ComponentOne

Studio for Windows Phone Extended Library

By GrapeCity, Inc.

Corporate Headquarters
ComponentOne, a division of GrapeCity
201 South Highland Avenue
3rd 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 Extended Library Overview

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

ComponentOne Studio for Windows Phone Extended Library includes:

- **C1CoverFlow:** Presents an animated, three dimensional display of selectable items, similar to the Apple iTunes® application.
- **C1Reflector:** A **ContentControl** that adds a real 3D reflection to its content.

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.

CoverFlow

Visually navigate through items in an animated, three-dimensional graphical UI with **ComponentOne CoverFlow™ for Windows Phone**. Browse the Cover Flow by sliding your finger across the touch screen or tapping adjacent items.

Getting Started

- Working with the C1CoverFlow Control (page 10)

- Quick Start (page 7)

CoverFlow for Windows Phone Key Features

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

- **Configure the Angle of Items**

  The unselected items appear at an angle of perspective creating the illusion that the items are stacked. You can easily change the angle perspective by setting the **ItemAngle** property. See Item Angle (page 11) for more information.

- **Configure the Distance of Items**

  Alter the spacing between the items as well as the eye distance. The eye distance is what determines how much perspective the items have. See Eye Distance (page 11) and Item Distance (page 11) for more information.

- **Control the Speed of Items**

  Control the speed when the user selects an item or scrolls through the control. You can also control the speed at which the items' angles change as well as the speed at which the camera position changes. See Speed Settings (page 11) for more information.

- **Virtualization**

  Easily show a large amount of items in your **C1CoverFlow**; it will load images on demand.

- **Host Any Content**

  You can host just simple images in your Cover Flow or you can display richer content using data templates.

CoverFlow for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **CoverFlow for Windows Phone**. In this quick start, you'll start in Expression Blend to create a new project with a C1CoverFlow control. Once the control is added to your project, you'll customize it by setting a few properties and adding content to its content panel. You will then run the project to see the results of the quick start.

Step 1 of 3: Creating the Project

In this step, you'll create a Windows Phone application using **CoverFlow for Windows Phone**. You will also add a **StackPanel** control, a **TextBlock** control, and a folder containing three album cover images to the 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:my="clr-namespace:C1.Phone.Extended;assembly=C1.Phone.Extended"` 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:my="clr-namespace:C1.Phone.Extended;assembly=C1.Phone.Extended"
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="CoverFlow" 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 x:Name="ContentPanel"></Grid>` tags to add a **StackPanel** containing a **C1CoverFlow** control:

```
<my:C1CoverFlow></my:C1CoverFlow>
```

In this step, you created a project and added a C1CoverFlow control to it. In the next step, you'll customize the control.

Step 2 of 3: Adding Items to the C1CoverFlow Control

In the last step, you created an application and added a **C1CoverFlow** control to the application. In this step, you'll customize and add images to the C1CoverFlow control.

Complete the following steps:

1. Add markup between the `<my:C1CoverFlow>` tags customize its appearance:

   ```
   <my:C1CoverFlow x:Name="C1CoverFlow1" EyeDistance=".25" EyeHeight="1"
   Height="610" VerticalAlignment="Top" Margin="-1"></my:C1CoverFlow>
   ```

   The **EyeDistance** property sets the distance between the eye and the C1CoverFlow items, the **EyeHeight** property sets the level at which the camera sits and adjusts the view of the items accordingly.

2. Add images to the project by completing the following steps:

   a. Right-click the project in the Solution Explorer and select **Add New Folder**; name the new folder "Images".

   b. Right-click the **Images** folder and select **Add | Existing Item**.

      The **Add Existing Item** dialog box opens.

   c. Select five images to add. In this example, the added images are named **cover1.jpg**, **cover2.jpg**, and **cover3.jpg** and were taken from the **ComponentOne Studio for Silverlight** samples.

   d. Click **Open** to close the **Add Existing Item** dialog box and to add the images to the folder.

      The images are added to the **Images** folder.

