox.ItemWidth
        End Get
        Set(value As Double)
            tileListBox.ItemWidth = value
        End Set
    End Property

    Public Property ItemHeight() As Double
        Get
            Return tileListBox.ItemHeight
        End Get
        Set(value As Double)
            tileListBox.ItemHeight = value
        End Set
    End Property

    Public Property ZoomMode() As ZoomMode
        Get
            Return tileListBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            tileListBox.ZoomMode = value
        End Set
    End Property
#End Region
```

- C#

```
#region ** public properties

        public Orientation Orientation
        {
            get
            {
                return tileListBox.Orientation;
```

```
            }
            set
            {
                tileListBox.Orientation = value;
            }
        }

        public double ItemWidth
        {
            get
            {
                return tileListBox.ItemWidth;
            }
            set
            {
                tileListBox.ItemWidth = value;
            }
        }

        public double ItemHeight
        {
            get
            {
                return tileListBox.ItemHeight;
            }
            set
            {
                tileListBox.ItemHeight = value;
            }
        }

        public ZoomMode ZoomMode
        {
            get
            {
                return tileListBox.ZoomMode;
            }
            set
            {
                tileListBox.ZoomMode = value;
            }
        }
        #endregion
```

The code above binds the **C1TileListBox** to a list of numbers.

5. Add the following code just below the **MainPage** class:

- Visual Basic

```
Public Class Item
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
```

```
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
End Class
```

- C#

```
public class Item
    {
        public string Title { get; set; }
        public string Thumbnail { get; set; }
    }
```

What You've Accomplished

You have successfully added data to **C1TileTileListBox**. In the next step, Step 3 of 3: Running the TileListBox Application, you'll examine the **TileListBox for Windows Phone** features.

Step 3 of 3: Running the TileListBox Application

Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **TileListBox for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging**.

   The application will appear in the Windows Phone emulator.
2. Use the scroll bar on the right of the control, to scroll through the numbered squares.
3. If you have touch capabilities, try pinching to zoom an image.

What You've Accomplished

Congratulations! You've completed the **TileListBox for Windows Phone** quick start and created an application using the **C1TileListBox** control and viewed some of the run-time capabilities of your application.

Top Tips

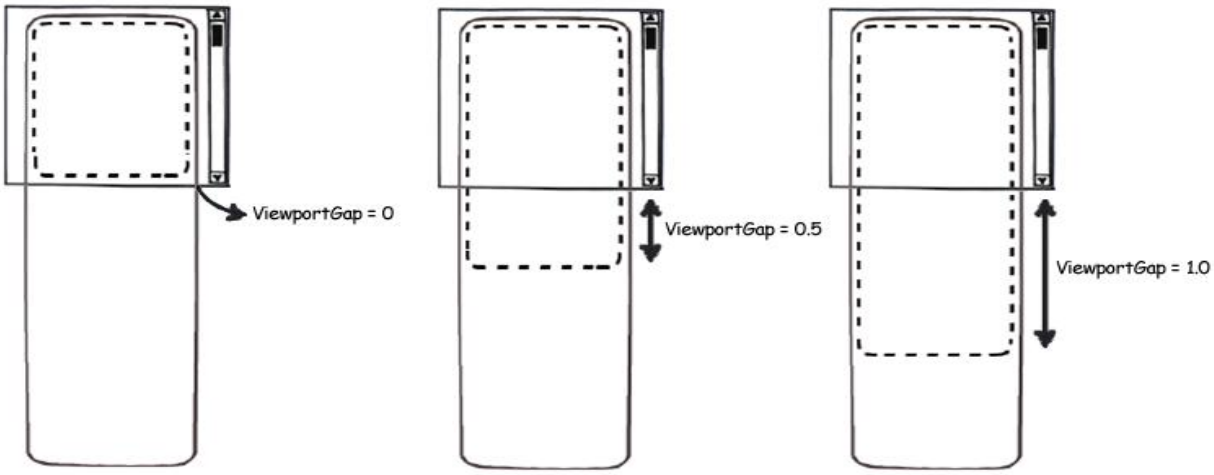
These tips will help you maximize your performance while using any of the **ComponentOne ListBox** controls.

- **Use PreviewTemplate** – In order to avoid making the layout heavier, the **PreviewTemplate** can be used to render a lighter template during preview states, such as while zooming and scrolling fast. For example you could display a thumbnail image in the **PreviewTemplate** and display the larger image in the full **ItemTemplate**.

```
<c1:C1ListBox x:Name="listBox"
                  ItemsSource="{Binding}"
                  RefreshWhileScrolling="False">
    <c1:C1ListBox.PreviewItemTemplate>
        <DataTemplate>
            <Grid Background="Gray">
                <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill"/>
            </Grid>
```

```
        </DataTemplate>
    </c1:C1ListBox.PreviewItemTemplate>
    <c1:C1ListBox.ItemTemplate>
        <DataTemplate>
            <Grid>
                <Image Source="{Binding Content}"
Stretch="UniformToFill"/>
            </Grid>
        </DataTemplate>
    </c1:C1ListBox.ItemTemplate>
</c1:C1ListBox>
```

- **Adjust the ViewportGap and ViewportPreviewGap Properties** – These coefficient values determine what size of items outside the viewport to render in advance. The larger the value the more quickly off-screen items will appear to render, but the slower the control will take on each layout pass. For example, if set to 0.5 the view port will be enlarged to take up a half screen more at both sides of the original view port.



- **Set RefreshWhileScrolling to False** – Determines whether the view port is refreshed while scrolling. If set to false items will appear blank or say "Loading" while the user scrolls real fast until they stop and allow items to render.

Working with ListBox for Windows Phone

ComponentOne ListBox for Windows Phone includes the C1ListBox and C1TileListBox controls. **C1ListBox** is similar to the standard Windows Phone **ListBox** control but it includes additional functionality, such as zooming. **C1TileListBox** allows you to create tiled lists of items. Display lists with tile layouts or with optical zoom using the **C1ListBox** and **C1TileListBox** controls.

Basic Properties

ComponentOne ListBox for Windows Phone includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below.

The following properties let you customize the C1ListBox control:

Property	Description
ActualMaxZoom	Gets the actual maximum zoom.

ActualMinZoom	Gets the actual minimum zoom.
ActualZoom	Gets the actual zoom.
IsScrolling	Gets a value indicating whether the list is scrolling.
IsZooming	Gets a value indicating whether this list is zooming.
ItemHeight	Gets or sets the height of each item.
Items	Gets the collection used to generate the content of the control. (Inherited from ItemsControl.)
ItemsPanel	Gets or sets the template that defines the panel that controls the layout of items. (Inherited from ItemsControl.)
ItemsSource	Gets or sets a collection used to generate the content of the ItemsControl. (Inherited from ItemsControl.)
ItemStyle	Style applied to all the items of this item control. (Inherited from C1ItemsControl.)
ItemTemplate	Template applied to all the items of the list. (Inherited from C1ItemsControl.)
ItemTemplateSelector	Template selector used to specify different templates applied to items of the same type. (Inherited from C1ItemsControl.)
ItemWidth	Gets or sets the width of each item.
MaxZoom	Gets or sets the maximum zoom available.
MinZoom	Gets or sets the minimum zoom available.
Orientation	Gets or sets the orientation in which the list is displayed.
Panel	Gets the panel associated with this items control.
PreviewItemTemplate	Gets or sets the template used for previewing an item.
RefreshWhileScrolling	Gets or sets a value indicating whether the viewport must be refreshed while the scroll is running.
ScrollViewer	Gets the scroll viewer template part belonging to this items control.
ViewportGap	Gets or sets a coeficient which will determine in each layout pass the size of the viewport. If zero is specified the size of the viewport will be equal to the scrollviewer viewport. If 0.5 is specified the size of the viewport will be enlarged to take up half screen more at both sides of the original viewport.
ViewportPreviewGap	Gets or sets a coeficient which will determine in each layout pass the size of the viewport to render items in preview mode.
Zoom	Gets or set the zoom applied to this list.
ZoomMode	Gets or sets whether the zoom is enabled or disabled.

The following properties let you customize the C1ListBoxItem:

Description	
PreviewContent	Gets or sets the content of the preview state.
PreviewContentTemplate	Gets or sets the DataTemplate used when in preview state.
State	Gets or sets the state of the item, which can be Preview or Full.

Optical Zoom

The **ListBox for Windows Phone** controls support optical zoom functionality so users can manipulate the size of the items intuitively through pinch gestures. The zooming transformation is smooth and fluid so the performance of your application is not sacrificed.

You can customize the zoom using the **ZoomMode** and **Zoom** properties. The **ZoomMode** property gets or sets whether the zoom is enabled or disabled. The **Zoom** property the **Zoom** value applied to the control. The **ZoomChanged** event is triggered when the zoom value of the control is changed.

UI Virtualization

The **ComponentOne ListBox** controls support UI virtualization so they are blazing-fast while able to display thousands of items with virtually no loss of performance. You can determine how many items are rendered in each layout pass by setting the **ViewportGap** and **ViewportPreviewGap** properties. These properties can be adjusted depending on the scenario.

The **ViewportGap** property gets or sets a coefficient which will determine in each layout pass the size of the viewport. If zero is specified the size of the viewport will be equal to the scrollviewer viewport. If 0.5 is specified the size of the viewport will be enlarged to take up half screen more at both sides of the original viewport.

The **ViewportPreviewGap** property gets or sets a coefficient which will determine in each layout pass the size of the viewport to render items in preview mode.

Orientation

The **ComponentOne ListBox** controls support both horizontal and vertical orientation, allowing for more layout scenarios. To set the orientation of the control, set the **Orientation** property to **Horizontal** or **Vertical**.

Preview State

In order to have the highest performance imaginable, the **ListBox** controls can render items outside the viewport in a preview state. Like the standard **ItemTemplate**, the **Preview** template defines the appearance of items when they are in a preview state, such as zoomed out or during fast scroll. The controls will then switch to the full item template when the items have stopped scrolling or zooming.

MaskedTextBox

Provide input mask validation with **ComponentOne MaskedTextBox™ for Windows Phone**. The **C1MaskedTextBox** control provides a text box with a mask that prevents users from entering invalid characters.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

MaskedTextBox for Windows Phone Features

ComponentOne MaskedTextBox for Windows Phone allows you to create customized, rich applications. Make the most of **MaskedTextBox for Windows Phone** by taking advantage of the following key features:

- **Mask Formatting**

  You can provide input validation and format how the text is displayed by setting the Mask property. **C1MaskedTextBox** supports the standard number formatting strings defined by Microsoft and uses the same syntax as the standard **MaskedTextBox** control in WinForms. This makes it easier to re-use masks across applications and platforms.

- **Include Prompts and Literals**

  Choose whether or not to show prompt characters and literals by simply setting one property. The prompt character indicates to the user that text can be entered (such as _ or *). Literals are non-mask characters that will appear as themselves within the mask (such as / or -).

- **Watermark Support**

  Using the **Watermark** property you can provide contextual clues of what value users should enter in the **C1MaskedTextBox** control. The watermark is displayed in the control while no text has been entered.

- **Metro UI Design**

  C1MaskedTextBox supports the Metro UI design and interaction guidelines specified by Microsoft.

MaskedTextBox for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **MaskedTextBox for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add **MaskedTextBox for Windows Phone** controls to your application, and customize the appearance and behavior of the controls.

You will create a simple form using several C1MaskedTextBox controls that will demonstrate the difference between the **Text** and **Value** properties. The controls will include various masks and different appearance and behavior settings so that you can explore the possibilities of using **MaskedTextBox for Windows Phone**.

Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **MaskedTextBox for Windows Phone**. When you add a C1MaskedTextBox control to your application, you'll have a complete, functional input editor. You can further customize the control to your application.

To set up your project and add C1MaskedTextBox controls to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="MaskedTextBox" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}" FontSize="64"/>
</StackPanel>
```

8. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
9. Add the following XAML markup cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a **StackPanel** to the application:

```
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
</StackPanel>
```

   Elements in the panel will appear centered and vertically positioned.
10. In the XAML window of the project, place the cursor between the `<StackPanel x:Name="sp1">` and `</StackPanel>` tags.
11. Add the following markup between the `<StackPanel x:Name="sp1">` and `</StackPanel>` tags:

```
<TextBlock Margin="2,2,2,10" Name="tb1" Text="Employee Information" />
<TextBlock FontSize="16" Margin="2,2,2,0" Text="Employee ID" />
<c1:C1MaskedTextBox Name="c1mtb1" VerticalAlignment="Top" Margin="2"
Mask="000-00-0000"
TextChanged="c1mtb1_TextChanged"></c1:C1MaskedTextBox>
<TextBlock x:Name="tb2" FontSize="16" Margin="2" />
```

```
<TextBlock FontSize="16" Margin="2,2,2,0" Text="Name"/>
<c1:C1MaskedTextBox Name="c1mtb2" VerticalAlignment="Top" Margin="2"
TextChanged="c1mtb2_TextChanged"></c1:C1MaskedTextBox>
<TextBlock x:Name="tb3" FontSize="16" Margin="2"/>
<TextBlock FontSize="16" Margin="2"  Text="Hire Date"/>
<c1:C1MaskedTextBox Name="c1mtb3" VerticalAlignment="Top" Margin="2"
Mask="00/00/0000"
TextChanged="c1mtb3_TextChanged"></c1:C1MaskedTextBox>
<TextBlock x:Name="tb4" FontSize="16" Margin="2"/>
<TextBlock FontSize="16" Margin="2,2,2,0" Text="Phone Number"/>
<c1:C1MaskedTextBox Name="c1mtb4" VerticalAlignment="Top" Margin="2"
Mask="(999) 000-0000"
TextChanged="c1mtb4_TextChanged"></c1:C1MaskedTextBox>
<TextBlock x:Name="tb5" FontSize="16" Margin="2"/>
```

You added several **TextBlock** controls and four **C1MaskedTextBox** controls to the application. Note that you set several options in each **C1MaskedTextBox** control: you added a **Name** to give each control a unique identifier to access the control in code, you set the **Margin**, you added a Mask to control what users can enter into the control, and added the **TextChanged** event handler. Note that you will add the event handler code later in the tutorial.

You've successfully set up your application's user interface. In the next step you'll add code to your application.

Step 2 of 3: Adding Code to the Application

In the previous step you set up the application's user interface and added controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. Select **View | Code** to switch to Code view.
2. In Code view, add the following import statement to the top of the page:
   - Visual Basic
     ```
     Imports C1.Phone
     ```
   - C#
     ```
     using C1.Phone;
     ```
3. Add the following **C1MaskedTextBox_TextChanged** event handlers to the project:
   - Visual Basic
     ```
     Private Sub c1mtb1_TextChanged(ByVal sender As System.Object, ByVal e
     As System.Windows.Controls.TextChangedEventArgs) Handles
     c1mtb1.TextChanged
         Me.lbl2.Content = "Mask: " & Me.c1mtb1.Mask & "  Value: " &
     Me.c1mtb1.Value & "  Text: " & Me.c1mtb1.Text
     End Sub
     Private Sub c1mtb2_TextChanged(ByVal sender As System.Object, ByVal e
     As System.Windows.Controls.TextChangedEventArgs) Handles
     c1mtb2.TextChanged
         Me.lbl3.Content = "Mask: " & Me.c1mtb2.Mask & "  Value: " &
     Me.c1mtb2.Value & "  Text: " & Me.c1mtb2.Text
     End Sub
     Private Sub c1mtb3_TextChanged(ByVal sender As System.Object, ByVal e
     As System.Windows.Controls.TextChangedEventArgs) Handles
     c1mtb3.TextChanged
         Me.lbl4.Content = "Mask: " & Me.c1mtb3.Mask & "  Value: " &
     Me.c1mtb3.Value & "  Text: " & Me.c1mtb3.Text
     End Sub
     ```

```
Private Sub c1mtb4_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
c1mtb4.TextChanged
    Me.lbl5.Content = "Mask: " & Me.c1mtb4.Mask & "  Value: " &
Me.c1mtb4.Value & "  Text: " & Me.c1mtb4.Text
End Sub
```

- C#

```
private void c1mtb1_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl2.Content = "Mask: " + this.c1mtb1.Mask + " Value: " +
this.c1mtb1.Value + " Text: " + this.c1mtb1.Text;
}
private void c1mtb2_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl3.Content = "Mask: " + this.c1mtb2.Mask + " Value: " +
this.c1mtb2.Value + " Text: " + this.c1mtb2.Text;
}
private void c1mtb3_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl4.Content = "Mask: " + this.c1mtb3.Mask + " Value: " +
this.c1mtb3.Value + " Text: " + this.c1mtb3.Text;
}
private void c1mtb4_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl5.Content = "Mask: " + this.c1mtb4.Mask + " Value: " +
this.c1mtb4.Value + " Text: " + this.c1mtb4.Text;
}
```