3. Add **Image** controls to the C1CoverFlow control by adding the following markup between the `<my:C1CoverFlow></my:C1CoverFlow>` tags:

   ```
   <Image Source="images/cover101.jpg" Height="100" Width="100"/>

   <Image Source="images/cover102.jpg" Height="100" Width="100"/>

   <Image Source="images/cover103.jpg" Height="100" Width="100"/>

   <Image Source="images/cover104.jpg" Height="100" Width="100"/>

   <Image Source="images/cover105.jpg" Height="100" Width="100"/>
   ```

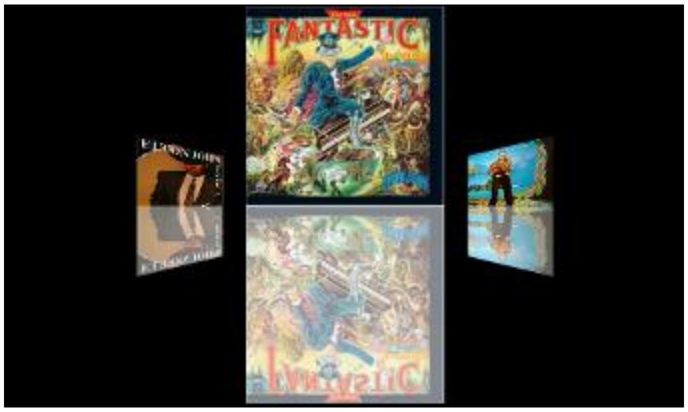
In this step, you added four items to the C1CoverFlow control and then set the items' properties. In the next step, you'll run program and see the results of this quick start topic.

Step 3 of 3: Running the Project

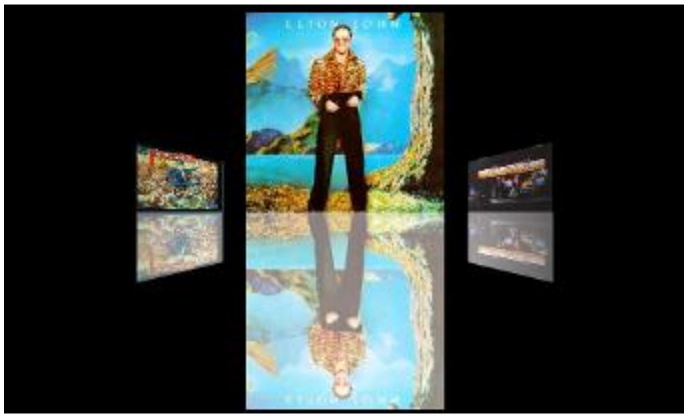
In the previous steps, you created a project with a C1CoverFlow control, set the control's properties, and added items to the C1CoverFlow control. All that's left for you to do now is run the project.

Complete the following steps:

1. Press F5 to run the project and observe that the project opens in the Windows Phone emulator and looks as follows:



2. Select an item in the **C1CoverFlow** and notice that item moves to the front:



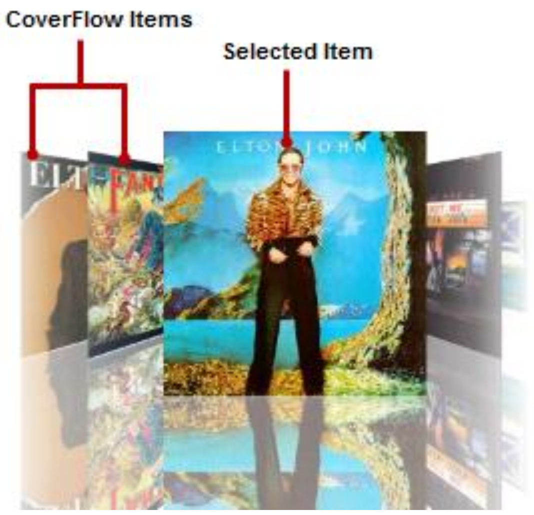
Congratulations! You have completed the **CoverFlow for Windows Phone** quick start. To learn more about the control, visit the Working with the C1CoverFlow Control (page 10) topic.

Working with the C1CoverFlow Control

The C1CoverFlow is an animated, three-dimensional user interface. It is an Items control and, as such, can hold text, images, and controls.

Users can browse through the C1CoverFlow control by selecting an item in the list. When a user touches an item on the list, that item will take focus by sliding into the center of the control, which will also cause other items to come into view.

The following image diagrams the elements of the C1CoverFlow control:



- **CoverFlow Items**

The CoverFlow items can be anything from text to images to controls. Creating a CoverFlow item is simple: Just add a C1CoverFlowItem to the C1CoverFlow control.

- **Selected Item**

  The selected item always appears at the center of the control. As users scroll through the items, the selected item will change. Users can also select items by clicking on them. You can change the selected item by setting the **SelectedIndex** property.

The following topics provide an overview of several of the C1CoverFlow control's features.

Eye Distance

The eye distance is the distance between the eye and the C1CoverFlow items. The eye distance is what determines how much perspective the items have.

You can modify the eye distance by setting the EyeDistance property. The value of this property is relative to the current cover size, meaning that setting it to a value of 0.5 means that the distance will be half the size of the CoverFlow item. By default, this property is set to a value of 1.5.

Eye Height

The EyeHeight property sets the level at which the camera sits and adjusts the view of the items accordingly. The value of this property is relative to the item size, meaning that a value of 0.5 will center the camera vertically. A value of 1 will place the camera at the bottom, and a value of 0 will place the camera overhead. The default value of this property is 0.25.

Item Angle

The ItemAngle property sets the angle of the control's unselected items. The value of the ItemAngle property is interpreted as a measurement of degree, so setting the ItemAngle property to 180 would flip the image horizontally. The default item angle is 60, which turns the unselected items at a 60-degree angle.

SelectedItem Offset

The SelectedItemOffset property determines how much closer the selected item is to the viewer than the unselected items. The value of this property is relative to the current item size, meaning that a value of 0.5 offsets the item to half of its own size.

Selected Item Distance

The SelectedItemDistance property sets the distance between the selected item and items directly next to it. The value for this property is relative to the current item size, meaning that a value of 0.5 places the distance between the selected item and the items next to it at half the size of the selected item.

Item Distance

The ItemDistance property sets the distance between the unselected items. The value for this property is relative to the current item size, meaning that the distance between the items will be half of the item's size.

Note: The ItemDistance property doesn't change the distance between the selected item and the items beside it.

Speed Settings

You can adjust the three types of speed on the C1CoverFlow control: ItemSpeed, ItemAngleSpeed, and EyeSpeed. The table below provides an overview of each speed property.

Property	Description

ItemSpeed property	Gets or sets the speed at which the items' positions change when a user selects an item or scrolls through the control. This value should be greater than 0 and less than or equal than 1, with 1 meaning instantaneous changes.
ItemAngleSpeed property	Gets or sets the speed at which the items' angles change when a user selects an item or scrolls through the control.
EyeSpeed property	Gets or sets the speed at which the camera position changes. This value should be greater than 0 and less than or equal than 1, with 1 meaning instantaneous changes.

CoverFlow for Windows Phone Layout and Appearance

The following topics detail how to customize the C1CoverFlow 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.

CoverFlow for Windows Phone Appearance Properties

ComponentOne CoverFlow 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.

Text Properties

The following properties let you customize the appearance of text in the C1CoverFlow 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.

Content Positioning Properties

The following properties let you customize the position content of the C1CoverFlow control.

Property	Description
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to a FrameworkElement when it is composed within a parent object such as a panel or items control.

HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to a FrameworkElement when it is composed within a layout parent, such as a panel or items control.

Color Properties

The following properties let you customize the colors used in the C1CoverFlow control.

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 border of the C1CoverFlow control.

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 C1CoverFlow 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.

CoverFlow 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 C1CoverFlow control in general. If you are unfamiliar with the **ComponentOne CoverFlow for Windows Phone** product, please see the CoverFlow for Windows Phone Quick Start (page 7) first.

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

Adding Images to the C1CoverFlow Control

In this topic, you'll learn how to add images to the C1CoverFlow control in Design view, in XAML, and in code. Note that you would need to add your image(s) to the application before completing the following steps..

In Design View

Complete the following steps:

1. Add a C1CoverFlow control to your project and select the control in Design view.
2. Navigate to the Toolbox and double-click the **Image** icon to add the **Image** control to the C1CoverFlow control. If the **Image** control was not added to the control, you may need to move its markup between the **C1CoverFlow** tags in XAML view.
3. Select the **Image** control in Design view.
4. In the **Properties** window, click the **Source** ellipsis button.

   The **Choose Image** dialog box opens.
5. Navigate to the location of your image, select the image file, and click **Open** to add the image to the **Image** control. You can also select the **Add** button to locate an image not listed in the dialog box.