4. Add code to the page's constructor so that it appears like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Me.c1mtb1_TextChanged(Nothing, Nothing)
    Me.c1mtb2_TextChanged(Nothing, Nothing)
    Me.c1mtb3_TextChanged(Nothing, Nothing)
    Me.c1mtb4_TextChanged(Nothing, Nothing)
End Sub
```

- C#

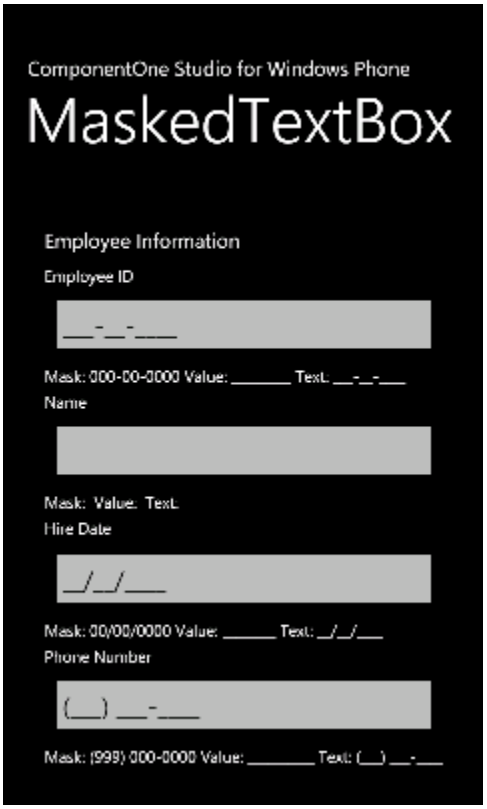
```
public Page()
{
    InitializeComponent();
    this.c1mtb1_TextChanged(null, null);
    this.c1mtb2_TextChanged(null, null);
    this.c1mtb3_TextChanged(null, null);
    this.c1mtb4_TextChanged(null, null);
}
```

In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the Application

Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **MaskedTextBox for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:



2. Enter a number in the first **C1MaskedTextBox** control.

   The label below the control displays the mask, current value, and current text.

3. Enter a string in the second **C1MaskedTextBox** control.

   Notice that there was no mask added to this control – if you chose, you could type numbers or other characters in the control.

4. Try entering a string in the third **C1MaskedTextBox** control. Notice that you cannot – that is because the Mask property was set to only accept numbers. Enter a numeric value instead – notice that this does work.

5. Enter numbers in each of the remaining control. The application will appear similar to the following image.

   Notice that the Value property displayed under each C1MaskedTextBox control does not include literal characters, while the **Text** property does.

Congratulations! You've completed the **MaskedTextBox for Windows Phone** quick start and created a **MaskedTextBox for Windows Phone** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Working With MaskedTextBox

ComponentOne MaskedTextBox for Windows Phone includes the C1MaskedTextBox control, a simple control which provides a text box with a mask that automatically validates entered input. When you add the

C1MaskedTextBox control to a XAML window, it exists as a completely functional text box which you can further customize with a mask.

The C1MaskedTextBox control appears like a text box and includes a basic text input area which can be customized.

Basic Properties

ComponentOne MaskedTextBox for Windows Phone includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see MaskedTextBox for Windows Phone Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1MaskedTextBox control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Text	Gets or sets the text content of this element.
TextMaskFormat	Gets or sets a value that determines whether literals and prompt characters are included in the Value property.
Value	Gets the formatted content of the control as specified by the TextMaskFormat property.
Watermark	Gets or sets the content of the watermark.

The **Text** property of the C1MaskedTextBox exposes the control's full content. The Value property exposes only the values typed by the user, excluding template characters specified in the Mask. For example, if the Mask property is set to "99-99" and the control contains the string "55-55", the **Text** property would return "55-55" and the Value property would return "5555".

Mask Formatting

You can provide input validation and format how the content displayed in the C1MaskedTextBox control will appear by setting the Mask property. **ComponentOne MaskedTextBox for Windows Phone** supports the standard number formatting strings defined by Microsoft and the Mask property uses the same syntax as the standard **MaskedTextBox** control in WinForms. This makes it easier to re-use masks across applications and platforms.

By default, the Mask property is not set and no input mask is applied. When a mask is applied, the Mask string should consist of one or more of the masking elements. Other elements that may be displayed in the control are literals and prompts which may also be used if allowed by the TextMaskFormat property.

The following table lists some example masks:

Mask	Behavior
00/00/0000	A date (day, numeric month, year) in international date format. The "/" character is a logical date separator, and will appear to the user as the date separator appropriate to the application's current culture.
00->L<LL-0000	A date (day, month abbreviation, and year) in United States format in which the three-letter month abbreviation is displayed with an initial uppercase letter followed by two lowercase letters.
(999)-000-	United States phone number, area code optional. If users do not want to enter the optional

0000	characters, they can either enter spaces or place the mouse pointer directly at the position in the mask represented by the first 0.
$999,999.00	A currency value in the range of 0 to 999999. The currency, thousandth, and decimal characters will be replaced at run time with their culture-specific equivalents.

You can set the TextMaskFormat property to one of the following elements to define what is included in the mask:

Option	Description
IncludePrompt	Return text input by the user as well as any instances of the prompt character.
IncludeLiterals	Return text input by the user as well as any literal characters defined in the mask.
IncludePromptAndLiterals	Return text input by the user as well as any literal characters defined in the mask and any instances of the prompt character.
ExcludePromptAndLiterals	Return only text input by the user.

The following topics detail mask, literal, and prompt elements that can be used or displayed.

Mask Elements

ComponentOne MaskedTextBox for Windows Phone supports the standard number formatting strings defined by Microsoft. The Mask string should consist of one or more of the masking elements as detailed in the following table:

Element	Description
0	Digit, required. This element will accept any single digit between 0 and 9.
9	Digit or space, optional.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the **Text** property. Plus (+) and minus (-) signs are allowed.
L	Letter, required. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z] in regular expressions.
?	Letter, optional. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z]? in regular expressions.
&	Character, required.
C	Character, optional. Any non-control character.
A	Alphanumeric, optional.
a	Alphanumeric, optional.
.	Decimal placeholder. The actual display character used will be the decimal symbol appropriate to the format provider.
,	Thousands placeholder. The actual display character used will be the thousands placeholder appropriate to the format provider.
:	Time separator. The actual display character used will be the time symbol appropriate to the format provider.

/	Date separator. The actual display character used will be the date symbol appropriate to the format provider.
$	Currency symbol. The actual character displayed will be the currency symbol appropriate to the format provider.
<	Shift down. Converts all characters that follow to lowercase.
>	Shift up. Converts all characters that follow to uppercase.
|	Disable a previous shift up or shift down.
\	Escape. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.
All other characters	Literals. All non-mask elements will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

The decimal (.), thousandths (,), time (:), date (/), and currency ($) symbols default to displaying those symbols as defined by the application's culture.

Literals

In addition to the mask elements defined in the Mask Formatting topic, other characters can be included in the mask. These characters are *literals*. Literals are non-mask elements that will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

For example, if the Mask property has been set to "(999)-000-0000" to define a phone number, the mask characters include the "9" and "0" elements. The remaining characters, the dashes and parentheses, are literals. These characters will appear as they in the C1MaskedTextBox control.

Note that the TextMaskFormat property must be set to **IncludeLiterals** or **IncludePromptAndLiterals** for literals to be used. If you do not want literals to be used, set TextMaskFormat to **IncludePrompt** or **ExcludePromptAndLiterals**.

Prompts

You can choose to include prompt characters in the **C1MaskedTextBox** control. The prompt character defined that text that will appear in the control to prompt the user to enter text. The prompt character indicates to the user that text can be entered, and can be used to detail the type of text allowed. By default the underline "_" character is used.

Note that the TextMaskFormat property must be set to **IncludePrompt** or **IncludePromptAndLiterals** for prompt characters to be used. If you do not want prompt characters to be used, set TextMaskFormat to **IncludeLiterals** or **ExcludePromptAndLiterals**.

Watermark

Using the Watermark property you can provide contextual clues of what value users should enter in a C1MaskedTextBox control. The watermark is displayed in the control while not text has been entered. To add a watermark, add the text `Watermark="Watermark Text"` to the `<c1:C1MaskedTextBox>` tag in the XAML markup for any **C1MaskedTextBox** control.

So, for example, enter `Watermark="Enter Text"` to the `<c1:C1MaskedTextBox>` tag so that appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Watermark="Enter Text" />
```

If you click within the control and enter text, you will notice that the watermark disappears.

MaskedTextBox for Windows Phone Layout and Appearance

The following topics detail how to customize the C1MaskedTextBox control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Windows Phone's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

MaskedTextBox for Windows Phone Appearance Properties

ComponentOne MaskedTextBox for Windows Phone includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Content Properties

The following properties let you customize the appearance of content in the **C1MaskedTextBox** control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Watermark	Gets or sets the content of the watermark.

Text Properties

The following properties let you customize the appearance of text in the **C1MaskedTextBox** control:

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
TextAlignment	Gets or sets how the text should be aligned in the **C1MaskedTextBox**.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the

	background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1MaskedTextBox** control:

Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

MaskedTextBox for Windows Phone Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1MaskedTextBox control in general. If you are unfamiliar with the **ComponentOne MaskedTextBox for Windows Phone** product, please see the MaskedTextBox for Windows Phone Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne MaskedTextBox for Windows Phone** product.

Each task-based help topic also assumes that you have created a new Windows Phone project.

Setting the Value

The Value property determines the currently visible text. By default the C1MaskedTextBox control starts with its Value not set but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To set the Value property in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter a number, for example "123", in the text box next to the Value property.

   This will set the Value property to the number you chose.

In XAML

To set the Value property add `Value="123"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Value="123"></c1:C1MaskedTextBox>
```

In Code

To set the Value property, add the following code to your project:

- Visual Basic
  ```
  C1MaskedTextBox1.Value = "123"
  ```
- C#
  ```
  c1MaskedTextBox1.Value = "123";
  ```

Run your project and observe:

Initially **123** (or the number you chose) will appear in the control.

Adding a Mask for Currency

You can easily add a mask for currency values using the Mask property. By default the C1MaskedTextBox control starts with its Mask not set but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code. For more details about mask characters, see Mask Elements.

At Design Time in Blend

To set the Mask property in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter "$999,999.00" in the text box next to the Mask property.

   This will set the Mask property to the number you chose.

In XAML

To set the Mask property add `Mask="$999,999.00"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Mask="$999,999.00"></c1:C1MaskedTextBox>
```

In Code

To set the Value property add the following code to your project:

- Visual Basic
  ```
  C1MaskedTextBox1.Mask = "$999,999.00"
  ```
- C#
  ```
  c1MaskedTextBox1.Mask = "$999,999.00";
  ```

Run your project and observe:

The mask will appear in the control. Enter a number; notice that the mask is filled.

Changing the Prompt Character

The PromptChar property sets the characters that are used to prompt users in the C1MaskedTextBox control. By default the PromptChar property is set to an underline character ("_") but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code. For more details about the PromptChar property, see Prompts.

At Design Time in Blend

To set the PromptChar property at in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter "0000" in the text box next to the Mask property to set a mask.
3. In the properties window, enter "#" (the pound character) in the text box next to the PromptChar property

In XAML

Tto set the PromptChar property add `Mask="0000" PromptChar="#"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
Mask="0000" PromptChar="#"></c1:C1MaskedTextBox>
```

In Code

To set the PromptChar property add the following code to your project:

- Visual Basic

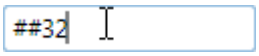
```
Dim x As Char = "#"c
C1MaskedTextBox1.Mask = "0000"
C1MaskedTextBox1.PromptChar = x
```

- C#

```
char x = '#';
this.c1MaskedTextBox1.Mask = "0000";
this.c1MaskedTextBox1.PromptChar = x;
```

Run your project and observe:

The pound character will appear as the prompt in the control. In the following image, the number 32 was entered in the control:



Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in Microsoft Expression Blend's Properties tab, through XAML, or through code.

At Design Time in Blend

To change the font of the grid to Arial 10pt in Blend, complete the following:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab, and set **FontFamily** property to "Arial" (or a font of your choice).
3. In the Properties window, set the **FontSize** property to **10**.

   This will set the control's font size and style.

In XAML

For example, to change the font of the control to Arial 10pt in XAML add `FontFamily="Arial" FontSize="10"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
FontSize="10" FontFamily="Arial"></c1:C1MaskedTextBox>
```

In Code

For example, to change the font of the grid to Arial 10pt add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.FontSize = 10
C1MaskedTextBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

- C#

```
c1MaskedTextBox1.FontSize = 10;
c1MaskedTextBox1.FontFamily = new
System.Windows.Media.FontFamily("Arial");
```

Run your project and observe:

The control's content will appear in Arial 10pt font.

Locking the Control from Editing

By default the C1MaskedTextBox control's Value property is editable by users at run time. If you want to lock the control from being edited, you can set the **IsReadOnly** property to **True**.

At Design Time in Blend

To lock the C1MaskedTextBox control from run-time editing, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and check the **IsReadOnly** check box.

   This will set the **IsReadOnly** property to **False**.

In XAML

To lock the C1MaskedTextBox control from run-time editing in XAML add `IsReadOnly="True"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
IsReadOnly="True"></c1:C1MaskedTextBox>
```

In Code

To lock the C1MaskedTextBox control from run-time editing add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.IsReadOnly = True
```

- C#

```
c1MaskedTextBox1.IsReadOnly = true;
```

Run your project and observe:

The control is locked from editing. Try to click the cursor within the control – notice that the text insertion point (the blinking vertical line) will not appear in the control.