In XAML

Complete the following steps:

1. Add a C1CoverFlow control so that the XAML appears similar to the following:

```
<my:C1CoverFlow c1ext:C1CoverFlow>
```

2. Place the following XAML between the `<my:C1CoverFlow>` and `</my:C1CoverFlow>` tags, replacing "YourImage.png" with the name of your image file:

```
<Image Height="100" Width="100" Source="YourImage.png"/>
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow1"` to the `<my:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml** code page (either **MainPage.xaml.cs or MainPage.xaml.vb**, depending on which language you've chosen for your project).
3. Import the following namespace:
   - Visual Basic

     ```
     Imports System.Windows.Media.Imaging
     ```
   - C#

     ```
     using System.Windows.Media.Imaging;
     ```
4. Add the following code beneath the **InitializeComponent** method:

- Visual Basic

```
' Create the Image control
Dim Image1 As New Image()

' Create a bitmap image and add your image as its source
Dim BitMapImage1 As New BitmapImage()
BitMapImage1.UriSource = New Uri("YourImage.png",
UriKind.RelativeOrAbsolute)

' Add the bitmap image as the Image control's source
Image1.Source = BitMapImage1

'Add the Image control to the C1CoverFlow control
C1CoverFlow1.Items.Add(Image1)
```

- C#

```
// Create the Image control
Image Image1 = new Image();

// Create a bitmap image and add your image as its source
BitmapImage BitMapImage1 = new BitmapImage();
BitMapImage1.UriSource = new Uri("YourImage.png",
UriKind.RelativeOrAbsolute);

// Add the bitmap image as the Image control's source
Image1.Source = BitMapImage1;

//Add the Image control to the C1CoverFlow control
C1CoverFlow1.Items.Add(Image1);
```

This Topic Illustrates the Following:

You've added an image to your application. If you run your application, the image will be included in the **C1CoverFlow** control.

Changing the Angle of CoverFlow Side Items

You can change the angle of the items that lie on each side of the C1CoverFlow control's selected item by setting the ItemAngle property to a numeric value (for more information, see the Item Angle (page 11) topic). In this topic, you will set the ItemAngle property to **15** so that the side items will be turned at a 15-degree angle.

In Design View

Complete the following steps:

1. Select the C1CoverFlow control.

2. In the **Properties** window set the ItemAngle property to "15".

In XAML

Add `ItemAngle="1"` to the `<my:C1CoverFlow>` tag so that the markup resembles the following:

```
<my:C1CoverFlow Margin="0,0,87,233" ItemAngle="15">
```

In Code

Complete the following steps:

1. In XAML view, add `x:Name="C1CoverFlow"` to the `<my:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.ItemAngle = 15
     ```
   - C#

     ```
     C1CoverFlow1.ItemAngle = 15;
     ```
3. Run the program.

This Topic Illustrates the Following:

Run the application and note that the C1CoverFlow control will appear with its ItemAngle property set to **15**.

Changing the Camera's Vertical Position

You can change the vertical position of the camera by setting the EyeHeight property (for more information, see the Eye Height (page 11) topic). In this topic, you'll set the EyeHeight property to **1** so that the camera will view the C1CoverFlow items from the bottom.

In Design View

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** window, set the EyeHeight property to "1".

In XAML

Add `EyeHeight="1"` to the `<my:C1CoverFlow>` tag so that the markup resembles the following:

```
<my:C1CoverFlow Margin="0,0,87,233" EyeHeight="1">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow1"` to the `<my:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.EyeHeight = 1
     ```
   - C#

     ```
     C1CoverFlow1.EyeHeight = 1;
     ```

3. Run the program.

✓ **This Topic Illustrates the Following:**

Run the application and note that the C1CoverFlow control will appear with its EyeHeight property set to **1**.

Setting the Distance Between the Selected Item and the Side Items

The SelectedItemDistance property sets the distance between the selected item, which appears in the center of the control, and the items that are on either side of it (for more information, see the Selected Item Distance (page 11) topic). In this topic, you will set the SelectedItemDistance property to 0.4 so that the distance between the selected item and its side items is 4/10ths the size of the selected item.

In Design View

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** window, set the SelectedItemDistance property to "0.4".

In XAML

Add `SelectedItemDistance="0.4"` to the `<my:C1CoverFlow>` tag so that the markup resembles the following:

```
<my:C1CoverFlow Margin="0,0,87,233" SelectedItemDistance="0.4">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow"` to the `<my:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.SelectedItemDistance = 0.4
     ```
   - C#

     ```
     C1CoverFlow1.SelectedItemDistance = 0.4;
     ```
3. Run the program.

✓ **This Topic Illustrates the Following:**

Run the application and note that the C1CoverFlow control will appear with its SelectedItemDistance property set to **0.4**.

Setting the Distance Between Items

The ItemDistance property sets the distance between the unselected C1CoverFlow items (for more information, see the Item Distance (page 11) topic). In this topic, you will set the ItemDistance property to 1 so that the distance between items will be equal to the item size.

> **Note**: The ItemDistance property doesn't change the distance between the selected item and the items beside it. For more information, see the Selected Item Distance (page 11) topic.

In Design View

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** window, set the ItemDistance property to "1".

In XAML

Add `ItemDistance="1"` to the `<my:C1CoverFlow>` tag so that the markup resembles the following:

```
<my:C1CoverFlow Margin="0,0,87,233" ItemDistance="1">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow"` to the `<my:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.ItemDistance = 1
     ```
   - C#

     ```
     C1CoverFlow1.ItemDistance = 1;
     ```
3. Run the program.

✓ **This Topic Illustrates the Following:**

Run the application and note that the C1CoverFlow control will appear with its ItemDistance property set to **1**.

Reflector

Display a 2D or 3D reflection of any **UIElement** with **ComponentOne Reflector™ for Windows Phone**. Enhance the look of your graphics with this simple control.

Getting Started

- Working with the C1Reflector Control

- Quick Start|tag=Reflector for Windows Phone Quick Start

- Task-Based Help|topic=Reflector for Windows Phone Task-Based Help

Reflector for Windows Phone Key Features

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

- **3D Reflection**

  C1Reflector supports Media plane projections. To generate a reflection, you simply add an effect to the **ReflectionEffects** collection. **C1Reflector** ships with an opacity effect, and you can also use the standard Media effects such as blur and drop-shadow to apply to the reflection.

- **Use any UIElement as Content**

  Use any **UIElement** as reflected content. To add multiple **UIElements** you would add one outer container element such as a grid and fill it with many sub elements.

- **Customize the Look of Reflections**

  Apply and configure opacity and blur effects. With opacity, for instance, you can configure how fast the opacity grows across the content and its starting value.

- **Auto-Update**

  The reflection is automatically updated when the content changes.

Reflector for Windows Phone Quick Start

The following quick start guide is intended to get you up and running with **Reflector for Windows Phone**. In this quick start, you'll start in Visual Studio and create a new project, add a C1Reflector control to your application, add content to the C1Reflector control, and then customize the appearance of the C1Reflector control and its contents.

Step 1 of 3: Creating an Application with a C1Reflector Control

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

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:my="clr-namespace:C1.Phone.Extended;assembly=C1.Phone.Extended"` 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:my="clr-namespace:C1.Phone.Extended;assembly=C1.Phone.Extended"
   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="Reflector" 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 `<my:C1Reflector></my:C1Reflector>` tags within the `<Grid x:Name="ContentPanel"></Grid>` tags to add a **C1Reflector** panel. The XAML will resemble the following:

   ```
   <Grid x:Name="ContentPanel">

   <my:C1Reflector></my:C1Reflector>

   </Grid>
   ```

   If you run the project now, you'll see a blank application as you have not added content to the C1Reflector panel. You'll learn how to do this in the next step.

You've successfully created a Windows Phone application containing a C1Reflector control. In the next step, you will add two controls to the C1Reflector control.

Step 2 of 3: Adding Content to the C1Reflector Control

In this section of the quick start tutorial, you will add two controls to the C1Reflector control; however, the **Content** property can only accept one control at a time. You can circumvent this issue by adding a panel-based control as the C1Reflector control's content and adding then stacking multiple controls in the panel. The C1Reflector control will create a reflection for all controls added to a panel.