NumericBox

ComponentOne NumericBox™ for Windows Phone provides a text box for displaying and editing formatted numeric values such as currencies and percentages. The control comes complete with smart input and increment buttons.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

NumericBox for Windows Phone Features

ComponentOne NumericBox for Windows Phone allows you to create customized, rich applications. Make the most of **NumericBox for Windows Phone** by taking advantage of the following key features:

- **Flexible Formatting**

  The **Format** property enables you to use the familiar .NET format strings to display numbers in several formats with support for decimal places. Supported formats include fixed-point (F), number (N), general (G), currency (C), exponential (E), hexadecimal (X) and percent (P).

- **Numeric Range Support**

  Restrict input to a specific numeric range by setting the **Minimum** and **Maximum** properties.

- **Increment/Decrement Buttons**

  C1NumericBox includes increment and decrement buttons to allow simple editing without requiring the keyboard.

- **Metro UI Design**

  C1NumericBox supports the Metro UI design and interaction guidelines as specified by Microsoft.

NumericBox for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **NumericBox for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add **NumericBox for Windows Phone** controls to your application, and customize the appearance and behavior of the controls.

You will create an application that includes five C1NumericBox controls. The controls will function as a lock and when the correct code number has been entered in each, the controls will become locked and inactive and a button will appear directing users to a Web site.

Step 1 of 3: Adding NumericBox for Windows Phone to your Project

In this step you'll begin in Visual Studio to create a Windows Phone application using **NumericBox for Windows Phone**. When you add a C1NumericBox control to your application, you'll have a complete, functional numeric editor. You can further customize the control to your application.

To set up your project and add a C1NumericBox control to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.

2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage x:Class="C1WP.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="NumericBox" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

8. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
9. Add the following XAML markup cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a **StackPanel** containing a **TextBlock** to the application:

```
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
<TextBlock Height="30" Name="tb1" Text="Enter Combination:" />
</StackPanel>
```

10. Add the following markup between the `<TextBlock />` and `</StackPanel>` tags to add five C1NumericBox controls to your application:

```
<c1:C1NumericBox x:Name="c1nb1" Width="190" Height="70" Minimum="0"
Maximum="9" ValueChanged="c1nb1_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb2" Width="190" Height="70" Minimum="0"
Maximum="9" ValueChanged="c1nb2_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb3" Width="190" Height="70" Minimum="0"
Maximum="9" ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb4" Width="190" Height="70" Minimum="0"
Maximum="9" ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb5" Width="190" Height="70" Minimum="0"
Maximum="9" ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
```

Note that in each of the **C1NumericBox** controls you set the following:

- **Name**: Adds a unique identifier that allows you to access the control in code.
- **Height**: Sets the vertical size of the control.
- **Width**: Sets the horizontal size of the control.
- **Minimum**: Sets the minimum valued that can be entered in the **C1NumericBox** control. Users will not be able to enter values below the minimum providing built-in data validation.
- **Maximum**: Sets the maximum valued that can be entered in the **C1NumericBox** control. Users will not be able to enter values above the maximum providing built-in data validation.
- **ValueChanged**: Sets the event handler that handles when the value of the **C1NumericBox** is changed. You'll add the code for this event handler in a later step.

11. Add the following markup between the last `</c1:C1NumericBox>` tag and the `</StackPanel>` tag to add a **TextBlock** and **Button** to the application:

```
<TextBlock Height="30" Name="tb2" Text="Invalid Combination."
Foreground="Red" />
<Button x:Name="btn1" Content="Enter" Visibility="Collapsed"
Height="70" Margin="2" Click="btn1_Click"></Button>
```

Your complete markup will appear like the following:

- XAML Markup

```
<phone: <phone:PhoneApplicationPage x:Class="C1WP.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone" mc:Ignorable="d"
d:DesignWidth="480" d:DesignHeight="768" FontFamily="{StaticResource
PhoneFontFamilyNormal}" FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page
title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="ComponentOne
Studio for Windows Phone" Style="{StaticResource
PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="NumericBox" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>
```

```
        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
                    <TextBlock Height="30" Name="tb1" Text="Enter
Combination:" />
                <c1:C1NumericBox x:Name="c1nb1" Width="190" Height="70"
Minimum="0" Maximum="9"
ValueChanged="c1nb1_ValueChanged"></c1:C1NumericBox>
                <c1:C1NumericBox x:Name="c1nb2" Width="190" Height="70"
Minimum="0" Maximum="9" ValueChanged="c1nb2_ValueChanged"
></c1:C1NumericBox>
                <c1:C1NumericBox x:Name="c1nb3" Width="190" Height="70"
Minimum="0" Maximum="9"
ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
                <c1:C1NumericBox x:Name="c1nb4" Width="190" Height="70"
Minimum="0" Maximum="9"
ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>
                <c1:C1NumericBox x:Name="c1nb5" Width="190" Height="70"
Minimum="0" Maximum="9"
ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
                    <TextBlock Height="30" Name="tb2" Text="Invalid
Combination." Foreground="Red" />
                    <Button x:Name="btn1" Content="Enter"
Visibility="Collapsed" Height="70" Margin="2"
Click="btn1_Click"></Button>
            </StackPanel>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```

You've successfully created a Windows Phone application, added **C1NumericBox** controls to the application, and customized those controls. In the next step you'll complete setting up the application.

Step 2 of 3: Adding Code to the Application

In the previous steps you set up the application's user interface and added five **C1NumericBox** controls to the application. In this step you'll add code to your application to finalize the application.

Complete the following steps:

1. Select **View | Code** to switch to Code view.
2. Add the following imports statements to the top of the page:
   - Visual Basic
     ```
     Imports C1.Phone
     Imports System.Windows.Media
     Imports Microsoft.Phone.Tasks
     ```
   - C#
     ```
     using C1.Phone;
     using System.Windows.Media;
     using Microsoft.Phone.Tasks;
     ```
3. Initialize the following global variables just inside the **MainPage** class:

- Visual Basic

```
Dim nb1 As Integer = 5
Dim nb2 As Integer = 2
Dim nb3 As Integer = 3
Dim nb4 As Integer = 7
Dim nb5 As Integer = 9
```

- C#

```
int nb1 = 5;
int nb2 = 2;
int nb3 = 3;
int nb4 = 7;
int nb5 = 9;
```

These numbers will be used as the correct 'code' in the application. When the user enters the correct combination of numbers at run time the button will appear.

4. Add the following **Click** event handler to the code:

- Visual Basic

```
Private Sub btn1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles btn1.Click
    Dim task As New WebBrowserTask()
    task.Uri = New Uri("http://www.componentone.com/")
    task.Show()
End Sub
```

- C#

```
private void btn1_Click(object sender, RoutedEventArgs e)
{
    WebBrowserTask task = new WebBrowserTask();
    task.Uri = new Uri("http://www.componentone.com/");
    task.Show();
}
```

When the button is pressed at run time it will open the ComponentOne Web site.

5. Next add the following custom **NBValidation** event to your code:

- Visual Basic

```
Private Sub NBValidation()
    If Me.c1nb1.Value = nb1 And Me.c1nb2.Value = nb2 And Me.c1nb3.Value
= nb3 And Me.c1nb4.Value = nb4 And Me.c1nb5.Value = nb5 Then
        Me.tb2.Foreground = New SolidColorBrush(Colors.Green)
        Me.tb2.Text = "Combination Valid"
        Me.c1nb1.IsReadOnly = True
        Me.c1nb2.IsReadOnly = True
        Me.c1nb3.IsReadOnly = True
        Me.c1nb4.IsReadOnly = True
        Me.c1nb5.IsReadOnly = True
        Me.btn1.Visibility = Windows.Visibility.Visible
    End If
End Sub
```

- C#

```
private void NBValidation()
{
    if (this.c1nb1.Value == nb1 & this.c1nb2.Value == nb2 &
this.c1nb3.Value == nb3 & this.c1nb4.Value == nb4 & this.c1nb5.Value ==
nb5)
```

```
    {
        this.tb2.Foreground = new SolidColorBrush(Colors.Green);
        this.tb2.Text = "Combination Valid";
        this.c1nb1.IsReadOnly = true;
        this.c1nb2.IsReadOnly = true;
        this.c1nb3.IsReadOnly = true;
        this.c1nb4.IsReadOnly = true;
        this.c1nb5.IsReadOnly = true;
        this.btn1.Visibility = Windows.Visibility.Visible;
    }
}
```

When the user enters the correct numbers (as indicated in step 3 above) the C1NumericBox controls will be set to read only and will no longer be editable, the text of the label below the controls will change to indicate the correct code has been entered, and a button will appear allowing users to enter the ComponentOne Web site.

6. Add **C1NumericBox_ValueChanged** event handlers to initialize **NBValidation**. The code will look like the following:

- Visual Basic

```
Private Sub c1nb1_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Phone.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb1.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb2_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Phone.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb2.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb3_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Phone.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb3.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb4_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Phone.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb4.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb5_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Phone.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb5.ValueChanged
    NBValidation()
End Sub
```

- C#

```
private void c1nb1_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb2_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
```