Complete the following steps:

1. Place your cursor between the `<my:C1Reflector>` and `</my:C1Reflector>` tags and press ENTER.
2. Add the following `<my:C1Reflector.ContentProjection>` markup between the `<my:C1Reflector>` and `</my:C1Reflector>` tags to customize the reflection displayed in the **C1Reflector** control so your markup will appear similar to the following:

   ```
   <my:C1Reflector>
       <my:C1Reflector.ContentProjection>
           <PlaneProjection GlobalOffsetX="20" GlobalOffsetY="20"
   GlobalOffsetZ="1"/>
       </my:C1Reflector.ContentProjection>
   </my:C1Reflector>
   ```

3. Add the `<StackPanel> </StackPanel>` tags between the `<my:C1Reflector>` and `</my:C1Reflector>` tags to add a **StackPanel** within the **C1Reflector** control. Your markup will appear similar to the following:

   ```
   <my:C1Reflector>
       <my:C1Reflector.ContentProjection>
           <PlaneProjection GlobalOffsetX="20" GlobalOffsetY="20"
   GlobalOffsetZ="1"/>
       </my:C1Reflector.ContentProjection>
       <StackPanel></StackPanel>
   </my:C1Reflector>
   ```

4. Add the `<Slider Width="400"/>` tag between the `<StackPanel></StackPanel>` tags to create a standard **Slider** control. Your markup will appear similar to the following:

   ```
   <my:C1Reflector>
       <my:C1Reflector.ContentProjection>
           <PlaneProjection GlobalOffsetX="20" GlobalOffsetY="20"
   GlobalOffsetZ="1"/>
       </my:C1Reflector.ContentProjection>
       <StackPanel>
           <Slider Width="400"/>
       </StackPanel>
   </my:C1Reflector>
   ```

   Since the **Slider** control is interactive, it will allow you to see the C1Reflector control's auto-update feature.

5. Add the `<TextBlock Text="The reflection will change when the slider moves."></TextBlock>` tag below the `<Slider Width="400"/>` tag to add a standard **TextBlock** control. Your markup will appear similar to the following:

```
<my:C1Reflector>
    <my:C1Reflector.ContentProjection>
        <PlaneProjection GlobalOffsetX="20" GlobalOffsetY="20"
GlobalOffsetZ="1"/>
    </my:C1Reflector.ContentProjection>
    <StackPanel>
        <Slider Width="400"/>
        <TextBlock FontSize="18" Text="The reflection will change when the
slider moves."></TextBlock>
    </StackPanel>
</my:C1Reflector>
```

You have successfully added custom content to the C1Reflector control and set properties that altered the appearance of the C1Reflector control and its contents.

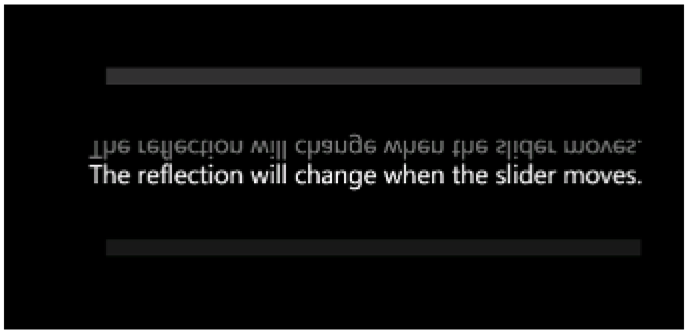
Step 3 of 3: Running the Project

In first previous steps of this quick start, you created a Windows Phone application containing a C1Reflector control, added content to the C1Reflector control, and modified the appearance of the C1Reflector control. In this step, you will run the project and observe the results.

Complete the following steps:

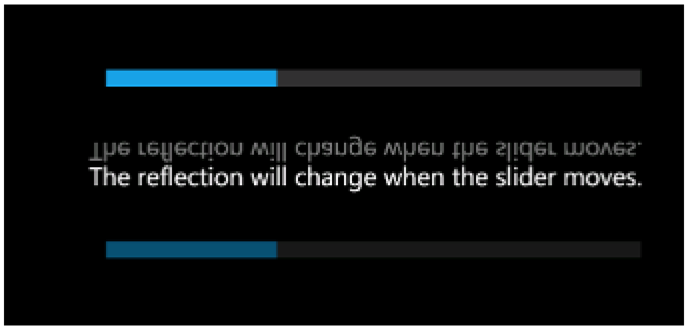
1. From the **Debug** menu, select **Start Debugging** to view you application in the Windows Phone emulator.