```
private void c1nb3_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb4_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb5_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
```

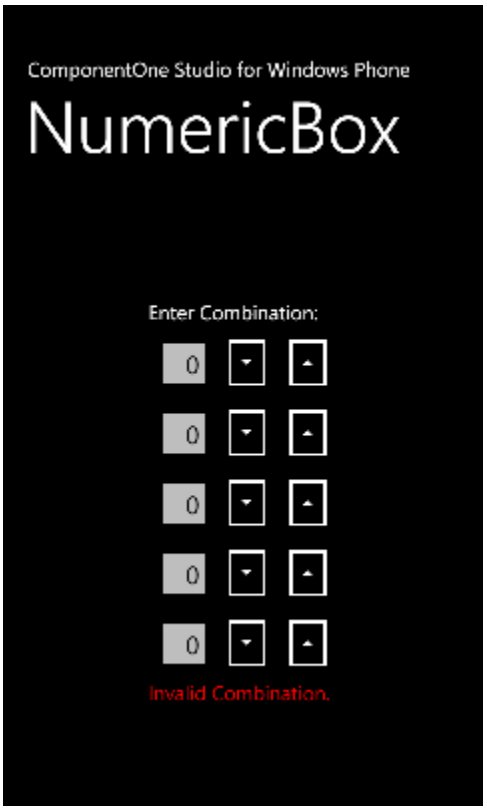
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the Application

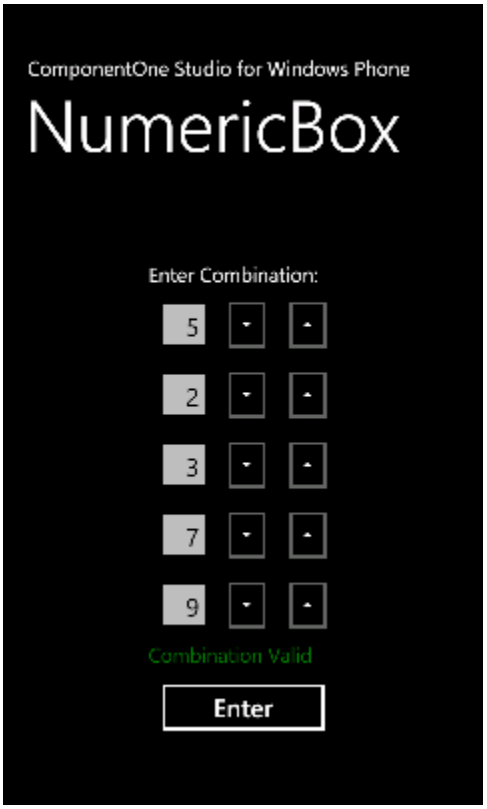
Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **NumericBox for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging**.

   The application will appear in the Windows Phone emulator and will appear similar to the following:



1. Select the **Up** arrow in the first (top-most) **C1NumericBox** control until **5** is displayed. Note that the number increased by 1 each time you select the arrow – this is because the Increment property is set to **1** by default.
2. Select the text portion of the second **C1NumericBox** and in the keyboard that appears, select **2**.
3. Try selecting the **Down** button in the third **C1NumericBox** control and notice that the number does not change. This is because the Minimum property was set to **0** and so the control will not accept values less than zero. Select the **Up** button until **3** is displayed.
4. Select the text portion of the fourth **C1NumericBox** and in the keyboard that appears, select **4**.
5. Select the **Up** arrow in the last (bottom-most) **C1NumericBox** control until **9** is displayed.
6. Select the **Down** button of the fourth **C1NumericBox** control twice so **7** is displayed. Note that the text of the second **TextBlock** changed and the button is now visible:

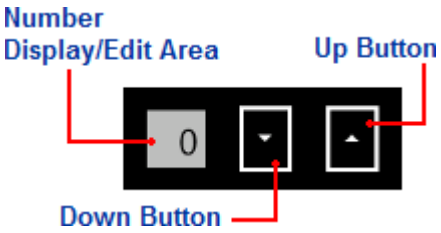


7. Try Selecting a **C1NumericBox** control or selecting its **Up** or **Down** buttons and notice that you cannot. That is because the IsReadOnly property was set to **True** when the correct number sequence was entered and the controls are now locked from editing.
8. Select the now-visible **Enter** button to navigate to the ComponentOne Web site.

Congratulations! You've completed the **NumericBox for Windows Phone** quick start and created a **NumericBox for Windows Phone** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

About C1NumericBox

ComponentOne NumericBox for Windows Phone includes the C1NumericBox control, a simple control which provides numeric input and editing. When you add the C1NumericBox control to a XAML window, it exists as a completely functional numeric editor. By default, the control's interface looks similar to the following image:



It consists of the following elements:

- **Up and Down Buttons**

  The **Up** and **Down** buttons allow users to change the value displayed in the control. Each time a button is pressed the Value changes by the amount indicated by the Increment property (by default 1). By default the **Up** and **Down** buttons are visible; to hide the buttons set the ShowButtons property to **False**.

- **Number Display/Edit Area**

  The current Value is displayed in the number display/editing area. Users can type in the box to change the Value property. By default users can edit this number; to lock the control from editing set IsReadOnly to **True**.

Basic Properties

ComponentOne NumericBox for Windows Phone includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see NumericBox for Windows Phone Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1NumericBox control:

Property	Description
Value	Gets or sets the numeric value in the C1NumericBox.
Minimum	Gets or sets the minimum value allowed for the C1NumericBox control.
Maximum	Gets or sets the maximum value allowed for the C1NumericBox.
Increment	Gets or sets the increment applied when the user presses the up/down arrow keys or the **Up** or **Down** buttons.
Format	Gets or sets the format of the C1NumericBox control.

Number Formatting

You can change how the number displayed in the C1NumericBox control will appear by setting the Format property. **ComponentOne NumericBox for Windows Phone** supports the standard number formatting strings defined by Microsoft. For more information, see MSDN.

The Format string consists of a letter or a letter and number combination defining the format. By default, the Format property is set to "F0". The letter indicates the format type, here "F" for fixed-point, and the number indicates the number of decimal places, here none.

The following formats are available:

Format Specifier	Name	Description
C or c	Currency	The number is converted to a string that represents a currency amount. The conversion is controlled by the currency format information of the current NumberFormatInfo object. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default currency precision given by the current NumberFormatInfo object is used.
D or d	Decimal	This format is supported only for integral types. The number is converted to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. The following example formats an Int32 value with the Decimal format specifier.
E or e	Scientific (exponential)	The number is converted to a string of the form "-d.ddd…E+ddd" or "-d.ddd…e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used. The case of the format specifier indicates whether to prefix the exponent with an 'E' or an 'e'. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
F or f	Fixed-point	The number is converted to a string of the form "-ddd.ddd…" where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.
G or g	General	The number is converted to the most compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. If the precision specifier is omitted or zero, the type of the

		number determines the default precision, as indicated by the following list. • Byte or SByte: 3 • Int16 or UInt16: 5 • Int32 or UInt32: 10 • Int64: 19 • UInt64: 20 • Single: 7 • Double: 15 • Decimal: 29 Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required and trailing zeroes are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, then the excess trailing digits are removed by rounding. The exception to the preceding rule is if the number is a Decimal and the precision specifier is omitted. In that case, fixed-point notation is always used and trailing zeroes are preserved. If scientific notation is used, the exponent in the result is prefixed with 'E' if the format specifier is 'G', or 'e' if the format specifier is 'g'. The exponent contains a minimum of two digits. This differs from the format for scientific notation produced by the 'E' or 'e' format specifier, which includes a minimum of three digits in the exponent.
N or n	Number	The number is converted to a string of the form "-d,ddd,ddd.ddd…", where '-' indicates a negative number symbol if required, 'd' indicates a digit (0-9), ',' indicates a thousand separator between number groups, and '.' indicates a decimal point symbol. The actual negative number pattern, number group size, thousand separator, and decimal separator are specified by the NumberNegativePattern, NumberGroupSizes, NumberGroupSeparator, and NumberDecimalSeparator properties, respectively, of the current NumberFormatInfo object. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.
P or p	Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property if the number is negative, or the NumberFormatInfo.PercentPositivePattern property if the number is positive. The converted number is multiplied by 100 in order to be presented as a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the current

		NumberFormatInfo object is used.
R or r	Round-trip	This format is supported only for the Single and Double types. The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same numeric value. When a numeric value is formatted using this specifier, it is first tested using the general format, with 15 spaces of precision for a Double and 7 spaces of precision for a Single. If the value is successfully parsed back to the same numeric value, it is formatted using the general format specifier. However, if the value is not successfully parsed back to the same numeric value, then the value is formatted using 17 digits of precision for a Double and 9 digits of precision for a Single. Although a precision specifier can be present, it is ignored. Round trips are given precedence over precision when using this specifier.
X or x	Hexadecimal	This format is supported only for integral types. The number is converted to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for the hexadecimal digits greater than 9. For example, use 'X' to produce "ABCDEF", and 'x' to produce "abcdef". The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.
Any other single character	(Unknown specifier)	(An unknown specifier throws a FormatException at runtime.)

Input Validation

You can use the Minimum and Maximum properties to set a numeric range that users are limited to at run time. If the Minimum and Maximum properties are set, users will not be able to pick a number larger than the Minimum or smaller than the Maximum.

When setting the Minimum and Maximum properties, the Minimum should be smaller than the Maximum. Also be sure to set the Value property to a number within the Minimum and Maximum range.

You can also choose a mode for range validation using the RangeValidationMode property. This property controls when the entered number is validated. You can set RangeValidationMode to one of the following options:

Option	Description
Always	This mode does not allow users to enter out of range values.
AlwaysTruncate	This mode does not allow users to enter out of range values. The value will be truncated if the limits are exceeded.
OnLostFocus	This mode truncates the value when the control loses focus.

NumericBox for Windows Phone Layout and Appearance

The following topics detail how to customize the C1NumericBox control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to

customize the appearance of the grid and take advantage of Windows Phone's XAML-based styling. You can also use templates to format and layout the grid and to customize grid actions.

NumericBox for Windows Phone Appearance Properties

ComponentOne NumericBox for Windows Phone includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Content Properties

The following properties let you customize the appearance of content in the **C1NumericBox** control:

Property	Description
Format	Gets or sets the value for the Format of the C1NumericBox.
Watermark	Gets or sets the watermark content displayed when the control is empty.

Text Properties

The following properties let you customize the appearance of text in the **C1NumericBox** control:

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
TextAlignment	Gets or sets how the text should be aligned in the **C1NumericBox**.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.
SelectionBackground	Gets or sets the brush that fills the background of the selected text.

SelectionForeground	Gets or sets the brush used for the selected text in the C1NumericBox.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Style Properties

The following properties let you set styles:

Property	Description
FocusVisualStyle	Gets or sets a property that enables customization of appearance, effects, or other style characteristics that will apply to this element when it captures keyboard focus. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1NumericBox** control:

Property	Description
ActualHeight	Gets the rendered height of this element. This is a dependency property.
ActualWidth	Gets the rendered width of this element. This is a dependency property.
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

NumericBox for Windows Phone Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1NumericBox control in general. If you are unfamiliar with the **ComponentOne NumericBox for Windows Phone** product, please see the NumericBox for Windows Phone Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne NumericBox for Windows Phone** product.

Each task-based help topic also assumes that you have created a new Windows Phone project.

Setting the Start Value

The Value property determines the currently selected number. By default the C1NumericBox control starts with its Value set to **0** but you can customize this number at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time

To set the Value property in Blend or Visual Studio, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "123", in the text box next to the Value property.

   This will set the Value property to the number you chose.

In XAML

For example, to set the Value property add `Value="123"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Value="123"></c1:C1NumericBox>
```

In Code

For example, to set the Value property add the following code to your project:

- Visual Basic

```
C1NumericBox1.Value = 123
```

- C#

```
c1NumericBox1.Value = 123;
```

Run your project and observe:

Initially **123** (or the number you chose) will appear in the control.

Setting the Increment Value

The Increment property determines by how much the Value property changes when the **Up** or **Down** button is pressed at run time. By default the C1NumericBox control starts with its Increment set to **1** but you can customize this number at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time

To set the Increment property in Blend or Visual Studio, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "20", in the text box next to the Increment property.

   This will set the Increment property to the number you chose.

In XAML

For example, to set the Increment property to **20** add `Increment="20"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Increment="20"></c1:C1NumericBox>
```

In Code

For example, to set the Increment property to **20** add the following code to your project:

- Visual Basic

```
C1NumericBox1.Increment = 20
```

- C#

```
c1NumericBox1.Increment = 20;
```

Run your project and observe:

Press the **Up** and then the **Down** button a few times. Notice that the Value changes in steps of 20. You can still edit the value directly by selecting the text box and entering a number that falls between that step.

Setting the Minimum and Maximum Values

You can use the Minimum and Maximum properties to set a numeric range that users are limited to at run time. If the Minimum and Maximum properties are set, users will not be able to pick a number larger than the Minimum or smaller than the Maximum.

> **Note:** When setting the Minimum and Maximum properties, the Minimum should be smaller than the Maximum. Also be sure to set the Value property to a number within the Minimum and Maximum range. In the following example, the default value **0** falls within the range chosen.

At Design Time

To set the Minimum and Maximum in Blend or Visual Studio, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example **500**, next to the Maximum property.
3. In the Properties tab, enter a number, for example **-500**, next to the Minimum property.

   This will set Minimum and Maximum values.

In XAML

To set the Minimum and Maximum in XAML add `Maximum="500" Minimum="-500"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Maximum="500" Minimum="-
500"></c1:C1NumericBox>
```

In Code

To set the Minimum and Maximum add the following code to your project:

- Visual Basic

```
C1NumericBox1.Minimum = -500
C1NumericBox1.Maximum = 500
```

- C#

```
c1NumericBox1.Minimum = -500;
c1NumericBox1.Maximum = 500;
```

Run your project and observe:

Users will be limited to the selected range at run time.

Hiding the Up and Down Buttons

By default buttons are visible in the C1NumericBox control to allow users to increment and decrement the value in the box by one step. You can choose to hide the **Up** and **Down** buttons in the C1NumericBox control at run time. To hide the **Up** and **Down** buttons you can set the ShowButtons property to **False**.

At Design Time in Blend

To hide the **Up** and **Down** buttons in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and uncheck the ShowButtons check box.

   This will set the ShowButtons property to **False**.

In XAML

For example, to hide the **Up** and **Down** buttons in XAML add `ShowButtons="False"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
ShowButtons="False"></c1:C1NumericBox>
```

In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic

```
C1NumericBox1.ShowButtons = False
```

- C#

```
c1NumericBox1.ShowButtons = false;
```

Run your project and observe:

The **Up** and **Down** buttons will not be visible.

Locking the Control from Editing

By default the C1NumericBox control's Value property is editable by users at run time. If you want to lock the control from being edited, you can set the IsReadOnly property to **True**.

At Design Time

To lock the C1NumericBox control from run-time editing in Blend or Visual Studio, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and check the IsReadOnly check box.

   This will set the IsReadOnly property to **False**.

In XAML

For example, to hide the **Up** and **Down** buttons in XAML add `IsReadOnly="True"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1"
IsReadOnly="True"></c1:C1NumericBox>
```

In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic

```
C1NumericBox1.IsReadOnly = True
```

- C#

```
c1NumericBox1.IsReadOnly = true;
```

Run your project and observe:

The control is locked from editing; notice that the **Up** and **Down** buttons are grayed out and inactive.

ProgressBar

Visually indicate progress of a lengthy operation with **ComponentOne ProgressBar™ for Windows Phone**. **C1ProgressBar** displays an animated repeating pattern of dots to indicate an indeterminate operation is in progress.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

ProgressBar for Windows Phone Key Features

ComponentOne ProgressBar for Windows Phone allows you to create customized, rich applications. Make the most of **ProgressBar for Windows Phone** by taking advantage of the following key features:

- **High Performance**

  C1ProgressBar does not block the UI thread from running, so you can show a progress indicator while the background UI is performing some operation.

- **User Friendly Experience**

  C1ProgressBar is similar to the native progress indicators used on the Windows Phone 7 to provide a familiar and mobile-appropriate user experience. Use it to indicate loading of data or some long calculation to the user in an intuitive manner.

ProgressBar for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne ProgressBar for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add the C1ProgressBar control to your application, and observe the control at run time.

Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **ProgressBar for Windows Phone**. Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="ProgressBar" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}" FontSize="64"/>
```

```
</StackPanel>
```

5. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
6. Navigate to the Toolbox and double-click the **C1ProgressBar** icon to add the control to the grid. The XAML markup should resemble the following:

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <my:C1ProgressBar HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1ProgressBar1" VerticalAlignment="Top"/>
        </Grid>
```

You have successfully created a Windows Phone application containing a **C1ProgressBar** control. In the next step, you will customize the control.

Step 2 of 3: Customizing the Control

In the previous step you set up the application's user interface and added a C1ProgressBar control to your application. In this step you will customize the control and add a check box.

1. With the C1ProgressBar control selected on the page, select **View | Properties**.
2. In the Properties window, locate the **Margin** property and enter "3" in the text box next to the property name.
3. Next to the **Height** property, enter "200" and next to the **Width** property enter "450". Your XAML markup should now look like this:

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <my:C1ProgressBar HorizontalAlignment="Left" Margin="3"
Name="C1ProgressBar1" VerticalAlignment="Top" Height="200" Width="450" />
</Grid>
```

4. In Source View edit the markup to add a check box to the application. The markup will now look like the following:

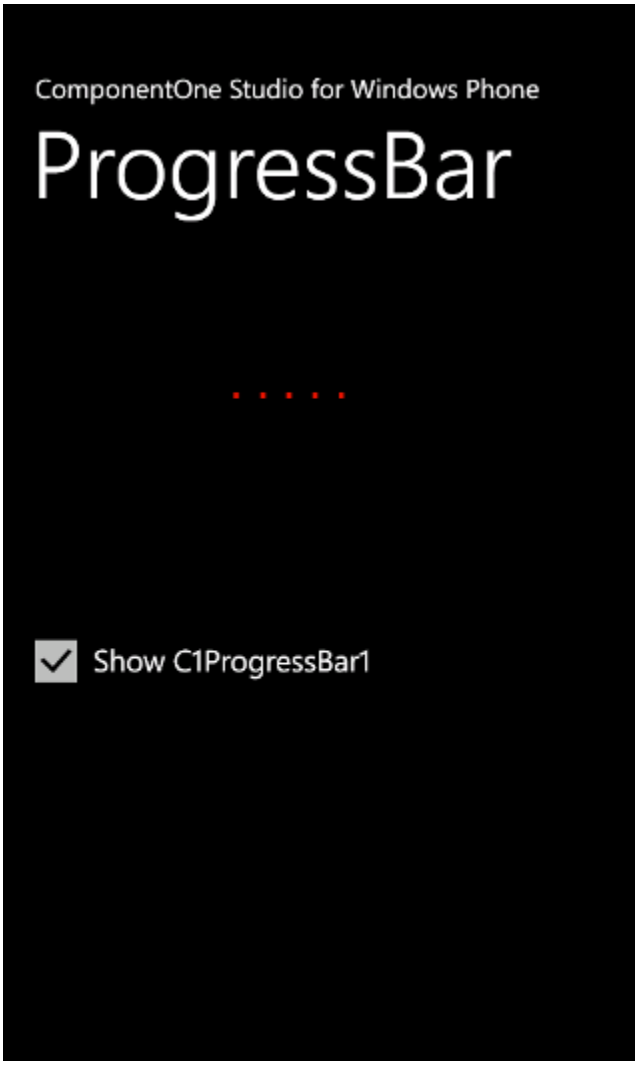
```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <my:C1ProgressBar HorizontalAlignment="Left" Margin="3"
Name="C1ProgressBar1" VerticalAlignment="Top" Height="200" Width="450" />
<CheckBox Content="Show C1ProgressBar1" IsChecked="{Binding
IsIndeterminate, ElementName=C1ProgressBar1, Mode=TwoWay}"/>
</Grid>
```

You have successfully customized the **C1ProgressBar** control. In the next step, you will observe the final application.

Step 3 of 3: Running the Application

To run your application and observe **ProgressBar for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.
2. Select the **Show C1ProgressBar1** check box. The progress bar will now appear as a series of red dots moving across the screen similar to the following image:



Congratulations! You've completed the **ProgressBar for Windows Phone** quick start.

About C1ProgressBar

C1ProgressBar displays an animated repeating pattern of dots to indicate an indeterminate operation is in progress. When active in an application, the progress bar appear similar to the following image:

The two main properties in the **C1ProgressBar** control are **ActualIsIndeterminate** and **IsIndeterminate**.

- **ActualIsIndeterminate**

  This property indicates whether the actual indeterminate property should be reflecting a specific value.
- **IsIndeterminate**

  This property indicates if the **C1ProgressBar** displays continuous, indeterminate feedback.

Popup

Quickly gather user input using **ComponentOne Popup™ for Windows Phone.** The **C1Popup** control is an advanced version of the primitive **Popup** class with added features such as animation and back button support. Popups are convenient because they eliminate the need of having to create a separate page to display or gather additional information from the user.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

Popup for Windows Phone Features

ComponentOne Popup for Windows Phone allows you to create customized, rich applications. Make the most of **Popup for Windows Phone** by taking advantage of the following key features:

- **Animation**

  Unlike the primitive **Popup** class, the **C1Popup** control fills the entire device frame and performs animation. The **IsAnimationEnabled** property determines whether the control plays the default tilt animation on open and close. See Disabling the Popup Animation for details.

- **Back Button Support**

  You can open and close the **C1Popup** control with the **IsOpen** property. Plus, with support for the device back button, users can also close the pop-up by simply pressing the back button.

- **Supports Any Content**

  Set any content within the **C1Popup** control. For example, you can specify a **UserControl** as the content. See Creating a Popup in a UserControl for an example.

- **Show or Hide Application Bar and System Tray**

  Set the **HideApplicationBarWhenOpen** property to control whether the application bar is open or closed when **C1Popup** is open. Also, set the **HideTrayBarWhenOpen** property to control the system tray visibility. See Showing the Application Bar with C1Popup Open and Showing the System Tray with C1Popup Open for details.

- **Optimized performance**

  Unlike the primitive **Popup** class, the **C1Popup** control makes special handling to suppress underlying UI updates while popup is opened. It improves the overall application performance in cases when underlying UI is complex.

Popup for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **Popup for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add the **Popup for Windows Phone** control to your application, and add content to the control.

Step 1 of 3: Creating a Phone Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **Popup for Windows Phone**. When you add a C1Popup control to your application, you'll have a complete, functional numeric editor. You can further customize the control to your application.

To set up your project and add a C1Popup control to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. Right-click the project in the Solution Explorer and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Phone.dll** assembly and select **OK**.
6. Add the XAML namespace to the `<phone:PhoneApplicationPage>` tag by adding `xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"` so it appears similar to the following:

```
<phone:PhoneApplicationPage
    x:Class="C1WP.MainPage"
    xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">
```

7. Edit the **TitlePanel** content to change the text in the **TextBlock** controls. It will appear similar to the following:

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="ComponentOne Studio for
Windows Phone" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="Popup" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

You've successfully created a Windows Phone application. In the next step you'll add a **C1Popup** control to the application, and customize the control.

Step 2 of 3: Adding Popup for Windows Phone to your Project

In the previous step you created a new Windows Phone Application. In this step you'll add code to your application to finalize the application.

Complete the following steps:

1. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
2. Add the following XAML markup cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags to add a **StackPanel** containing a **Button** to the application:

```
<StackPanel VerticalAlignment="Center">
    <Button Content="Open Popup" Height="72"
HorizontalAlignment="Center" Margin="10,10,0,0" Name="btnOpen"
VerticalAlignment="Top" Width="200" Click="btnOpen_Click" />
</StackPanel>
```

   Note that you'll add code for the button's **Click** event handler later.
3. Add the following markup between the `</StackPanel>` and `</Grid>` tags to add a C1Popup control with content to your application:

```
<c1:C1Popup x:Name="c1Popup1">
    <!--Content here-->
    <StackPanel>
        <TextBlock Text="Select an option:" Margin="15" />
        <RadioButton x:Name="radio1" Content="Option 1" />
        <RadioButton x:Name="radio2" Content="Option 2" />
        <RadioButton x:Name="radio3" Content="Option 3" />
        <Button Content="Close Popup" Width="200" x:Name="btnClose"
Click="btnClose_Click" />
    </StackPanel>
</c1:C1Popup>
```

   Note that the **C1Popup** control contains text and buttons. In the next steps you'll add code to initialize one of the buttons.

   Your complete markup will appear like the following:

- XAML Markup

```
<phone:PhoneApplicationPage
    x:Class="C1WP.MainPage"
    xmlns:c1="clr-namespace:C1.Phone;assembly=C1.Phone"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
```

```
        <!--TitlePanel contains the name of the application and page
title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="ComponentOne
Studio for Windows Phone" Style="{StaticResource
PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="Popup" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>


        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <StackPanel VerticalAlignment="Center">
                <Button Content="Open Popup" Height="72"
HorizontalAlignment="Center" Margin="10,10,0,0" Name="btnOpen"
VerticalAlignment="Top" Width="200" Click="btnOpen_Click" />
            </StackPanel>
            <c1:C1Popup x:Name="c1Popup1">
                <!--Content here-->
                <StackPanel Background="DarkSlateBlue">
                    <TextBlock Text="Select an option:" Margin="15" />
                    <RadioButton x:Name="radio1" Content="Option 1" />
                    <RadioButton x:Name="radio2" Content="Option 2" />
                    <RadioButton x:Name="radio3" Content="Option 3" />
                    <Button Content="Close Popup" Width="200"
x:Name="btnClose" Click="btnClose_Click" />
                </StackPanel>
            </c1:C1Popup>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```

4. Select **View | Code** to switch to Code view.
5. Add the following **Click** event handler to the code:

- Visual Basic

```
Private Sub btnOpen_Click(sender As System.Object, e As
System.Windows.RoutedEventArgs) Handles btnOpen.Click
    ' open popup
    c1Popup1.IsOpen = True
End Sub
Private Sub btnClose_Click(sender As System.Object, e As
System.Windows.RoutedEventArgs)
    ' close popup
    c1Popup1.IsOpen = False
End Sub
```

- C#

```
private void btnOpen_Click(object sender, RoutedEventArgs e)
{
    // open popup
    c1Popup1.IsOpen = true;
}
private void btnClose_Click(object sender, RoutedEventArgs e)
```

```
{
    // close popup
    c1Popup1.IsOpen = false;
}
```

These buttons will open and close the **C1Popup** control.

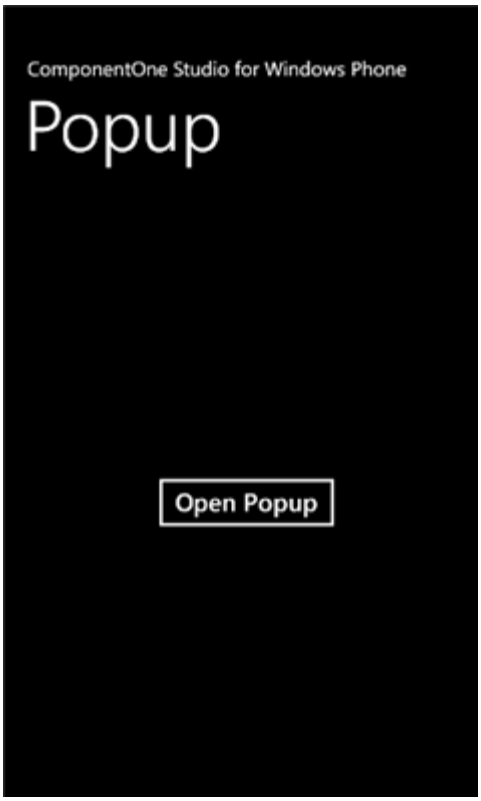
In this step you added the **C1Popup** control your application. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the Application

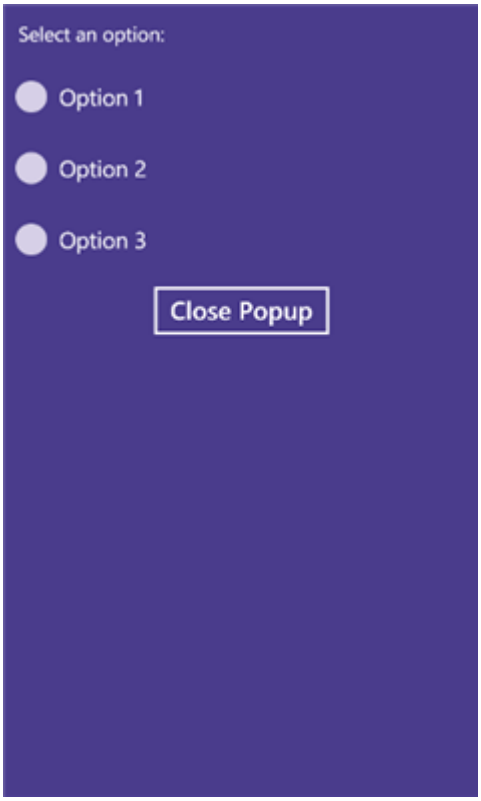
Now that you've created a Windows Phone application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **Popup for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging**.

   The application will appear in the Windows Phone emulator and will appear similar to the following:



1. Select the **Open Popup** button. The **C1Popup** control will appear:



2. Select the **Close Popup** button to close the **C1Popup** control.

Congratulations! You've completed the **Popup for Windows Phone** quick start and created a **Popup for Windows Phone** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

About C1Popup

The **C1Popup** control is an advanced version of the primitive **Popup** class with added features such as animation and back button support. Popups are convenient because they eliminate the need of having to create a separate page to display or gather additional information from the user.

C1Popup and the Popup class

You might ask why you would use the **C1Popup** control when you can just use the **Popup** class. While the **C1Popup** control is based on the primitive **Popup** class, it provides additional functionality and flexibility.

For example:

- The **C1Popup** control works like a regular UI control in that it adheres to standard layout conventions.
- You can work with **C1Popup** in XAML to have more control over the size and appearance of the control and its content.
- The **C1Popup** control also includes additional features, such as built-in support for animation, back button support, and better performance.

Basic Properties

ComponentOne Popup for Windows Phone includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below.

The following properties let you customize the C1Popup control:

Property	Description
Content	Gets or sets the value of the wrapped **ContentControl Content** property.
ContentTemplate	Gets or sets the data template that is used to display the content of the wrapped **ContentControl**.
HideApplicationBarWhenOpen	Gets or sets a value indicating whether the application bar must be hidden when the popup is open.
HideTrayBarWhenOpen	Gets or sets a value indicating whether the tray bar must be hidden when the popup is open.
IsAnimationEnabled	Gets or sets the Boolean value specifying whether control plays animations.
IsOpen	Gets or sets a value indicating whether this popup is open or not.

Using UserControls

The Popup for Windows Phone Quick Start demonstrates how to create a **C1Popup** inline in the XAML for conciseness; while certainly acceptable; this is generally not typical. You can create a Popup in XAML (for example, you might define the visual appearance of the Popup as a resource in a XAML **ResourceDictionary**). More likely you'll create a popup as part of a **UserControl**'s logic. You'd create the popup's content in a separate user control and use an instance of that control as the content of the popup.

For an example, see the Creating a Popup in a UserControl topic.

Hiding the Keyboard

The **C1Popup** control sets focus to itself upon load, so the virtual keyboard (SIP) will close automatically when **C1Popup** is open.

Popup for Windows Phone Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Popup control in general. If you are unfamiliar with the **ComponentOne Popup for Windows Phone** product, please see the Popup for Windows Phone Quick Start first.

Each topic in this section provides a solution for specific task using the **ComponentOne Popup for Windows Phone** product.

Each task-based help topic also assumes that you have created a new Windows Phone project, added a reference to the C1.Phone assembly, and added a namespace to the MainPage.

Creating a Popup in a UserControl

In this example, you'll create the popup's content in a separate user control and use an instance of that control as the content of the popup. Note that the application you'll create will look just like the application created in the Popup for Windows Phone Quick Start but the content of the popup is handled differently.

Complete the following steps:

1. Right-click the project name in the Solution Explorer and select **Add | New Item**.
2. In the Add New Item dialog box, choose **Windows Phone User Control**, name the control "C1PopupUserControl", and click **Add**.
3. Edit the UserControl's markup so it appears similar to the following:

   - XAML

```
<UserControl x:Class="C1Popup.C1PopupUserControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    d:DesignHeight="480" d:DesignWidth="480">

    <Grid x:Name="LayoutRoot">
        <StackPanel Background="DarkSlateBlue">
            <TextBlock Text="Select an option:" Margin="15" />
            <RadioButton x:Name="radio1" Content="Option 1" />
            <RadioButton x:Name="radio2" Content="Option 2" />
            <RadioButton x:Name="radio3" Content="Option 3" />
            <Button Content="Close Popup" Width="200" x:Name="btnClose"
/>
        </StackPanel>
    </Grid>
</UserControl>
```

   This is the content that will be used in the **C1Popup** control. Note that it is the same content used in the Popup for Windows Phone Quick Start.

4. Return to the **MainPage.xaml** source view. Add markup to the main **Grid ContentPanel** so it appears like the following:

   - XAML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel VerticalAlignment="Center">
        <Button Content="Open Popup" Height="72"
HorizontalAlignment="Center" Margin="10,10,0,0" Name="btnOpen"
VerticalAlignment="Top" Width="200" Click="btnOpen_Click" />
    </StackPanel>
</Grid>
```

5. Select **View | Code** to switch to Code View and add the **btnOpen_Click** event handler so the code looks like the following:

   - C#

```
public partial class MainPage : PhoneApplicationPage
{
    // Constructor
    public MainPage()
    {
        InitializeComponent();
    }
    private void btnOpen_Click(object sender, RoutedEventArgs e)
```

```
    {
        C1Popup popup = new C1Popup();
        C1PopupUserControl control = new C1PopupUserControl();
        popup.Content = control;
        popup.IsOpen = true;

        control.btnClose.Click += (s, args) =>
        {
            popup.IsOpen = false;
        };
    }
}
```

This markup will initialize the popup wen the button on the main page is selected.

What You've Accomplished

In this topic you created a **UserControl** to house the **C1Popup** control's content. Note that you'd need to set the Content property to the control.

Showing the Application Bar with C1Popup Open

By default, the application bar will hide when **C1Popup** is opened. You can control this behavior by setting the **HideApplicationBarWhenOpen** property. You can customize this property at design time, in XAML, and in code.

At Design Time

To set the HideApplicationBarWhenOpen property in Blend or Visual Studio, complete the following steps:

1. Click the C1Popup control once to select it.
2. Navigate to the Properties tab, and unselect the check box next to the HideApplicationBarWhenOpen property.

   This will set the HideApplicationBarWhenOpen property to show the application bar when **C1Popup** is open.

In XAML

For example, to set the HideApplicationBarWhenOpen property add `HideApplicationBarWhenOpen="False"` to the `<c1:C1Popup>` tag so that it appears similar to the following:

```
<c1:C1Popup x:Name="C1Popup1"
HideApplicationBarWhenOpen="False"></c1:C1Popup>
```

In Code

For example, to set the HideApplicationBarWhenOpen property add the following code to your project:

- Visual Basic
  ```
  C1Popup1.HideApplicationBarWhenOpen = False
  ```
- C#
  ```
  c1Popup1.HideApplicationBarWhenOpen = false;
  ```

What You've Accomplished

The application bar will no longer be hidden when the popup is displayed.

Showing the System Tray with C1Popup Open

By default, the system tray will hide when **C1Popup** is opened. You can control this behavior by setting the **HideTrayBarWhenOpen** property. You can customize this property at design time, in XAML, and in code.

At Design Time

To set the HideTrayBarWhenOpen property in Blend or Visual Studio, complete the following steps:

1. Click the C1Popup control once to select it.
2. Navigate to the Properties tab, and unselect the check box next to the HideTrayBarWhenOpen property.

   This will set the HideTrayBarWhenOpen property to show the system tray when **C1Popup** is open.

In XAML

For example, to set the HideTrayBarWhenOpen property add `HideTrayBarWhenOpen="False"` to the `<c1:C1Popup>` tag so that it appears similar to the following:

```
<c1:C1Popup x:Name="C1Popup1" HideTrayBarWhenOpen="False"></c1:C1Popup>
```

In Code

For example, to set the HideTrayBarWhenOpen property add the following code to your project:

- Visual Basic

```
C1Popup1.HideTrayBarWhenOpen = False
```

- C#

```
c1Popup1.HideTrayBarWhenOpen = false;
```

What You've Accomplished

The system tray bar will no longer be hidden when the popup is displayed.

Disabling the Popup Animation

By default, the **C1Popup** control will run a tilting animation when the **IsOpen** property changes and the popup opens. You can control this behavior by setting the **IsAnimationEnabled** property. You can customize this property at design time, in XAML, and in code.

At Design Time

To set the IsAnimationEnabled property in Blend or Visual Studio, complete the following steps:

1. Click the C1Popup control once to select it.
2. Navigate to the Properties tab, and unselect the check box next to the IsAnimationEnabled property.

   This will set the IsAnimationEnabled property so popup animation is disabled.

In XAML

For example, to set the IsAnimationEnabled property add `IsAnimationEnabled="False"` to the `<c1:C1Popup>` tag so that it appears similar to the following:

```
<c1:C1Popup x:Name="C1Popup1" IsAnimationEnabled="False"></c1:C1Popup>
```

In Code

For example, to set the IsAnimationEnabled property add the following code to your project:

- Visual Basic

```
C1Popup1.IsAnimationEnabled = False
```

- C#

```
c1Popup1.IsAnimationEnabled = false;
```

What You've Accomplished

No animation effect will be now be used when the popup is opened.

ToggleSwitch

Provide a simple on-off switch for user input using **ComponentOne ToggleSwitch™ for Windows Phone**. **C1ToggleSwitch** provides a more gesture-friendly experience for simple true/false values than a standard check box.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

ToggleSwitch for Windows Phone Key Features

ComponentOne ContextMenu for Windows Phone allows you to create customized, rich applications. Make the most of **ToggleSwitch for Windows Phone** by taking advantage of the following key features:

- **Supports Default Phone Theme**

  By default, the color used when **C1ToggleSwitch** is in the 'On' state matches the user-defined accent color of the phone. This enables you to maintain uniformity for your app and the user's phone without any additional coding.

- **Metro UI Design**

  C1ToggleSwitch supports the Metro UI design and interaction guidelines specified by Microsoft. It is similar to the toolkit **ToggleSwitchButton** control in features and functionality.

ToggleSwitch for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne ToggleSwitch for Windows Phone**. In this quick start you'll start in Visual Studio and create a new project, add the **C1ToggleSwitch** control to your application, and create a message box that appears when the C1ToggleSwitch is tapped, or turned off and on.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a Windows Phone application using **ToggleSwitch for Windows Phone**. Complete the following steps:

1. In Visual Studio, select **File | New | Project** to open the **New Project** dialog box.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Windows Phone Application**. Enter a **Name** for your project and click **OK**. The **New Windows Phone Application** dialog box will appear.
3. Click **OK** to close the **New Windows Phone Application** dialog box and create your project.
4. In the XAML window of the project, place the cursor between the `<Grid x:Name="ContentPanel"></Grid>` tags and click once.
5. Navigate to the Toolbox and double-click the **C1ToggleSwitch** icon to add the control to the grid. Create an opening and closing tag. The XAML markup should resemble the following:

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <my:C1ToggleSwitch>
```

```
            </my:C1ToggleSwitch>

        </Grid>
```

You have successfully created a Windows Phone application containing a **C1ToggleSwitch** control. In the next step, you will customize the control.

Step 2 of 4: Customizing the Control

In the previous step you set up the application's user interface and added a C1ToggleSwitch control to your application. In this step you will customize the control.

1. With the C1ToggleSwitch control selected on the page, select View | Properties.
2. Next to the **Content** property, enter "**Security System ON.**" Your XAML should now look like this:

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <my:C1ToggleSwitch Name="C1ToggleSwitch1" Content="Security
System ON.">

            </my:C1ToggleSwitch>
</Grid>
```

And the page will look like this:

Now you can add code to show a message when the switch is turned off and on.

Step 3 of 4: Adding Code

In the previous step, you added some content to the control. In this step, you'll add some code to show a message when the switch is tapped, or turned on and off.

1. In Design view, select the control and select **View | Properties** from the Visual Studio menu.
2. In the Properties window, click the **Events** tab.

3. Next to the **Checked** event, enter **C1ToggleSwitch1_Checked**. An event is created in the code.
4. Enter the following code for the event. The code will show a message that the switch is off when tapped.

```
private void C1ToggleSwitch1_Checked(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Home Security System is now OFF.");
            C1ToggleSwitch1.Content = "Security System is OFF.";
        }
```

5. Go back to the Visual Studio designer and locate the **Unchecked** event.
6. Enter the following code for the event. The code will show a message that the switch is on when tapped.

```
private void C1ToggleSwitch1_Unchecked(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Home Security System is now ON.");
            C1ToggleSwitch1.Content = "Security System is ON.";
        }
```

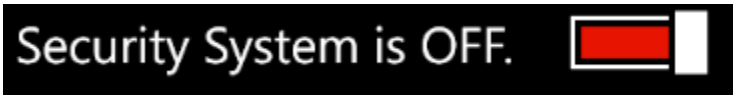
Now run the application to see what happens when the switch is tapped.

Step 4 of 4: Running the Application

To run your application and observe **ToggleSwitch for Windows Phone**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.
2. Tap the C1ToggleSwitch. A message box appears, stating the system is now off.

3. Tap ok and notice the content of the C1ToggleSwitch now states that the system if off.



4. If you tap the C1ToggleSwitch again, a message now appears stating that it is on.

Congratulations! You've completed the **ToggleSwitch for Windows Phone** quick start.