   The application appears similar to the following:



The reflection will change when the slider moves.

2. Select a point on the Slider control. Observe that there is a real-time reflection of each change that you make.



This is a result of the C1Reflector control's automatic updating feature.

Congratulations! You have successfully completed the **Reflector for Windows Phone** quick start. In this quick start, you've created a **Reflector for Windows Phone** application, added content to the C1Reflector control, customized the C1Reflector control's appearance, and viewed some of the run-time capabilities of the C1Reflector control.

Working with the C1Reflector Control

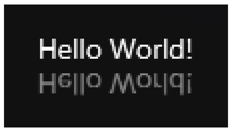
ComponentOne Reflector for Windows Phone includes one content control, C1Reflector, that can be used to create 2D and 3D reflections of text and Windows Phone controls. When it is added to the project, you will have to add the content that you want to have reflected to it. You can do this at desgin time using a simple drag-and-drop operation, in XAML, or in code. This section covers the basics of the C1Reflector control.

Reflector Content

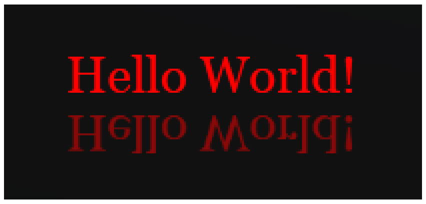
The C1Reflector control can hold text content, Windows Phone controls, and DataTemplates as its content. The C1Reflector control will cast a reflection of the content.

Text Content

If you just wanted to reflect a short phrase, you would just set the **Content** property to a string. It would look similar to this:



The example above is a little plain, but you can spruce up the text by setting just a few properties. You can change the font to a built-in Windows Phone font (or add your own font to the project), change the font color, and customize the font size from the **Properties** panel in Blend or Visual Studio. The image below uses a 36 point Georgia font in red.



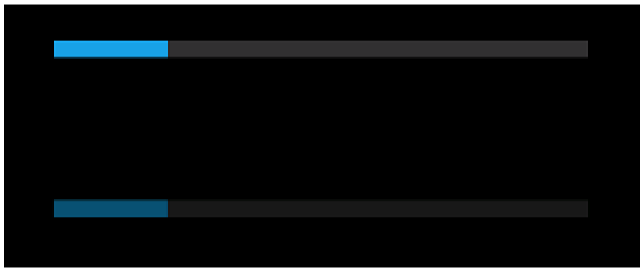
For task-based help about how to add text content, see Adding Simple Text Content to the Reflector (page 25).

Control Content

Adding control content is just as simple. You can add controls to the C1Reflector control in Blend by selecting a control icon and then using a drag-and-drop operation to add it to the C1Reflector control 's container. If you prefer to use XAML, you just wrap the `<c1ext:C1Reflector>` tags around the control so that your markup looks as follows:

```
<my:C1Reflector Width="400">
    <Slider Width="400"/>
</my:C1Reflector>
```

The control will appear similar to the following:



For task-based help about how to use controls with the C1Reflector control, see Adding a Control to the Reflector (page 26).

Adding Multiple Controls

Content property can only accept one control at a time. However, you can get around this limitation by adding a panel-based control as the C1Reflector control's child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the C1Reflector control, but its ability to hold multiple elements will allow you to show several controls in the content area at the same time.

Automatic Updating

The C1Reflector control will, by default, automatically update the reflection each time the control's content is updated. This is useful when you're using controls with interactive parts, such as the **C1Reflector** control or the **C1Accordion** control. The automatic update ensures that every movement of these controls will be shown in the reflection, so that if a user expands an accordion pane, for example, the accordion pane in the reflection will also be expanded.

If you'd prefer not to have your content automatically updated, you can turn the AutoUpdate property to **False**.

Note: AutoUpdate can slow down the application, especially if you intend to use multiple reflectors.

Plane Projection

Reflector for Windows Phone supports plane projections, which allows a two-dimensional control to be drawn on a three-dimensional plane. All parts of the control will still function as they would on a two-dimensional plane.

The three-dimensional effect is created by rotating the control along three separate planes – X, Y, and Z. You can set the rotation, center of rotation, global offset, and local offset of each of the three planes; this means that, in total, there are twelve properties that can be set to alter the projection of the C1Reflector control. Each property is described in the table below.

Property	Description
RotationX	Gets or sets the number of degrees to rotate the object around the x-axis of rotation.
RotationY	Gets or sets the number of degrees to rotate the object around the y-axis of rotation.
RotationZ	Gets or sets the number of degrees to rotate the object around the z-axis of rotation.
CenterOfRotationX	Gets or sets the x-coordinate of the center of rotation of the object you rotate.
CenterOfRotationY	Gets or sets the y-coordinate of the center of rotation of the object you rotate.
CenterOfRotationZ	Gets or sets the z-coordinate of the center of rotation of the object you rotate.
GlobalOffsetX	Gets or sets the distance the object is translated along the x-axis of the screen.
GlobalOffsetY	Gets or sets the distance the object is translated along the y-axis of the screen.
GlobalOffsetZ	Gets or sets the distance the object is translated along the z-axis of the screen.
LocalOffsetX	Gets or sets the distance the object is translated along the x-axis of the plane of the object.
LocalOffsetY	Gets or sets the distance the object is translated along the y-axis of the plane of the object.
LocalOffsetZ	Gets or sets the distance the object is translated along the z-axis of the plane of the object.

You can alter the projection of the entire C1Reflector control using the **Projection** property or you can just alter the projection of the control's content with the ContentProjection property. The results will be slightly different. If you alter the projection of the entire control, the project of both the content of the reflection will change. If you alter the projection of the content, only the projection of the content will change and then the reflection will mirror that change. You can set the control or content projection in XAML or in code, but it's easier to adjust the settings in Blend until you have achieved the look that you want.

Reflector 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 C1Reflector control in general. If you are unfamiliar with the **ComponentOne Reflector for Windows Phone** product, please see the **Reflector for Windows Phone** Quick Start first.

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

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

Adding Simple Text Content to the Reflector

You can add simple text to the C1Reflector control by setting the **Content** property to a string. Once the text is added, you will see the text and its reflection and the C1Reflector control.

At Design Time in Blend

To add simple text to the control, complete the following steps:

1. Select the C1Reflector control once to select it.
2. Under the **Properties** panel, set the **Content** property to "Hello World!".

In XAML

To add simple text to the control, add `Content ="Hello World"` to the `<my:C1Reflector>` tag so that the markup resembles the following:

```
<my:C1Reflector Content="Hello World!">
```

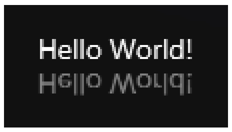
In Code

To add simple text to the control, complete the following steps:

1. Add `x:Name=C1Reflector1` to the `<my:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
2. Switch to Code view and add the following beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Reflector1.Content = "Hello World!"
     ```
   - C#
     ```
     C1Reflector1.Content = "Hello World!";
     ```
3. Run the program

This Topic Illustrates the Following:

The result of this topic will resemble the following image:



Adding a Control to the Reflector

The C1Reflector control can accept Windows Phone UI controls. To illustrate this feature, this topic will have you add a standard Windows Phone Button to the reflector. Once it is added to the control, you will see the control and its reflection in the C1Reflector control.

At Design Time

To add a control, complete the following steps:

1. Select the **C1Reflector** in the Design view.
2. From the Toolbox, select and double-click the **Button** icon to add it to the C1Reflector container.
3. If the control was not added to the **C1Reflector**, edit the markup in XAML view to move the control's tags within the **C1Reflector**.

In XAML

To add a control, place the following markup between the `<my:C1Reflector>` and `</my:C1Reflector>` tags:

```
<Button Content="Button"></Button>
```

In Code

To add a control, complete the following steps:

1. Add `x:Name=C1Reflector1` to the `<my:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.

2. Switch to Code view and add the following beneath the **InitializeComponent()** method:

   - Visual Basic

     ```
     'Create a button object and name it
     Dim Button1 As New Button()
     Button1.Content = "Button"
     'Set the C1Reflector control's content to the button
     C1Reflector1.Content = Button1
     ```

   - C#

     ```
     //Create a button object and name it
     Button Button1 = new Button();
     Button1.Content = "Button";
     //Set the C1Reflector control's content to the button
     C1Reflector1.Content = Button1;
     ```

3. Run the program

This Topic Illustrates the Following:

The result of this topic will resemble the following image: