ComponentOne

Extended Library for UWP

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

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Extended Library for UWP

The extended library consists of the following controls:

- **Book for UWP**

  Book for UWP allows you to present **UI elements** as the pages in a real book or magazine. A page-turning book control for innovative navigation, the control provides an interactive and unique way to visualize items and turn pages with gestures.

- **ColorPicker for UWP**

  ColorPicker for UWP provides a rich and interactive color selection interface. The control enables you to select colors from professionally-designed palettes and build custom palette based on your selection. You can work with the Basic color palette with standard color options and the Advanced color palette to customize your selection.

Book for UWP

Present information using a familiar book metaphor with **Book for UWP**, a page-turning book control for innovative navigation. With **Book for UWP** you can present **UIElement** objects as if they were pages in a regular paper book. You can see two elements at a time, turn pages with gestures, and more with **Book for UWP**.

Getting Started

Help with UWP Edition

Getting Started

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

Book for UWP Key Features

Book for UWP allows you to create customized, rich applications. Make the most of **Book for UWP** by taking advantage of the following key features:

- **Familiar Book Metaphor**

  C1Book enables you to present information innovatively using a familiar mental model – that of a book. But **Book for UWP** is not a typical static book, it's dynamic and interactive.

- **Real Book-like Visuals**

  C1Book enables you to customize the look and feel of the book pages; for example, show page folds. It not only looks like a book, but can be interacted with like a book.

- **Flexible Data Binding**

  C1Book is an **ItemsControl**, so you can bind it to any data source. Each item in the data source can be a **UIElement** or a generic object that gets converted into a **UIElement** using templates.

- **Gesture-based Navigation**

  C1Book allows users to navigate with gestures. Use tap, swipe, and slide gestures to turn pages.

Book Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to create a **C1Book** control with five items and its current page set to 3 (page 4, since this is on a zero-based index). For more information, see the Book for UWP Task-Based Help section.

Markup

```
<Extended:C1Book x:Name="c1book1" Height="300" Width="450" CurrentPage="3">
<TextBlock Text="Hello World! 1"/>
<TextBlock Text="Hello World! 2"/>
<TextBlock Text="Hello World! 3"/>
<TextBlock Text="Hello World! 4"/>
<TextBlock Text="Hello World! 5"/>
</Extended:C1Book>
```

Book for UWP Quick Start

The following quick start guide is intended to get you up and running with **Book for UWP**. In this quick start you'll create a new project, add a C1Book control to your application, and customize the appearance and behavior of the control.

Step 1 of 4: Creating the Book Application

In this step you'll create a new Universal Windows Application in Visual Studio. When you add a C1Book control to your application, you'll have a complete, functional book-like interface to which you can add images, controls, and other elements. To set up your project and add C1Book controls to your application, complete the following steps:

1. In Visual Studio select **File | New | Project**.
2. In the **New Project** dialog box, expand a language in the left pane
    1. Under the language, select **Windows Store**.
    2. In the templates list, select **Blank App (XAML)**.
    3. Enter a **Name** and click **OK** to create your project.
3. Open MainPage.xaml if it isn't already open, place the cursor between the <Grid> and </Grid> tags, and click once.
4. Navigate to the Toolbox and double-click the **C1Book** icon to add the control to the grid. This will add the reference and XAML namespace automatically. The XAML markup resembles the following:

    Markup

    ```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Extended:C1Book />
    </Grid>
    ```

5. Click once on the **C1Book** control in design view, navigate to the Properties window and set the following

properties:

- Set **Name** to "book" to give the control a name so it is accessible in code.
- Set **Width** to "450" and **Height** to "300".
- Set **IsFirstPageOnTheRight** to "true."

The XAML will appear similar to the following:

Markup

```
<Grid>
 <Extended:C1Book x:Name="book" Height="450" Width="600">
 </Extended: C1Book>
</Grid>
```

You've successfully set up your application's user interface, but **C1Book** control currently contains no content. In the next step you'll add content to the **C1Book** control, and then you'll add code to your application to add functionality to the control.

Step 2 of 4: Adding Content to the Book Control

In this step you'll add content to the C1Book control in design-time, XAML markup, and code to create a virtual book with several pages that can be turned. To customize your project and add content to the **C1Book** control in your application, complete the following steps:

1. Edit the markup so that it resembles the following:

Markup

```
<Border Grid.Row="1">
  <Grid>
    <Extended:C1Book x:Name="book" Height="450" Width="600">
      </Extended:C1Book>
  </Grid>
</Border>
```

2. Within the <Extended: C1Book> </Extended: C1Book> tags, add the following XAML markup. This will add several templates to the markup:

Markup

```
<Extended:C1Book.LeftPageTemplate>
                    <ControlTemplate TargetType="ContentControl">
                        <Border Background="WhiteSmoke" BorderBrush="WhiteSmoke"
BorderThickness="10 10 0 10">
                            <ContentPresenter Content="{TemplateBinding
Content}" ContentTemplate="{TemplateBinding ContentTemplate}" Margin="
{TemplateBinding Padding}" />
                        </Border>
                    </ControlTemplate>
                </Extended:C1Book.LeftPageTemplate>
                <Extended:C1Book.RightPageTemplate>
                    <ControlTemplate TargetType="ContentControl">
```

```
                        <Border Background="WhiteSmoke" BorderBrush="WhiteSmoke"
BorderThickness="10 10 0 10">
                            <ContentPresenter Content="{TemplateBinding
Content}" ContentTemplate="{TemplateBinding ContentTemplate}" Margin="
{TemplateBinding Padding}" />
                        </Border>
                    </ControlTemplate>
                </Extended:C1Book.RightPageTemplate>
                <Extended:C1Book.ItemTemplate>
                    <DataTemplate>
                        <Grid>
                            <Grid.Background>
                                <LinearGradientBrush EndPoint="1,1"
StartPoint="0,0">
                                    <GradientStop Color="#FFE2E8EB"
Offset="0.2"/>
                                    <GradientStop Color="#FFEEF4F7"
Offset="0.3"/>
                                    <GradientStop Color="#FFE2E8EB"
Offset="0.4"/>
                                </LinearGradientBrush>
                            </Grid.Background>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto" />
                                <RowDefinition Height="*"/>
                                <RowDefinition Height="Auto" />
                            </Grid.RowDefinitions>
                            <Image Source="{Binding Path=CoverUri}"
Stretch="Uniform" HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.Row="1"/>
                            <TextBlock Text="{Binding Path=Title}"
TextWrapping="Wrap" TextAlignment="Left" FontSize="11" FontWeight="Bold"
Margin="10,7,10,10" Foreground="#FF22445F"/>
                            <Grid HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Grid.Row="2" Grid.RowSpan="1" Margin="10,7,10,10">
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition />
                                    <ColumnDefinition />
                                </Grid.ColumnDefinitions>
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="Auto" />
                                    <RowDefinition Height="Auto" />
                                </Grid.RowDefinitions>
                                <TextBlock HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Text="{Binding Path=Price}" TextWrapping="NoWrap"
Grid.ColumnSpan="1" Grid.Row="1" Grid.Column="1" FontSize="11"
Foreground="#FF086C8E" FontWeight="Bold"/>
                                <TextBlock HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Text="{Binding Path=Id}" Grid.ColumnSpan="1"
Grid.Column="1" TextWrapping="NoWrap" FontSize="11" Foreground="#FF383838"/>
                                <TextBlock Text="Book Code:"
```

```
TextWrapping="NoWrap" FontSize="11" Foreground="#FF383838"/>
                                <TextBlock Text="Price:" TextWrapping="NoWrap"
Grid.Row="1" Margin="0,4,0,2" FontSize="11" Foreground="#FF383838"/>
                            </Grid>
                        </Grid>
                    </DataTemplate>
                </Extended:C1Book.ItemTemplate>
```

3. In Code view, add the following import statements to the top of the page:

C#

```
using C1.Xaml.Extended;
```

4. Add the following code just after the page's constructor. This code will allow you to call data from a separate code file:

C#

```
InitDataSource();
        }

        private void InitDataSource()
        {
            // load book descriptions from xml
            string peopleXMLPath =
Path.Combine(Package.Current.InstalledLocation.Path, "Amazon.xml");
            XDocument doc = XDocument.Load(peopleXMLPath);

            //XDocument doc = XDocument.Load(@"..\..\..\Amazon.xml");
            var books = from reader in doc.Descendants("book")
                        select new AmazonBookDescription
                        {
                            Title = reader.Attribute("title").Value,
                            CoverUri = reader.Attribute("coverUri").Value,
                            Id = reader.Attribute("id").Value,
                            Price = reader.Attribute("price").Value,
                            StockAmount =
int.Parse(reader.Attribute("stockAmount").Value)
                        };

            // set the book's item source
            book.ItemsSource = books;
        }
        #region Object Model

        public Orientation Orientation
        {
            get
            {
                return book.Orientation;
            }
            set
```

```
        {
            book.Orientation = value;
        }
    }

    public bool IsFirstPageOnTheRight
    {
        get
        {
            return book.IsFirstPageOnTheRight;
        }
        set
        {
            book.IsFirstPageOnTheRight = value;
        }
    }

    public PageFoldVisibility ShowPageFold
    {
        get
        {
            return book.ShowPageFold;
        }
        set
        {
            book.ShowPageFold = value;
        }
    }

    public PageFoldAction PageFoldAction
    {
        get
        {
            return book.PageFoldAction;
        }
        set
        {
            book.PageFoldAction = value;
        }
    }

    public double FoldSize
    {
        get
        {
            return book.FoldSize;
        }
        set
        {
            book.FoldSize = value;
        }
```

```
        }

        public int CurrentPage
        {
            get
            {
                return book.CurrentPage;
            }
            set
            {
                book.CurrentPage = value;
            }
        }

        #endregion
```

In this step you added templates to the **C1Book** control and added code to call the content from a separate file. In the next step you'll add files to the application.

Step 3 of 4: Adding Files to the Application

You've added templates and some code to the C1Book control. In this step, you'll add files to the application. These content files are the ones that the code calls in order to display the data.

1. Right-click your project name and select **Add | New Item** from the menu.
2. In the **Add New Item** window, select **XML File**. Name the file Amazon.xml. The file should open automatically.
3. Add the following markup to the Amazon.xml file:

Markup

```
<books>
  <book id="0672328917" coverUri="http://www.coverbrowser.com/image/bestsellers-2007/1943-1.jpg" price="$49.99" title="Windows Presentation Foundation Unleashed (WPF) (Unleashed)" stockAmount="1" />
  <!--
  <book id="073562528X" coverUri="http://borntolearn1.mslearn.net/images/2009/06/9780735625730f.jpg" price="$34.99" title="Introducing Microsoft Silverlight 3.0" stockAmount="200" />
  <book id="0596510373" coverUri="http://ecx.images-amazon.com/images/I/51DF0boY5fL.jpg" price="$49.99" title="Programming WPF" stockAmount="30" />
  -->
  <book id="0596527438" coverUri="http://www.coverbrowser.com/image/oreilly-books/97-1.jpg" price="$49.99" title="Programming C# 3.0 (Programming)" stockAmount="5" />
  <book id="0596519982" coverUri="http://ecx.images-amazon.com/images/I/51U9eZeXhuL._SL500_.jpg" price="$34.99" title="Essential Silverlight 2 Up-to-Date (Up-To-Date)" stockAmount="8" />
  <book id="1590599594" coverUri="http://ecx.images-amazon.com/images/I/51DedRUoZGL.jpg" price="$44.99" title="Beginning Web
```

```
Development, Silverlight, and ASP.NET AJAX: From Novice to Professional
(Beginning from Novice to Professional)" stockAmount="178" />
  <book id="059651509X" coverUri="http://www.coverbrowser.com/image/oreilly-
books/30-1.jpg" price="$44.99" title="Painting the Web" stockAmount="1" />
  <book id="0672330075" coverUri="http://vig-
fp.pearsoned.co.uk/bigcovers/0672330075.jpg" price="$39.99" title="Silverlight
1.0 Unleashed" stockAmount="1" />
  <book id="0672329689" coverUri="http://images.pearsoned-
ema.com/jpeg/large/9780672329685.jpg" price="$34.99" title="Creating Vista
Gadgets: Using HTML, CSS and JavaScript  with Examples in RSS, Ajax, ActiveX
(COM) and Silverlight" stockAmount="1" />
  <book id="1590599764" coverUri="http://knowfree.net/wp-
content/uploads/2008/05/1590599764011-250x299.jpg" price="$39.99"
title="Foundation Expression Blend 2: Building Applications in WPF and
Silverlight (Foundation)" stockAmount="13" />
  <!--
  <book id="1590599497" coverUri="http://i39.tinypic.com/dqhnoi.jpg"
price="$44.99" title="Pro Silverlight 2 in C# 2008 (Windows.Net)"
stockAmount="1" />
  <book id="159059939X"
coverUri="http://www.sql163.com/UploadFile/Book_Image/2009-
3/Sql163_2009329192012.jpg" price="$14.99" title="Silverlight and ASP.NET
Revealed" stockAmount="1" />
  -->
</books>
```

4. Right-click your project name again and select **Add | New Item** from the list. When the **Add New Item** window opens, choose **Code File** and name it AmazonBookDescription.cs.

5. The new code file should open automatically. Add the following code to the file:

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Input;

namespace BookTest
{
    public class AmazonBookDescription
    {
        public string Title { get; set; }
        public string CoverUri { get; set; }
        public string Id { get; set; }
        public string Price { get; set; }
        public int StockAmount { get; set; }
    }
}
```

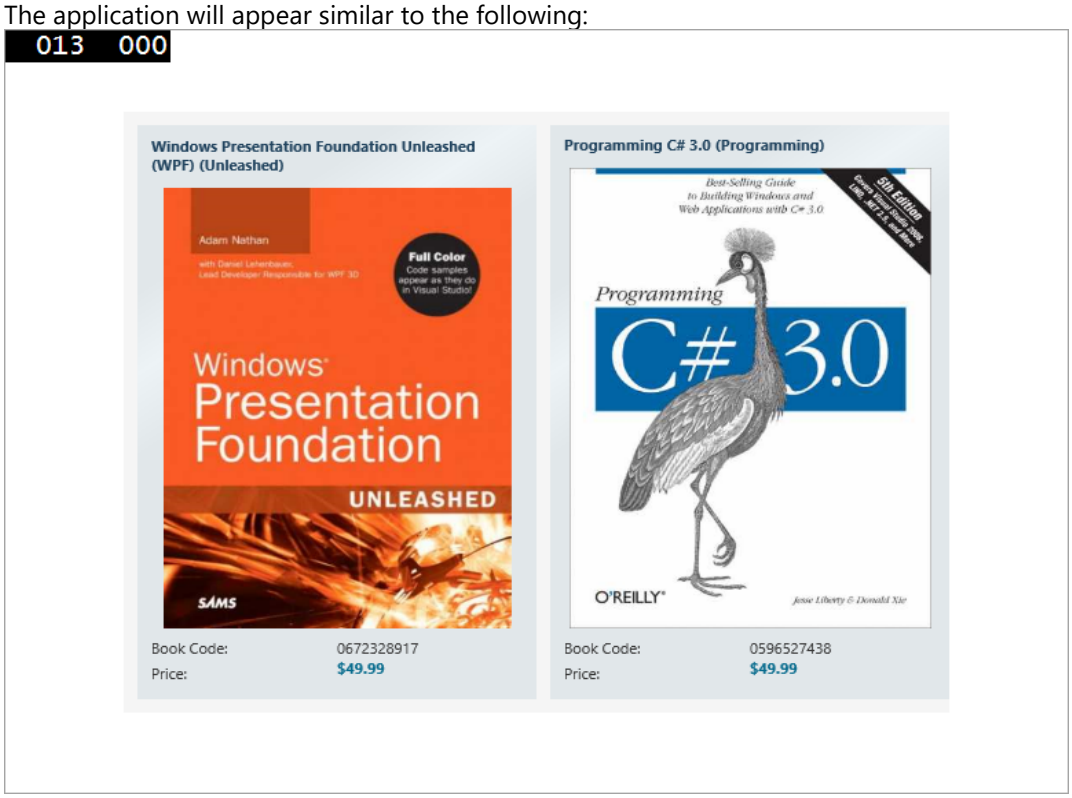
In this step, you added two files to your application. In the next step, you will run your application.

Step 4 of 4: Running the Book Application

Now that you've created an application the only thing left to do is run your application. To run your application and observe **Book for UWP**'s run-time behavior, complete the following steps:

1. From the **Project** menu, select **Run Project** to view how your application will appear at run time.

   The application will appear similar to the following:



   You set the C1Book.IsFirstPageOnTheRight property so that only one page is initially visible. Notice that when you hover over the lower or upper-right corner of the page folds back slightly to prompt you to turn the page; see Book Zones for more information.

2. Tap the upper-right corner of the page and notice that the page turns and the second and third pages are

004 000

visible:

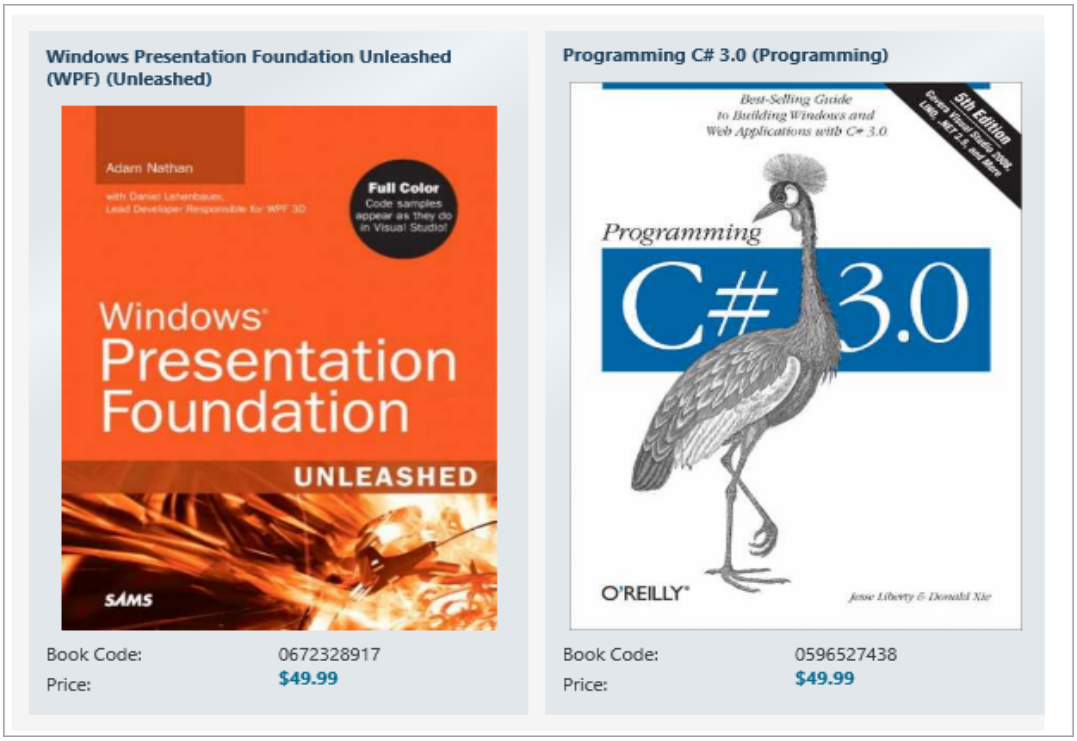
Congratulations!

You've completed the **Book for UWP** quick start and created a simple application, added and customized a **Book for UWP** control, and viewed some of the run-time capabilities of the control.

Working with Book for UWP

Book for UWP includes the **C1Book** control, a simple book control that acts as a container, allowing you to add controls, images, and more in a familiar book format. When you add the **C1Book** control to a XAML window, it exists as a container, similar to a panel, that can be customized and include added content.

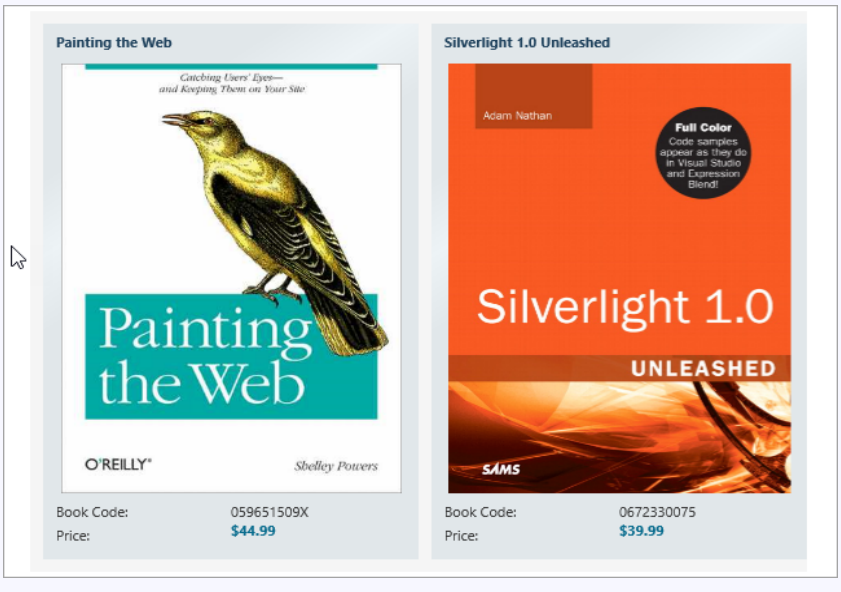
The control's interface looks similar to the following image:

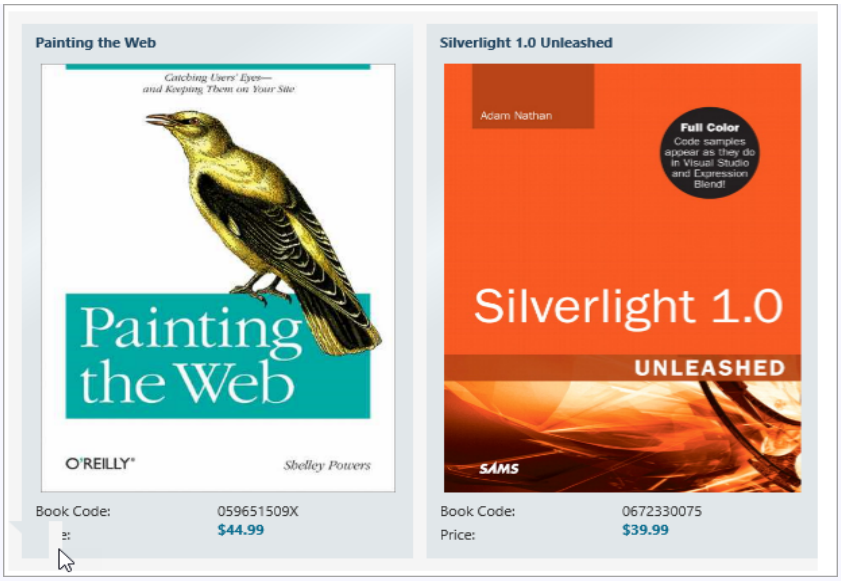
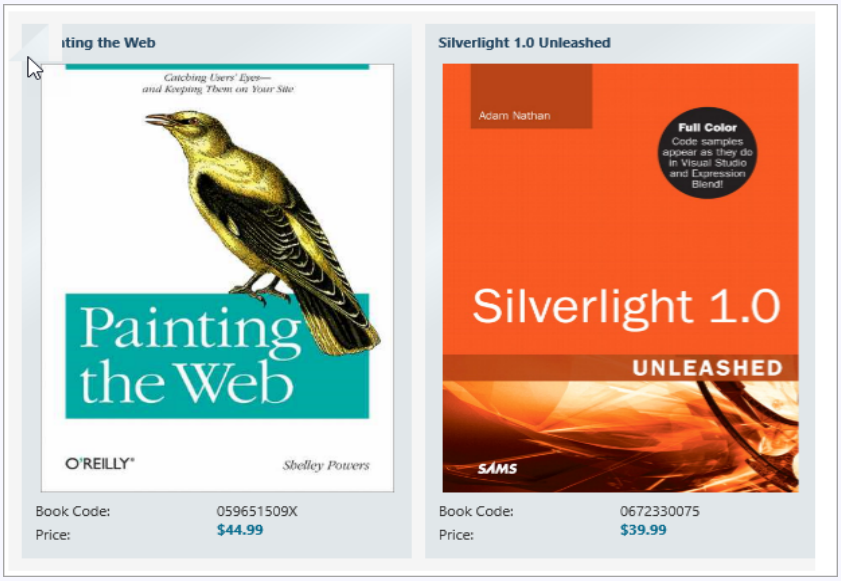
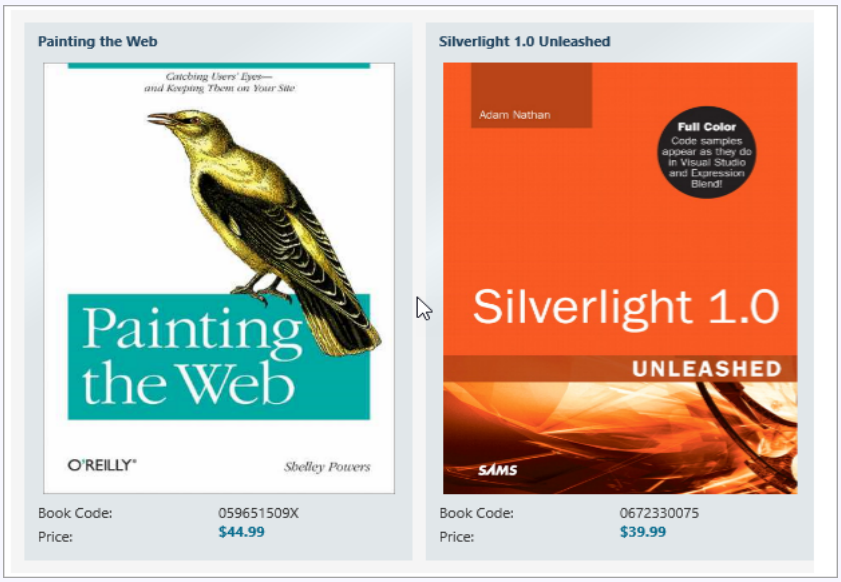


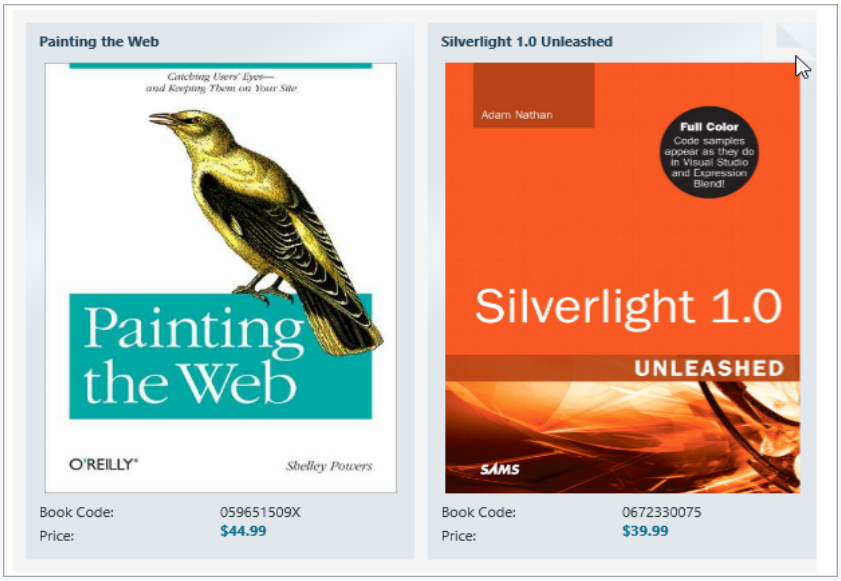
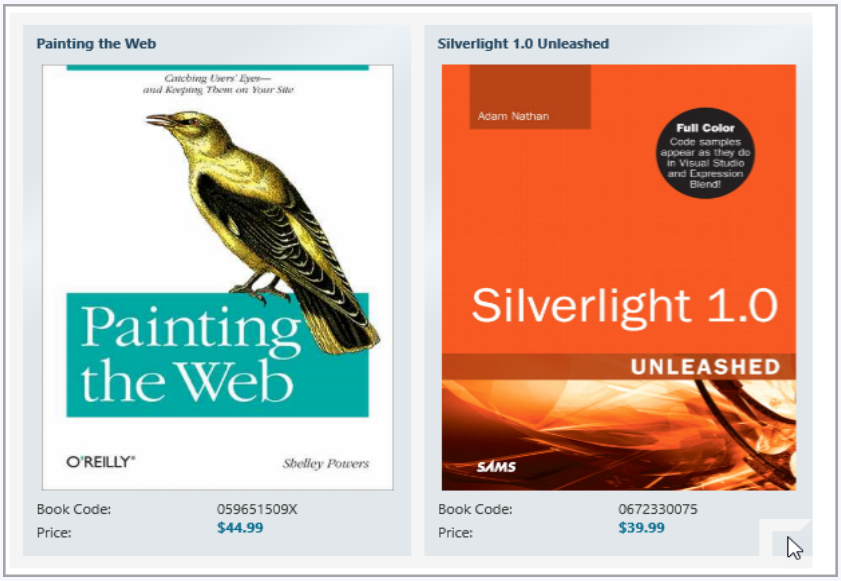
Book Zones

The **C1Book** control includes several zones. These zones let you customize what happens when users interact with various sections of the control. You can use the C1Book.CurrentZone property to get the user's current zone and you can use the C1Book.CurrentZoneChanged event to customize what happens when users move to a different zone.

There are six separate zones in the C1Book control. For an illustration of each zone, note the mouse's position in each of the images in the following table:

Zone	Description	Example
Out	Specifies the zone outside the borders of the book.	

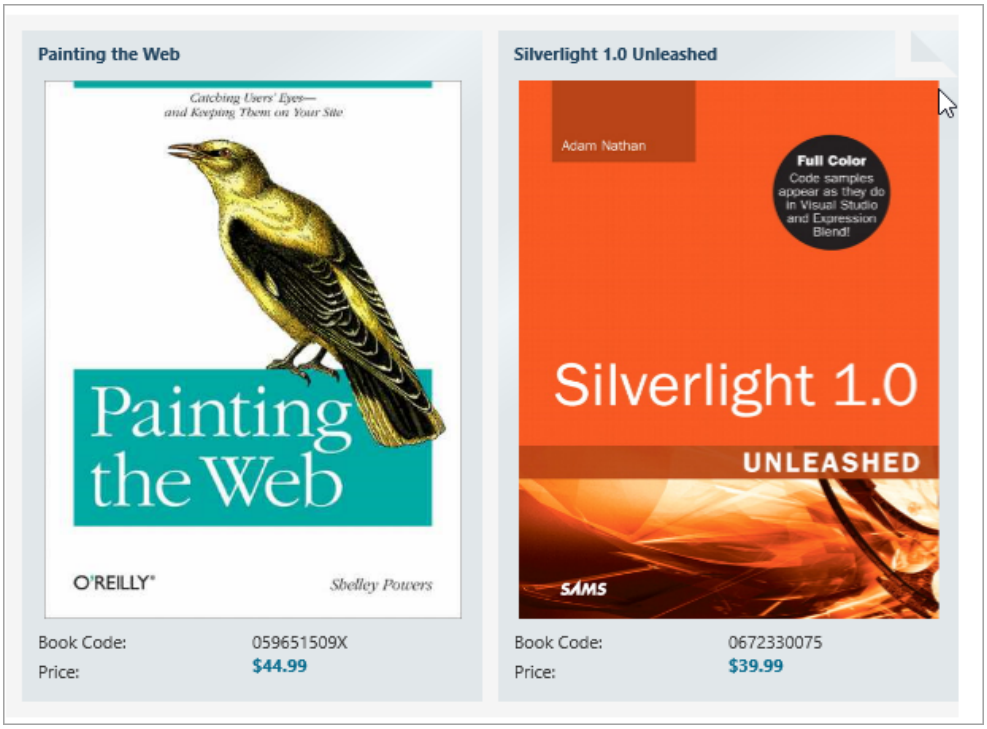
BottomLeft	Specifies bottom left fold zone.	
TopLeft	Specifies top left fold zone.	
Center	Specifies the center of the book (no fold zone).	

TopRight	Specifies top right fold zone.	
BottomRight	Specifies bottom right fold zone.	

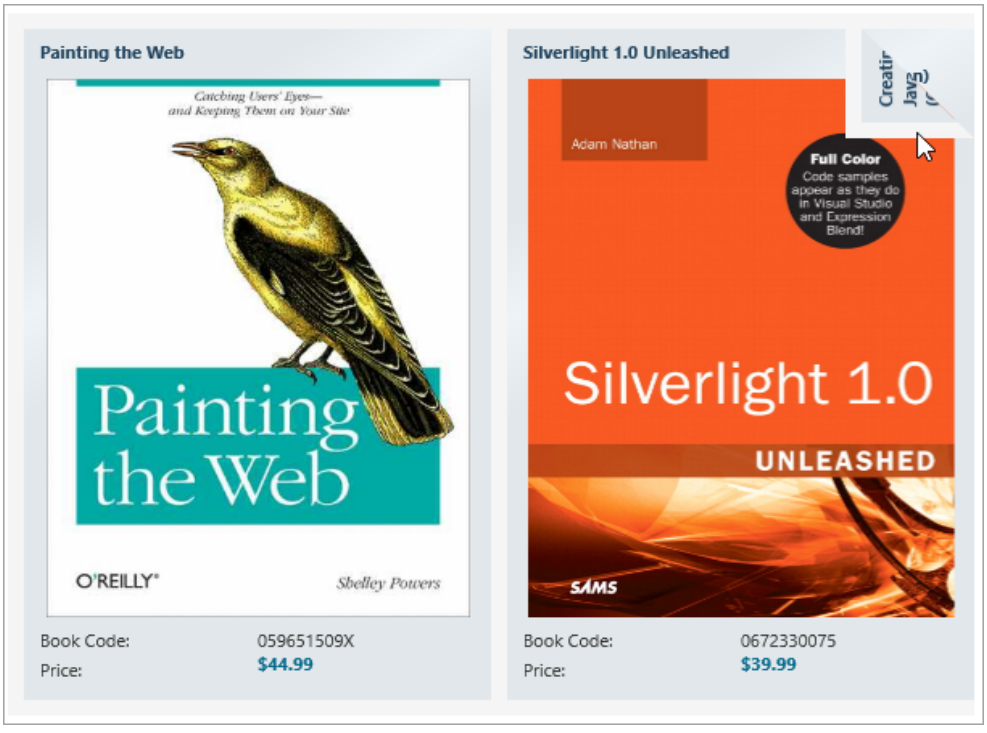
Page Fold Size

One way to customize the appearance of the book as users flip pages is by setting the size of the page fold using the C1Book.FoldSize property. Page folds appear when you hover the mouse over certain book zones, which serves as a cue that a page can be turned.

When you set the **C1Book.FoldSize** property, you will be setting the size of the page fold for all the pages – this includes the right top and bottom folds and the left top and bottom folds. For example, when **C1Book.FoldSize** is set to **40**, all the page folds will appear similar to the following image:



If set to a higher number, the folds will appear more prominent. For example, when **C1Book.FoldSize** is set to **80**, all the page folds will appear similar to the following image:



Page Fold Visibility

C1Book allows you to set the page fold visibility. By default, users will see a page fold when their mouse is over certain book zones. If you choose to, you can change the page fold visibility. You can set the C1Book.ShowPageFold property to any of the values from PageFoldVisibility to determine how users interact with the **C1Book** control:

Value	Description
OnMouseOver	The fold will be visible when the user drags the mouse over the edge of the page. This is the default setting.
Never	The fold will not be visible.
Always	The fold will always be visible.

Page Turning Options

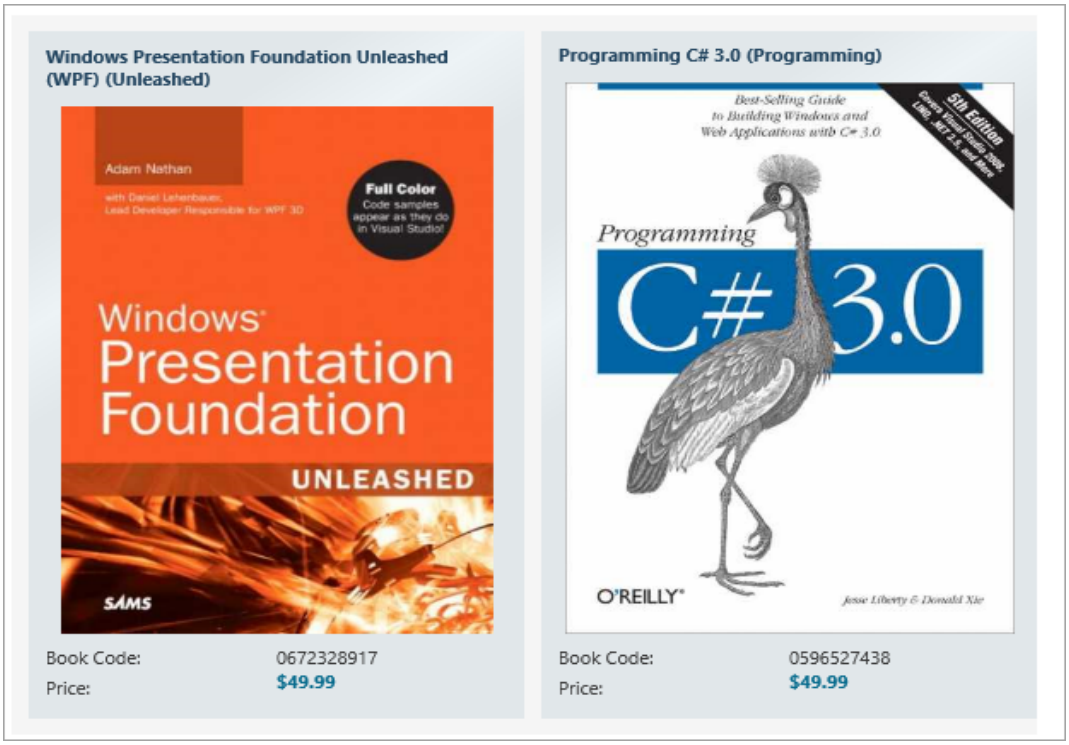
When users tap once on a page fold the book turns the current page to the previous or next page. This is the default behavior of C1Book. However, you can customize how pages turn by using the C1Book.PageFoldAction property. For example you can set **C1Book.PageFoldAction** so that users must double-tap on the page fold to turn the page, or you can prevent page turning on mouse click altogether, requiring that users perform a slide operation on the page fold to turn a page.

You can set the **C1Book.PageFoldAction** property to any of the following values from PageFoldAction to determine how users interact with the C1Book control:

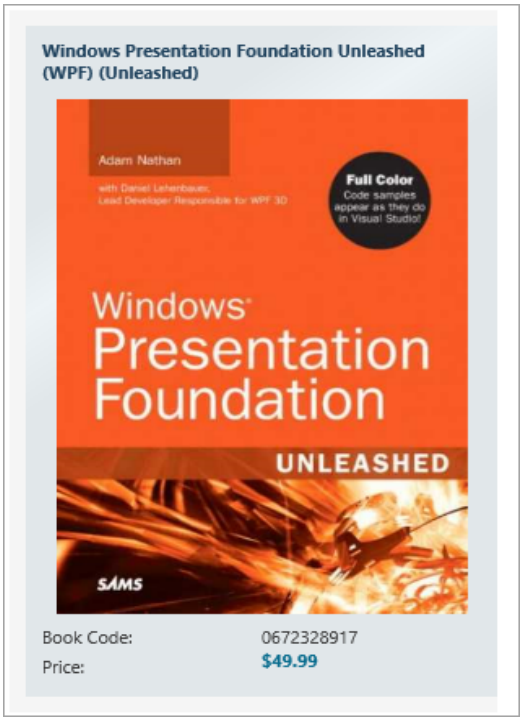
Value	Description
TurnPageOnClick	Turn the page when the user clicks the page fold.
TurnPageOnDoubleClick	Turn the page when the user double clicks the page fold.
None	Turn page when user drags the page fold across the book.

First Page Display

By default, the first page in the **C1Book** control is displayed on the left hand side. This makes it appear as if the book is open:



You can also change the display of the first page to appear on the right by setting the C1Book.IsFirstPageOnTheRight property to **True**. When the first page is set to display on the right side, it will appear similar to a cover, as if the book is closed:



Book Navigation

At run time users can navigate through the **C1Book** control using gestures. Users can tap on one of the book zones or perform a slide operation to turn the page. The **C1Book** control includes navigation-related methods, properties, and events to make it easier for you to determine what page a user is currently viewing, and to set the application's actions as users navigate through a book.

- C1Book.CurrentPage: Gets or sets the page that is currently displayed at run time.

  > **Note:** When you turn a page, the page displayed on the left of the two-page spread will be the **C1Book.CurrentPage**. Page numbering begins with **0** and page 0 is always displayed on the left. So if C1Book.IsFirstPageOnTheRight property is set to **True**, the first initial page of the book displayed on the right side will be page 1 with a hidden page 0 on the left side.

- C1Book.TurnPage: This method turns book pages forward or back one page. This method can be used to change the current page at run time.
- C1Book.CurrentPageChanged: This event will specify actions that happen when the current page is changed.
- C1Book.DragPageStarted and C1Book.DragPageFinished: These events specify actions to take when the user turns the page using a slide operation.

Book for UWP Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and know how to use the **C1Book** control in general. If you are unfamiliar with the **Book for UWP** product, please see the Book for UWP Quick Start first.

Each topic in this section provides a solution for specific tasks using the **Book for UWP** product. Most task-based help topics also assume that you have created a new blank Windows Store App and added the appropriate references and a **C1Book** control to the project – for information about creating the control, see Creating a Book.

Creating a Book

You can easily create a C1Book control at design time in XAML and in Code.

> If you create a **C1Book** control as in the following steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see Adding Items to a Book.

In XAML

To create a **C1Book** control using XAML markup, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select **UWP Edition**, and click **OK**.
2. Place your cursor between the <Grid> and </Grid> tags. Double-click the **C1Book** control in the Visual Studio Toolbox to add it to your application. This will also add the following namespace to the <Page> tag:

Markup
`xmlns:Extended="using:C1.Xaml.Extended"`

3. Add the following to the <Extended: C1Book> tag:

   - Name="c1book1"
   - Height="300"
   - Width="450"

   This markup will create an empty **C1Book** control named "c1book1" and set the control's size.

In Code

To create a **C1Book** control in code, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select **UWP Edition**, and click **OK**.
2. In the **MainPage.xaml** window, place your cursor within the <Grid> tag and add the following:
    - Name="Grid1"
3. Right-click within the **MainPage.xaml** window and select **View Code** to switch to Code view.
4. Add the following import statements to the top of the page:

Visual Basic

```
Imports C1.Xaml.Extended
```

C#

```
using C1.Xaml.Extended;
```

5. Add code to the page's constructor to create the **C1Book** control. It will look similar to the following:

Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim c1book1 as New C1Book
    c1book1.Height = 300
    c1book1.Width = 450
    Grid1.Children.Add(c1book1)
End Sub
```

C#

```
public MainPage()
{
    this.InitializeComponent();
    C1Book c1book1 = new C1Book();
    c1book1.Height = 300;
    c1book1.Width = 450;
    Grid1.Children.Add(c1book1);
}
```

This code will create an empty **C1Book** control named "c1book1", set the control's size, and add the control to the page.

What You've Accomplished

You've created a **C1Book** control. When you create a **C1Book** control as in the above steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see Adding Items to a Book.

Adding Items to a Book

You can add any sort of arbitrary content to a **C1Book** control. This includes text, images, layout panels, and other standard and 3rd-party controls. In this example, you'll add a **TextBlock** control to a C1Book control, but you can

customize the steps to add other types of content instead.

In XAML

For example, to add a **TextBlock** control to the book add <TextBlock Text="Hello World!"/> within the <Extended:C1Book> tag so that it appears similar to the following:

Markup

```
<Extended:C1Book Name="c1book1" Height="300" Width="450">
    <TextBlock Text="Hello World!"/>
</Extended:C1Book>
```

In Code

For example, to add a **TextBlock** control to the book, add code to the page's constructor so it appears like the following:

Visual Basic

```
Public Sub New()
    InitializeComponent()
Dim c1book1 as New C1Book
    c1book1.Height = 300
    c1book1.Width = 450
    Grid1.Children.Add(c1book1)
    Dim txt1 as New TextBlock
    txt1.Text = "Hello World!"
    c1book1.Items.Add(txt1)
End Sub
```

C#

```
public MainPage()
{
    this.InitializeComponent();
    C1Book c1book1 = new C1Book();
    c1book1.Height = 300;
    c1book1.Width = 450;
    Grid1.Children.Add(c1book1);

    TextBlock txt1 = new TextBlock();
    txt1.Text = "Hello World!";
    c1book1.Items.Add(txt1);
}
```

What You've Accomplished

You've added a control to the **C1Book** control. Run the application and observe that the **TextBlock** control has been added to the **C1Book** control. You can similarly add other content and controls.

Clearing Items in a Book

You may choose to allow users to clear all items from the **C1Book** control at run time, or you may need to clear the items collection when binding and then rebinding the control to another data source.

For example, to clear the book's content, add the following code to your project:

Visual Basic
```
c1book1.Items.Clear()
```

C#
```
c1book1.Items.Clear();
```

What You've Accomplished

The control's content will be cleared. If you run the application, you will observe that the book is blank.

Displaying the First Page on the Right

The C1Book.IsFirstPageOnTheRight property defines how the first page is displayed, whether it is on the right side or the left side. See First Page Display for more information. By default the C1Book control starts with the first page displayed on the left and two pages displayed, but you can customize this by setting the **C1Book.IsFirstPageOnTheRight** property in XAML and in code.

In XAML

For example, to set the **C1Book.IsFirstPageOnTheRight** property, add IsFirstPageOnTheRight="True" to the <Extended:C1Book> tag so that it appears similar to the following:

Markup
```
<Extended:C1Book Name="c1book1" Height="300" Width="450"
IsFirstPageOnTheRight="True">
```

In Code

For example, to set the **C1Book.IsFirstPageOnTheRight** property, add the following code to your project in the page's constructor:

Visual Basic
```
c1book1.IsFirstPageOnTheRight = True
```

C#
```
c1book1.IsFirstPageOnTheRight = true;
```

What You've Accomplished

You've set the first page to appear on the right. If you run the application, the first page will appear as a single page, like the book's cover:

Setting the Current Page

The C1Book.CurrentPage property gets or sets the value of the **C1Book** control's current page. By default the **C1Book** control starts with the first page displayed, but you can customize this by setting the **C1Book.CurrentPage** property in XAML and in code.

In XAML

For example, to set the **C1Book.CurrentPage** property to **3**, add CurrentPage="3" to the <Extended:C1Book> tag so that it appears similar to the following:

Markup

```
<Extended:C1Book x:Name="c1book1" Height="300" Width="450" CurrentPage="3">
```

In Code

For example, to set the **C1Book.CurrentPage** property to **3**, add the following code to your project:

Visual Basic

```
c1book1.CurrentPage = 3
```

C#

```
c1book1.CurrentPage = 3;
```

What You've Accomplished

You've changed the book's initial starting page. If you run the application, the initial page that appears will be page 3.

Navigating the Book with Code

You can set the displayed page using the C1Book.CurrentPage property, but you can also use the C1Book.TurnPage method to change the current page at run time. For more information, see Book Navigation. In this topic you'll add two buttons to your application, one that will turn to the previous page and one that will turn to the next page of the book.

To add additional navigation to your book, complete the following steps:

1. Set the name of the C1Book control as **c1book1** and add the following XAML markup between the <Extended:C1Book> and </Extended:C1Book> tags. This will add six pages that look like checkerboards to your application:

   Markup

   ```
   <Grid Name="checkers" Background="White" >
        <Grid.RowDefinitions>
           <RowDefinition Height=".25*" />
           <RowDefinition Height=".25*" />
           <RowDefinition Height=".25*" />
           <RowDefinition Height=".25*" />
         </Grid.RowDefinitions>
         <Grid.ColumnDefinitions>
           <ColumnDefinition Width=".25*" />
           <ColumnDefinition Width=".25*" />
           <ColumnDefinition Width=".25*" />
           <ColumnDefinition Width=".25*" />
         </Grid.ColumnDefinitions>
         <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
         <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
         <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
         <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
   ```

```
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid Name="checkers2" Background="White">
     <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
     </Grid.ColumnDefinitions>
     <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
     <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
     <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
     <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid x:Name="checkers3" Background="White" >
     <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions>
```

```
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid Name="checkers4" Background="White" >
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
```

```
</Grid>
<Grid x:Name="checkers5" Background="White" >
     <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
      </Grid.ColumnDefinitions>
      <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
      <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid Name="checkers6" Background="White" >
      <Grid.RowDefinitions>
         <RowDefinition Height=".25*" />
         <RowDefinition Height=".25*" />
         <RowDefinition Height=".25*" />
         <RowDefinition Height=".25*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width=".25*" />
         <ColumnDefinition Width=".25*" />
         <ColumnDefinition Width=".25*" />
         <ColumnDefinition Width=".25*" />
      </Grid.ColumnDefinitions>
      <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
      <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
      <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
      <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
```

```
        <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
        <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
        <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
        <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
        <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
        <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
        <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
        <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
        <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
```

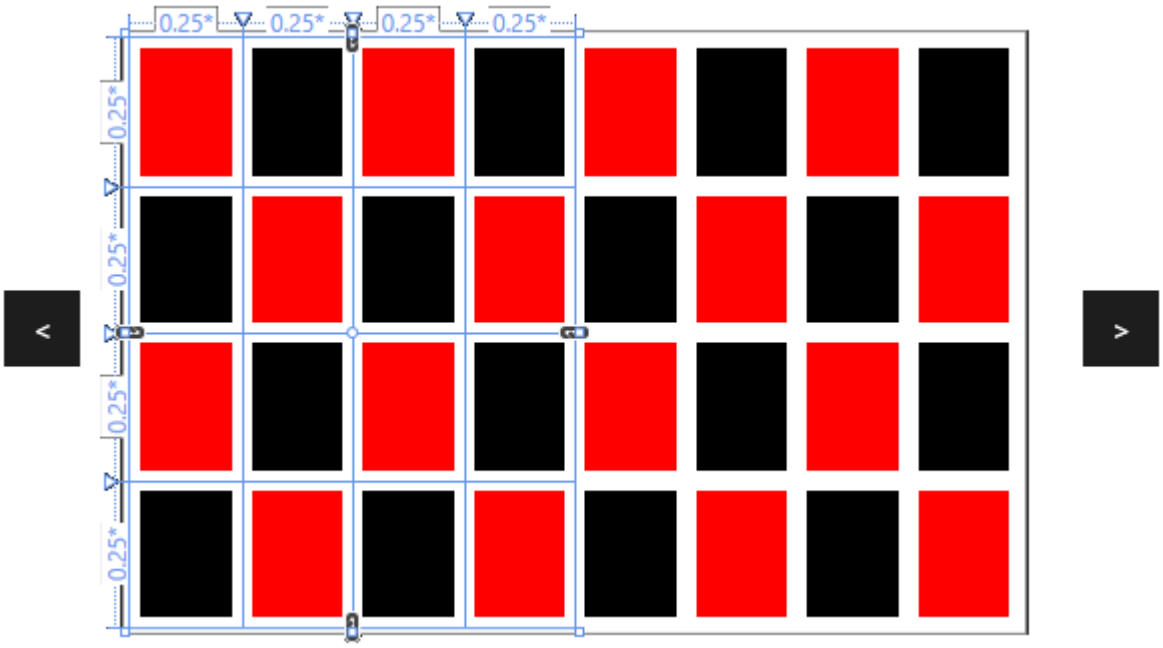
2. Navigate to the Toolbox and double-click the **Button** item twice to add two **Button** controls to your application.
3. Select the first button, navigate to the Properties window and set the following properties:
   - Set **Name** to "btn_last".
   - Set **Content** to "<".
   - Set **Height** and **Width** to "48"
4. Select the second button, navigate to the Properties window and set the following properties:
   - Set **Name** to "btn_next".
   - Set **Content** to ">".
   - Set **Height** and **Width** to "48"
5. Relocate the buttons by placing the **btn_last** button to the left of the book, and the **btn_next** button to the right of the book.
6. Double-click the left button to create the **Click** event.
7. Return to Design view and repeat the previous step with the right button so each button has a **Click** event specified.

   The XAML markup will appear similar to the following:

Markup

```
<Button x:Name="btn_last" HorizontalAlignment="Left" Margin="49,223,0,229"
Width="48" Height="48" Content="&amp;lt;" Click="btn_last_Click"/>
<Button x:Name="btn_next" HorizontalAlignment="Right" Margin="0,224,49,228"
Width="48" Height="48" Content="&amp;gt;" Click="btn_next_Click"/>
```

The page should now look similar to the following:



8. Switch to Code view and add the following import statements to the top of the page:

Visual Basic

```
Imports C1.Xaml.Extended
```

C#

```
using C1.Xaml;
using C1.Xaml.Extended;
```

9. Add code to the **Click** event handlers so they look like the following:

Visual Basic

```
Private Sub btn_next_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    c1book1.TurnPage(True)
End Sub
Private Sub btn_last_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    c1book1.TurnPage(False)
End Sub
```

C#

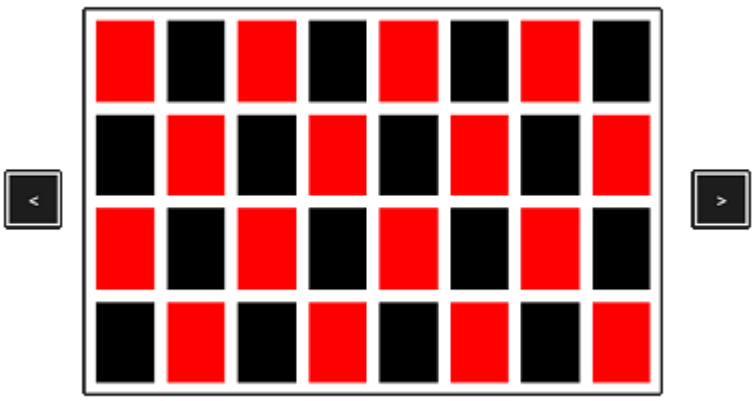
```
 public MainPage()
 {
  private void btn_next_Click(object sender,   System.Windows.RoutedEventArgs e)
  {
    c1book1.TurnPage(true);
}
```

```
private void btn_last_Click(object sender, System.Windows.RoutedEventArgs e)
{
    c1book1.TurnPage(false);
 }}
```

This code will turn the book a page forward or back depending on the button tapped.

What You've Accomplished

You've customized navigation in the book. To view the book's navigation, run the application and tap the right button. Notice that the page turns to the next page with a page turning animation:



Tap the left button and observe that the book returns to the previous page.

ColorPicker for UWP

ColorPicker for UWP is a color input editor designed for providing an interactive and rich color selection interface to users. You can select colors from professionally-designed palettes or custom colors based on your selection. With ColorPicker for UWP, you can choose to use a **Basic** color palette with standard color options, an **Advanced** palette that enables users to customize their color selection, or both.

The C1ColorPicker class also includes C1SpectrumColorPicker, Hexadecimal Color, and RGB color support to provide an enhanced visual color input interface.

Color Picker Key Features

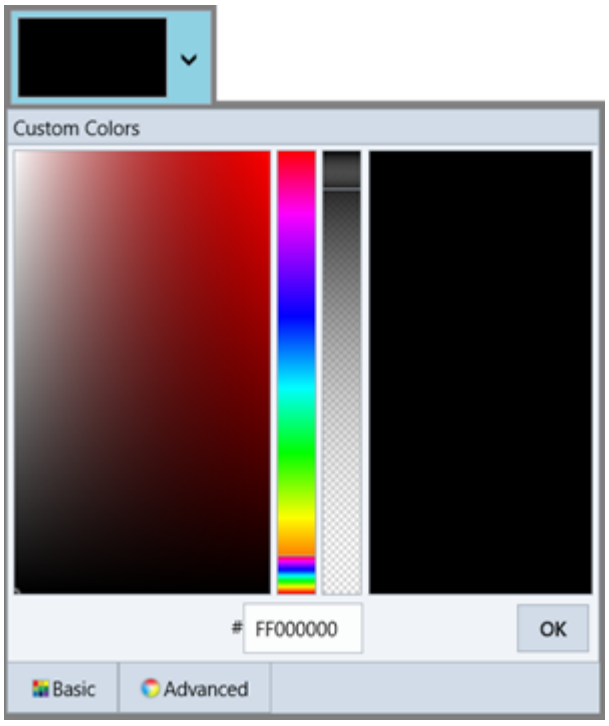
ColorPicker for UWP enables you to create customized, visually-advanced applications enriched with multi-color palettes. **ColorPicker for UWP** provides well-defined large buttons and icons to make them easy to use in supported devices.

- **20+ Professionally-Designed Color Palettes to Choose from**

  C1ColorPicker includes more than 20 predefined color palettes that match the themes available in Microsoft Office suite. Colors available in each palette go well with each other and let you create polished, professional applications.

- **Built-In Color Editor for Custom Color Selection**

  C1ColorPicker also includes a color editor. This allows users to create custom colors that are not available on standard color palette. The editor uses RGB models, and other features such as transparency and opacity for customizing color selection. These features can be viewed in Visual Studio project within **Properties** window for C1ColorPicker under **Background** property.



- **Modes**

  C1ColorPicker supports **Basic** and **Advanced** modes for color selection at run-time.

- **Composable Parts**

  C1ColorPicker includes three additional controls namely C1SpectrumColorPicker, C1HexColorBox and C1CheckeredBorder to support customization while selecting colors. The **C1SpectrumColorPicker** control provides access to advanced color picking functionality, while **C1HexColorBox** control is used for data

validation of hexadecimal code entries. The **C1CheckeredBorder** is simply added to display colors with varying transparency.

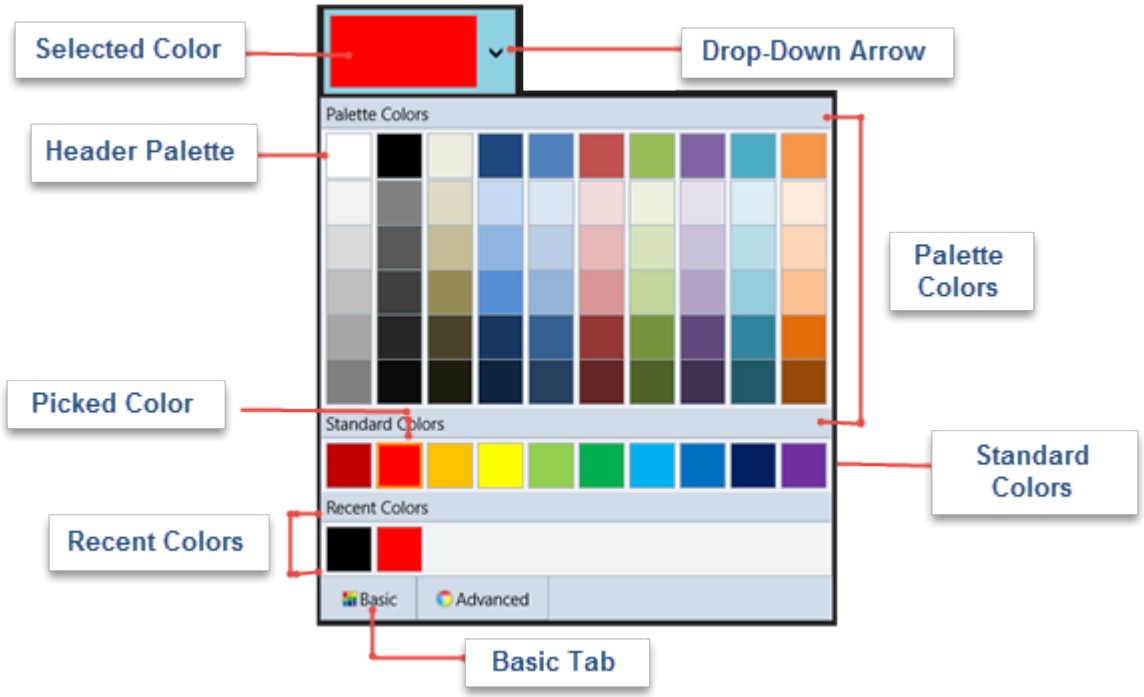
Visual Elements

ColorPicker for UWP allows users to create customized, visually-advanced applications enriched with multi-color palettes. The following illustrations show various elements of a C1ColorPicker.

Basic Mode

The Mode property of **C1ColorPicker** allows users to either choose colors from a preselected color palette or customize their own palette as per their choice. C1ColorPicker supports three modes provided by C1ColorPickerMode, namely **Basic, Advanced** and **Both.** By default, the Mode property is set to **Both** with Basic as well as Advanced color palettes visible to users.

Basic Mode: By default, C1ColorPicker opens with Basic tab open when the control's drop-down arrow is clicked. The Basic Tab appears similar to the following image.

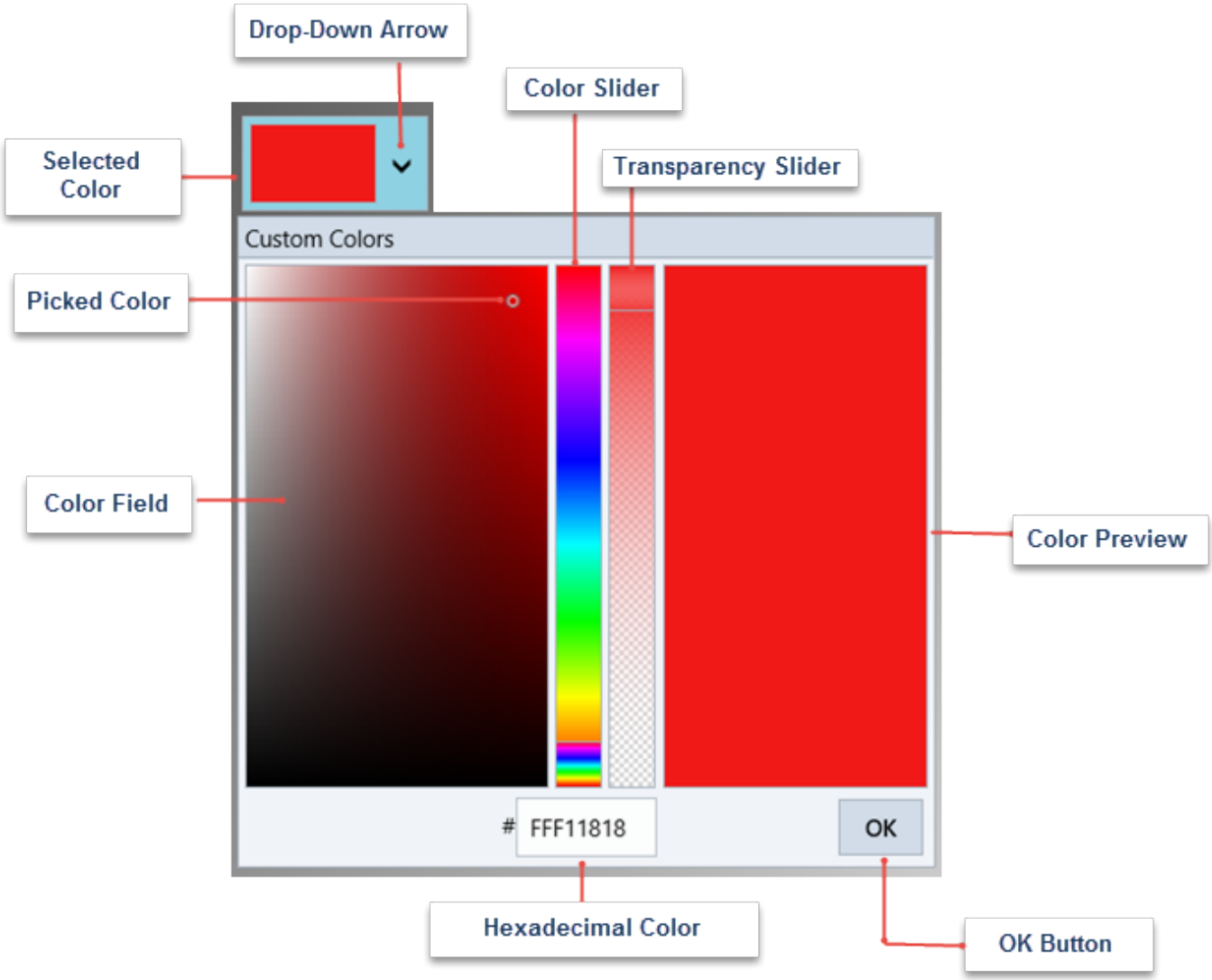


The Basic tab includes:

- **Drop-Down Arrow:** Click the drop-down arrow to open C1ColorPicker's window.
- **Selected Color:** The currently selected color appears the ColorPicker's window.
- **Picked Color:** The currently picked color appears with a highlighted border in the list of colors.
- **Palette Color:** Palette colors reflect the currently selected color palette. You can choose a palette by setting the Palette property.
- **Header Palette:** These colors are the basic colors available in the selected palette. The expanded list of palette colors are simply variations of these basic color tones.
- **Standard Colors:** Lists out ten standard colors, including dark brick red, red, orange, yellow, light green, green, sky blue, blue, navy blue and purple.
- **Recent Colors:** Lists up to ten recently used colors. You can choose to hide recent colors by setting the ShowRecentColors property to **False.**

Advanced Mode

The **Advanced** mode appears in the Advanced tab, which is placed on the right side of the Basic tab.
This mode enables users to customize the color palette by selecting various color tones available with every single color. This mode also supports **RGB Colors**, which can be viewed in C1ColorPicker's **Properties** window in **Design** view. The **Advanced** tab appears similar to the following image.



The **Advanced** tab mainly includes:

- **Drop-Down Arrow:** Click the drop-down arrow to open C1ColorPicker's window.
- **Color Slider:** Color slider lets you select color from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune your selection in the Color Field.
- **Color Field/Picked Color:** The **Color Field** enables you to select a color tone. The **Picked Color** indicates the currently selected color.
- **Transparency Slider:** This slider allows you to set the color's transparency, which can be set to opaque or partially/completely transparent.
- **Color Preview:** This component enables you to preview the selected color.
- **OK Button:** Once you are satisfied with the color choice, click the **OK** button to close the drop down and set it as the selected color.

Additional Controls

C1SpectrumColorPicker**: Colorpicker for UWP** includes **C1SpectrumColorPicker** control that allows you to access the advanced color picking functionality. The C1SpectrumColorPicker appears similar to the following

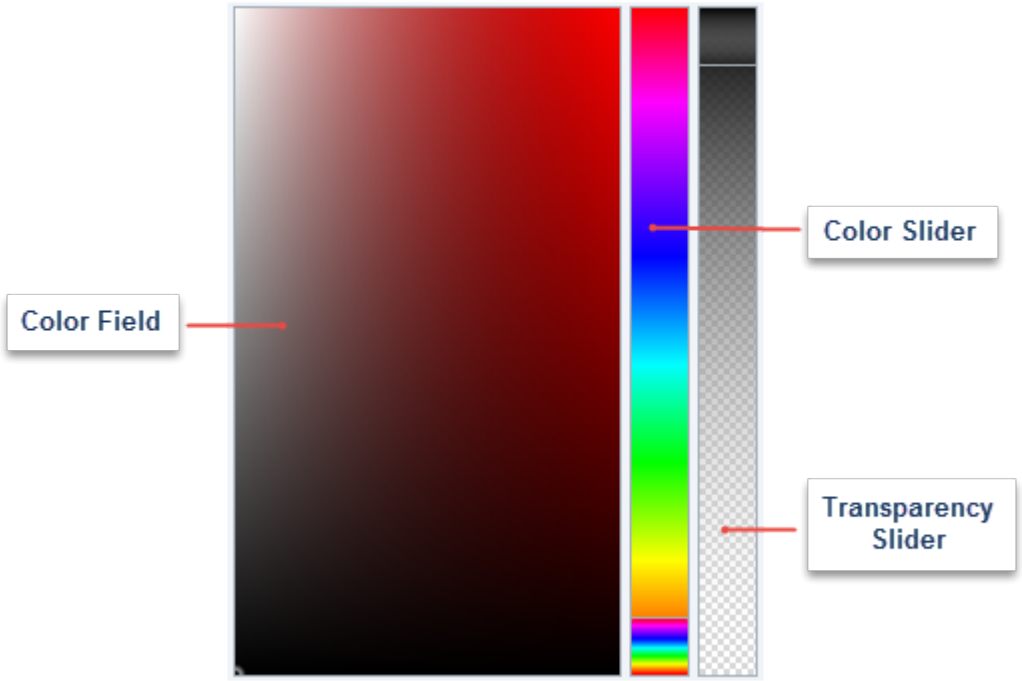


image.

The C1SpectrumColorPicker control includes:

- **Color Field:** The **Color Field** enables you to select a color tone.
- **Color Slider:** Color slider lets you select color from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune your selection in the Color Field.
- **Transparency Slider:** This slider allows you to set the color's transparency, which can be set to opaque or partially/completely transparent. This slider is visible only when the ShowAlphaChannel property is set to **True** (default)**.**

C1HexColorBox**: Colorpicker for UWP** includes **C1HexColorBox** control for data validation of hexadecimal code entries. Similar to a regular textbox in appearance, C1HexColorBox control displays an 8-character code, wherein the first two characters represents the color's transparency level. For instance, FF represents opaque and 00 represents transparent. The remaining 6-characters represent standard hexadecimal color selection. C1HexColorBox appears similar to the following image.

ColorPicker Quick Reference

This topic is dedicated to providing a quick overview of the XAML code used to create a C1ColorPicker**,** with select properties such as DropDownDirection set to AboveOrBelow, **Mode** to Basic, and Background to Red. For more information, see Features section.

XAML

```
<Extended:C1ColorPicker x:Name="C1ColorPicker1" HorizontalAlignment="Left"
VerticalAlignment="Top"
 Margin="100,73,0,0" Height="77" Width="170" DropDownDirection="AboveOrBelow"
Background="Red"
 Mode="Basic"/>
```

Quick Start

This topic describes how to add and configure a C1ColorPicker control in a UWP application. In this quick start, you will create a new Visual Studio project, add ColorPicker controls to your application and customize the appearance of controls. You will also explore select properties and features of C1ColorPicker by adding it on a standard Rectangle control. The C1ColorPicker will control the gradient applied to the Rectangle, so that choosing colors at runtime will change the gradient of the rectangle.

Step 1: Setting Up the Application

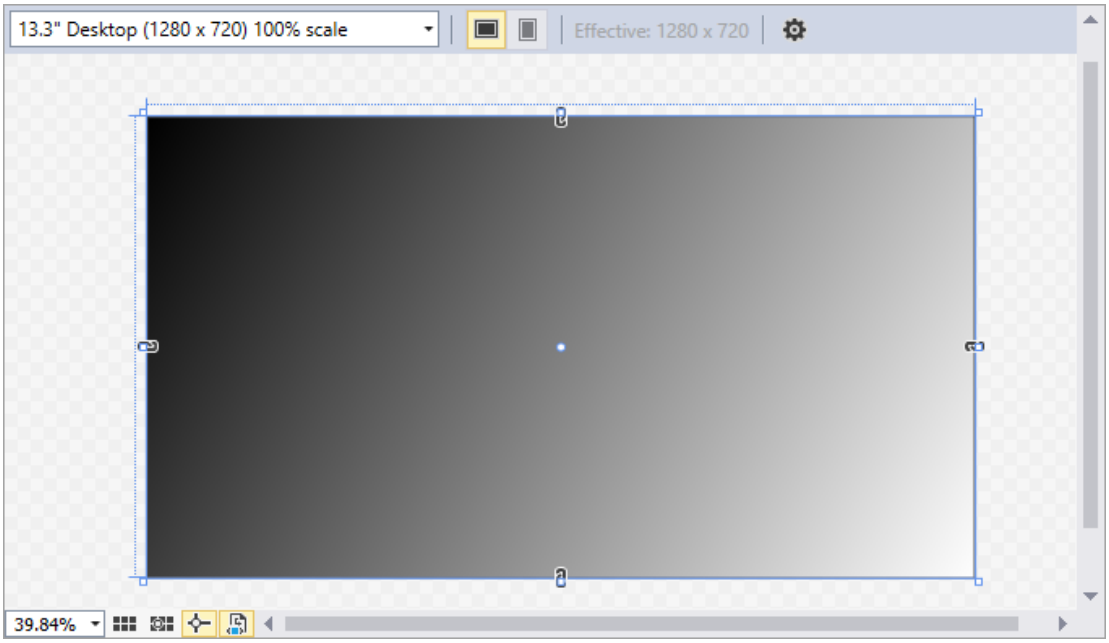
In this step, you will create a new UWP project and add a standard **Rectangle** control to it. You will also apply gradient to the added rectangle control to customize its appearance.

1. Open Visual Studio and select **File** | **New** | **Project**.
2. In **New Project** dialog box, expand a language in the left pane.
    - Under the chosen language, select **Windows Store**.
    - In the templates list, select **Blank App (XAML)**.
    - Enter a **Name** and click **OK** to create your project.
3. Switch to **Design** view, navigate to the Toolbox and double-click the **Rectangle** icon to add standard rectangle control to the grid.
4. Resize the window and expand the rectangle to fill the window.
5. Switch to **XAML** view again and add a Fill by replacing the <Rectangle> tag with the following code.

    XAML

    ```
    <Rectangle x:Name="Rectangle1" Stroke="Black">
      <Rectangle.Fill>
        <LinearGradientBrush x:Name="colors">
          <GradientStop x:Name="col1" Color="Black" Offset="0" />
          <GradientStop x:Name="col2" Color="White" Offset="1" />
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>
    ```

This adds a black and white linear gradient fill to the rectangle. The design view of the page should now appear similar to the following image.



With this, you have successfully created a UWP application, added a standard **Rectangle** control to it and customized its appearance. In the next step, you will add and customize C1ColorPicker control to your application.

Step 2: Adding C1ColorPicker Controls

In this step, you will continue by adding C1ColorPicker controls to the UWP application created in Step1. You will add two C1ColorPicker controls in order to control gradient fill in the existing Rectangle control.

1. In **Design** view, select the rectangle and navigate to the Visual Studio Toolbox.
2. In the Toolbox, locate and double-click **C1ColorPicker** icon twice to add two ColorPicker controls to the rectangle.
3. Resize the added **C1ColorPicker** controls and place them in the middle of the rectangle.
4. In **Design** view, click the first ColorPicker control, C1ColorPicker1, and navigate to the **Properties** window to set its properties as follows:
    - Set DropDownDirection property to **AboveorBelow** to control the direction in which the ColorPicker opens.
    - Set Mode property to **Advanced** to open advanced color options
    - Set SelectedColor property to **Black** or ("FF000000").
5. After setting these properties, the XAML view should appear similar to the follows:

    XAML

    ```
    <Extended:C1ColorPicker x:Name="C1ColorPicker1" HorizontalAlignment="Left"
    VerticalAlignment="Top" DropDownDirection="AboveOrBelow" Mode="Advanced"
    Margin="302,285,0,0" Width="275"
    SelectedColorChanged="C1ColorPicker1_SelectedColorChanged" Height="85"/>
    ```

6. Click on the second ColorPicker control, C1ColorPicker2, and set its **SelectedColor** property to **White**, leaving other properties as default. The page's design view would appear as below.

With this, you have successfully designed the user interface for your UWP application. However, on running the application, you will only see two C1ColorPicker controls on the output window. These controls will not perform any function even if you select a color. In the next step, you will add code to your application to provide functionality to the added ColorPicker controls.

Step 3: Adding Code to the Application

In this step, you will add code to your UWP application to provide functionality to the added C1ColorPicker controls. Since you have already designed the user interface for your application in the previous step, complete the following steps to add functionality.

1. In **Design** view, click once on the C1ControlPicker1 to select it and navigate to the **Properties** window.
2. In **Properties** window, select **Events** icon, locate SelectedColorChanged event and double-click in the text area.
3. This opens the code view (MainPage.xaml.cs) for the selected control with an event handler created as **C1ColorPicker1_SelectedColorChanged**.
4. Ensure that the following import statements are added to the top in the code.

Visual Basic

```
Imports C1.Xaml
Imports C1.Xaml.Extended
```

C#

```
using C1.Xaml;
using C1.Xaml.Extended;
```

5. To update the gradient values and subscribe **SelectedColorChanged** event handler for **C1ColorPicker1**, add the following code just below the MainPage's constructor in code view (MainPage.xaml.cs).

Visual Basic

```
Private Sub UpdateGradient()

    If C1ColorPicker1 IsNot Nothing And C1ColorPicker2 IsNot Nothing Then

        Me.col1.Color = Me.C1ColorPicker1.SelectedColor
```

```
        Me.col2.Color = Me.C1ColorPicker2.SelectedColor

    End If

End Sub


Private Sub C1ColorPicker1_SelectedColorChanged(sender As Object,
e As PropertyChangedEventArgs(Of Windows.UI.Color))Handles
C1ColorPicker1.SelectedColorChanged
    UpdateGradient()

End Sub
```

C#

```
void UpdateGradient()
{

  if (C1ColorPicker1 != null & C1ColorPicker2 != null)

  {

    this.col1.Color = this.C1ColorPicker1.SelectedColor;

    this.col2.Color = this.C1ColorPicker2.SelectedColor;

  }

}

private void C1ColorPicker1_SelectedColorChanged(object sender,
C1.Xaml.PropertyChangedEventArgs < Windows.UI.Color > e)
{
  UpdateGradient();
}
```

6. As done above in Step 1, 2 and 3, add and subscribe **SelectedColorChanged** event for the second ColorPicker control, C1ColorPicker2. When then event is created, update the gradient values through code as follows:

Visual Basic

```
Private Sub C1ColorPicker2_SelectedColorChanged(sender As Object,
e As PropertyChangedEventArgs(Of Windows.UI.Color)) Handles
C1ColorPicker2.SelectedColorChanged
    UpdateGradient()

End Sub
```

C#

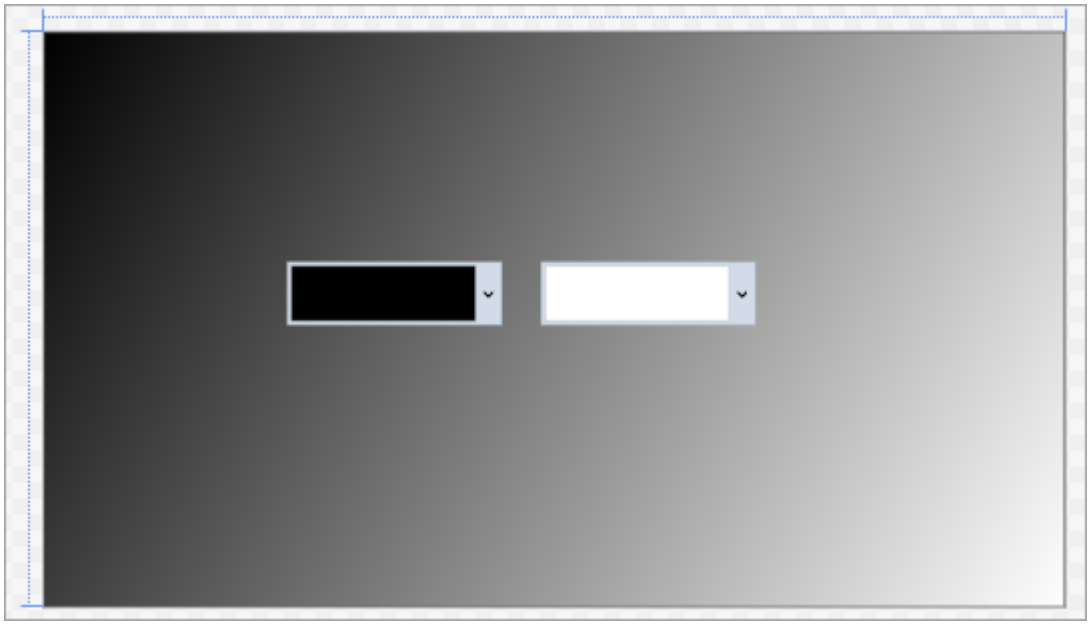
```
private void C1ColorPicker2_SelectedColorChanged(object sender,
PropertyChangedEventArgs < Windows.UI.Color > e)
{
  UpdateGradient();
}
```

With this, you have completed the addition of code to your UWP application and added functionality to the added ColorPicker controls. In the next step, you will run the application to see how the controls function at runtime.

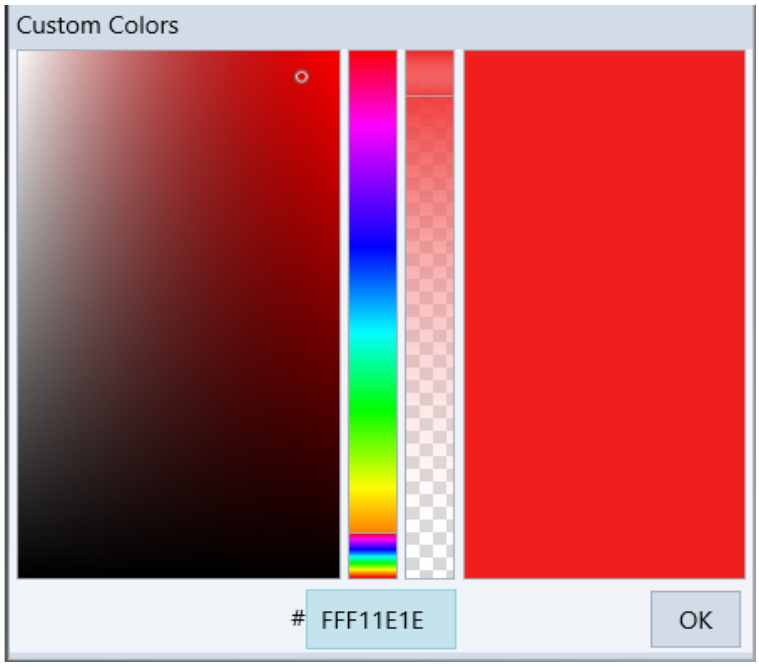
Step 4: Running the Application

As you are done with creating a UWP application that adds C1ColorPicker controls and customizes their appearance and behavior, it is time to run the application and observe the result.

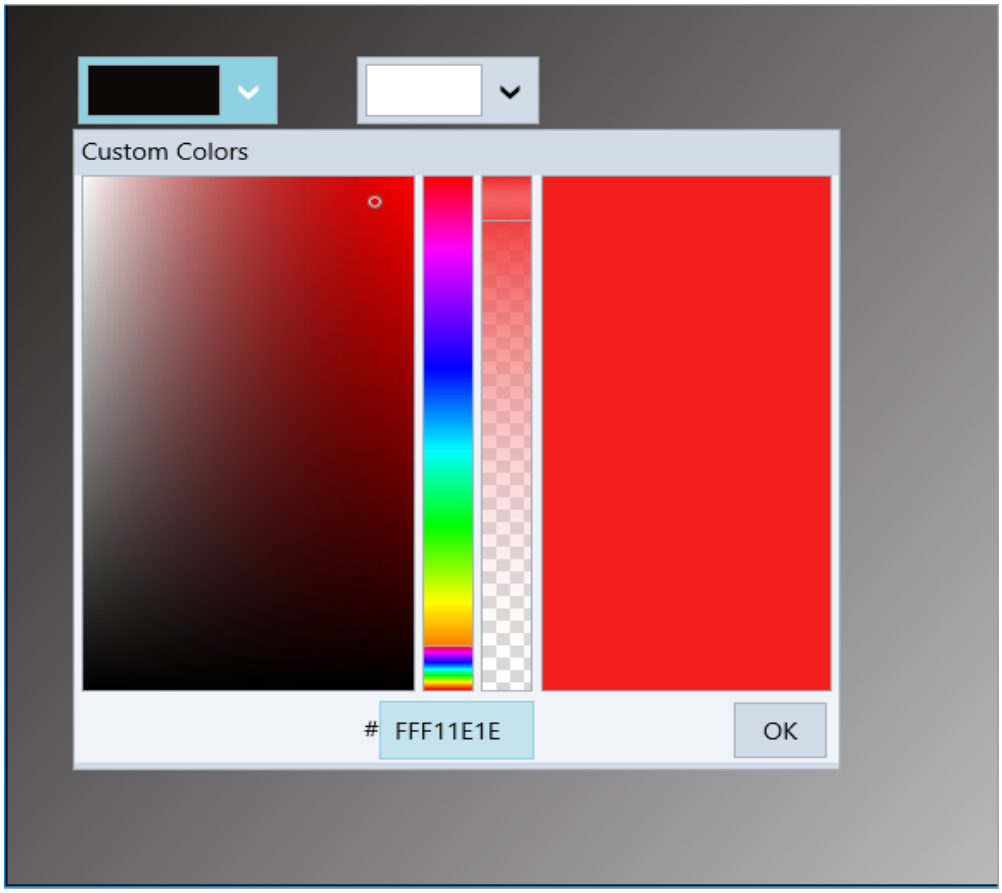
1. In **Debug** menu, select **Start Debugging** option to view the output. You will see a black and white window with two C1ColorPicker controls positioned in the middle of the screen as follows.



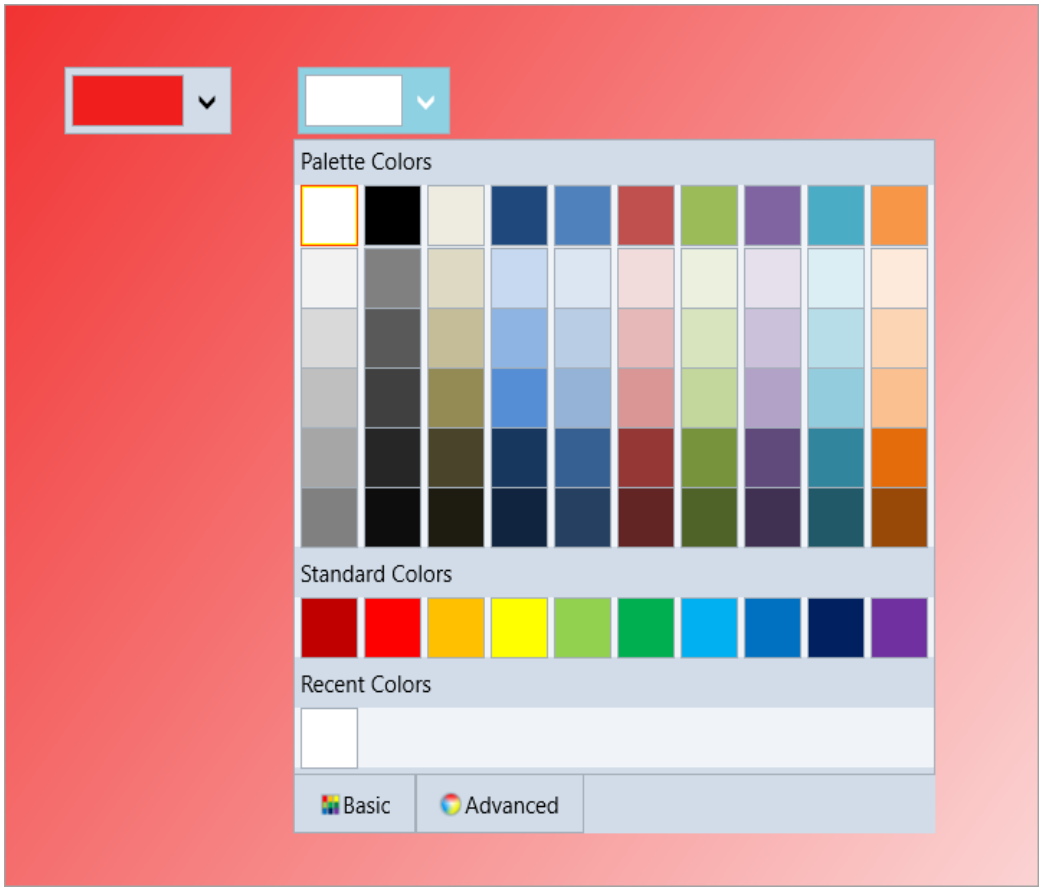
2. In the ColorPicker appearing on the left side, click the drop-down arrow to see that the drop-down box opens below with advanced mode visible, reflecting the changes you made to the control in **Step 2**.



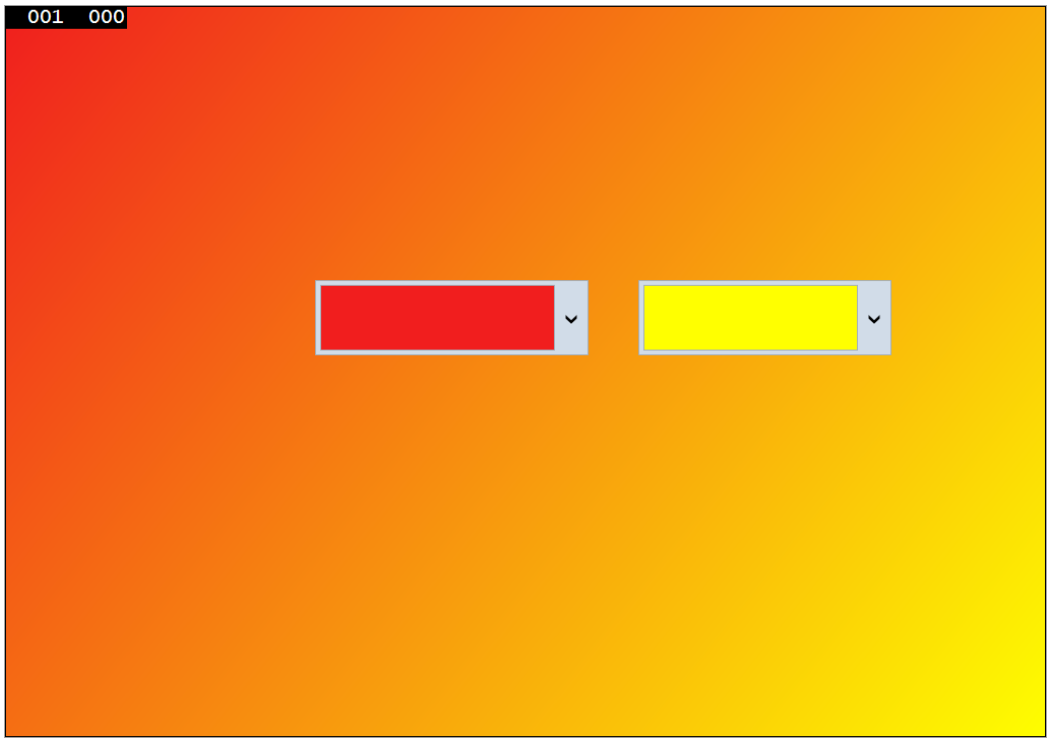
3. You can select a color, for instance Red, by various methods and click OK. Notice that the selected color and rectangle's gradient changes, reflecting your choice.



4. Click the drop-down arrow in the ColorPicker control appearing on the right side, C1ColorPicker2, to observe that the drop-down box opens in default **Basic** mode displaying tabs like **Color Palette**, **Standard Colors** and **Recent Colors.**



5. Pick a color, say Yellow, in the drop-down box to see that the rectangle's gradient changes as per your selection and would appear as below.



This complete the Quick Start section, wherein you created a simple application to add and customize the appearance of C1ColorPicker control, and observed some of its key features at runtime.

Features

This section elaborates the key features, properties and other important aspects related to the new **ColorPicker for UWP**.

Setting the Palette

ColorPicker for UWP comes with 20 pre-defined color palettes to match with themes available in Microsoft Office suite. To change the Color Palette, complete the following steps for seting up the Palette property of C1ColorPicker control for UWP.

1. In **Design** view, add standard Button control from Toolbox and set its **Content** property to **"Change Palette"**.
2. Switch to **Design** view again, and double-click the **Button** control. This opens the code view with a **Button_Click** event handler created for the same.
3. Add the following code to **Button_Click** event handler.

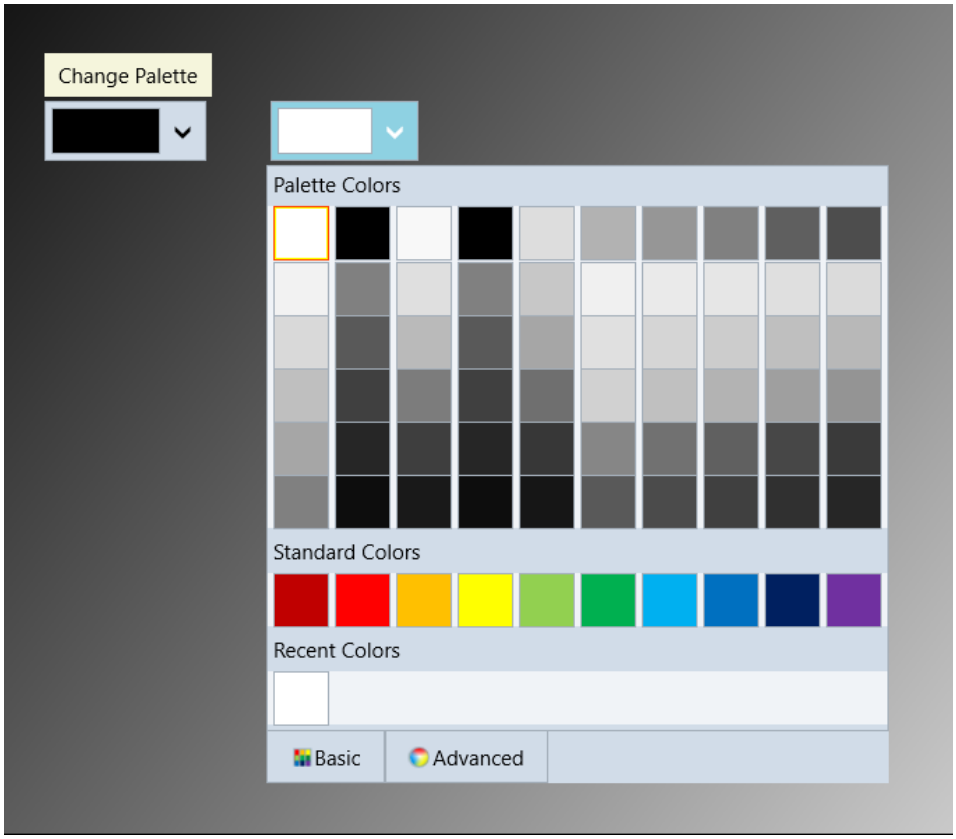
Visual Basic

```
Private Sub Button1_Click(sender As Object, e As RoutedEventArgs) Handles
Button1.Click
    Me.C1ColorPicker2.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale)
End Sub
```

C#

```
private void Button1_Click(object sender, RoutedEventArgs e)
{
    this.C1ColorPicker2.Palette =
ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale);
}
```

4. To run the application and observe the change in color palette, Select **Debug** menu and click **Start Debugging.**
5. In the runtime environment, click on the drop-down to observe the default palette with standard color options.
6. Now, click the Button visible above the C1Colorpicker and notice that a grayscale palette appears as per the change implemented in the code.



Customizing the Palette

Bored of using the predefined color palettes in ColorPicker within your application! You can choose to customize the color palette in **ColorPicker for UWP** by simply adding a few more lines to your code. Follow the steps given below to create a custom palette and apply it to the C1ColorPicker through a click of a button.

1. Navigate to the Toolbox, locate **Button** icon and double-click to add it to **Design** view in the project.
2. Resize the button and place it above the already added C1ColorPicker control in **Design** view.
3. Navigate to the **Properties** window and set button's **Content** property to **"Change Palette"**.
4. Switch to code view by double-clicking the button in **Design** view. This automatically creates the **Button_Click** event handler in the code.
5. Add the following Imports statements on the top of the code. Ignore in case they are already added.

Visual Basic

```
Imports C1.Xaml
Imports C1.Xaml.Extended
Imports Windows.UI
```

C#

```
using C1.Xaml;
using C1.Xaml.Extended;
using Windows.UI;
```

6. Add the following code to the **Button_Click** event to customize the palette.

Visual Basic

```
Private Sub Button1_Click(sender As Object, e As RoutedEventArgs) Handles
Button1.Click
```

```
    Dim cp1 As New ColorPalette("Pittsburgh")

    cp1.Clear()

    cp1.Add(Color.FromArgb(255, 0, 0, 0))

    cp1.Add(Color.FromArgb(255, 99, 107, 112))

    cp1.Add(Color.FromArgb(255, 255, 255, 255))

    cp1.Add(Color.FromArgb(255, 247, 181, 18))

    cp1.Add(Color.FromArgb(255, 253, 200, 47))

    cp1.Add(Color.FromArgb(255, 43, 41, 38))

    cp1.Add(Color.FromArgb(255, 149, 123, 77))

    cp1.Add(Color.FromArgb(255, 209, 201, 157))

    cp1.Add(Color.FromArgb(255, 0, 33, 71))

    cp1.Add(Color.FromArgb(255, 99, 177, 229))

    C1ColorPicker.Palette = cp1
End Sub
```

C#

```
private void Button1_Click(object sender, RoutedEventArgs e)
{
    ColorPalette cp1 = new ColorPalette("Pittsburgh");

    cp1.Clear();

    cp1.Add(Color.FromArgb(255, 0, 0, 0));

    cp1.Add(Color.FromArgb(255, 99, 107, 112));

    cp1.Add(Color.FromArgb(255, 255, 255, 255));

    cp1.Add(Color.FromArgb(255, 247, 181, 18));

    cp1.Add(Color.FromArgb(255, 253, 200, 47));

    cp1.Add(Color.FromArgb(255, 43, 41, 38));

    cp1.Add(Color.FromArgb(255, 149, 123, 77));

    cp1.Add(Color.FromArgb(255, 209, 201, 157));
```

```
    cp1.Add(Color.FromArgb(255, 0, 33, 71));

    cp1.Add(Color.FromArgb(255, 99, 177, 229));

    C1ColorPicker1.Palette = cp1;
}
```

7. Run the application. Observe that on clicking the C1ColorPicker's drop-down arrow, the default palette appears.
8. Now click the **Change Palette** button and again click the C1ColorPicker's drop-down arrow. You will find that the custom color palette appears.

Implementing Background Color

C1ColorPicker's Background property enables users to get or set the control's background color. In UWP applications, you can implement or change background color either at Design-time or in XAML view.

At Design Time

To implement background color for C1ColorPicker at design time, complete the following steps:

1. Select the C1ColorPicker control in **Design** view by a single click and navigate to the **Properties** window.
2. Locate **Background** property and select a Color, for instance Green, from the underlying **Editor** tab.

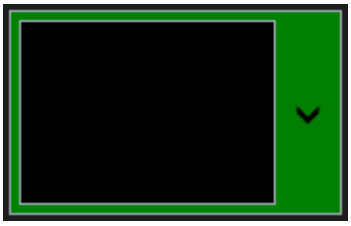
In XAML

To implement background color for C1ColorPicker through XAML, add **Background="Green"** to the <Extended:C1ColorPicker> tag. The XAML view should appear similar to the following:

XAML

```
<Extended:C1ColorPicker x:Name="C1ColorPicker1" Margin="283,119,883,0" Height="90"
 VerticalAlignment="Top" Background="Green"/>
```

Run the application and notice that the ColorPicker appears Green.



Changing Drop-Down Window Direction

By default, C1ColorPicker appears below the control when the user clicks the drop-down arrow at runtime. However, users can change the direction in which the control appears at runtime simply by making some simple changes either at Design-time, in XAML or in the code.

At Design Time

To change the drop-down window direction at design time:

1. Select the C1ColorPicker by clicking it once in **Design** view.
2. Navigate to the **Properties** window and locate the DropDownDirection property.
3. Click the drop-down arrow alongside the **DropDownDirection** property and select any option, for instance **ForceAbove**.

This will set the DropDownDirection property of ColorPicker control as per your selection.

In XAML

Change the drop-down window direction in XAML by adding DropDownDirection="ForceAbove" to the <Extended:C1ColorPicker> tag. The XAML view should appear similar to the following:

XAML

```
<Extended:C1ColorPicker x:Name="C1ColorPicker1" HorizontalAlignment="Left"
VerticalAlignment="Top"
 Margin="147,553,0,0" Height="77" DropDownDirection="ForceAbove"/>
```

In Code

Through code, you can customize the direction in which the drop-down window should appear by adding the following code below the constructor.

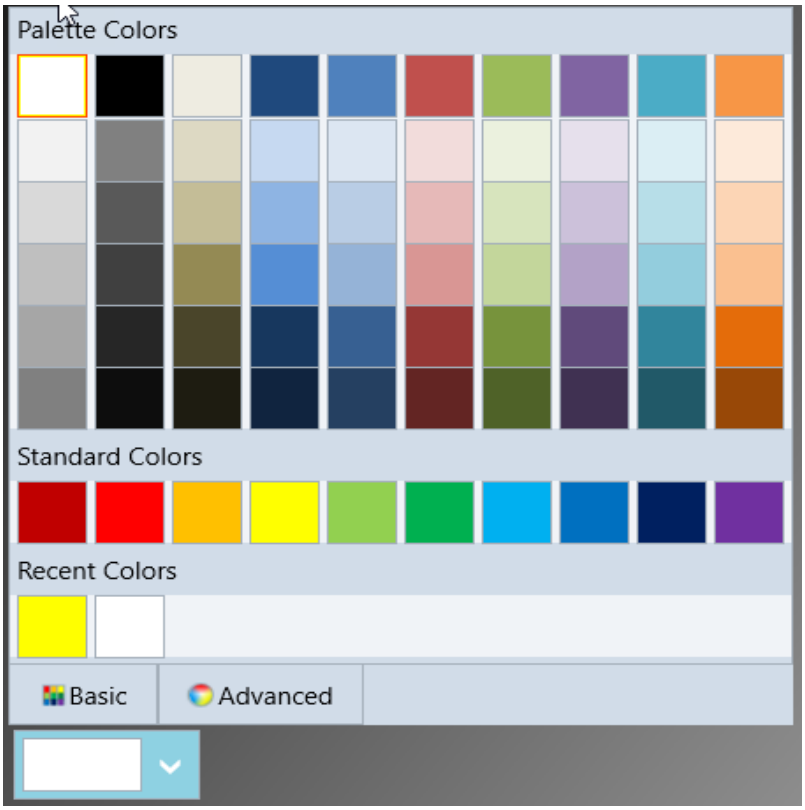
Visual Basic

```
Me.C1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove
```

C#

```
this.C1ColorPicker2.DropDownDirection = DropDownDirection.ForceAbove;
```

Run the application. Observe that on clicking the C1ColorPicker's drop-down arrow, the drop-down appears above the C1ColorPicker control similar to the following image.



Hiding Recent Colors Tab in Basic Mode

In **Basic** mode, C1ColorPicker displays a **Recent Colors** tab that displays the recent color icons selected on the ColorPicker. You can customize the appearance of C1ColorPicker by hiding Recent Colors tab either at Design time, in XAML or in code.

At Design Time

To hide Recent Colors tab at design time:

1. Select the C1ColorPicker Control by clicking it once in **Design** view.
2. Navigate to the **Properties** window and locate the ShowRecentColors property.
3. Check off the **ShowRecentColors** property to hide Recent Colors tab.

In XAML

To hide Recent Colors tab in XAML, set **ShowRecentColors** property to false by adding ShowRecentColors="False" in <Extended:C1ColorPicker> tag. The XAML code should appear similar to the follows:

XAML

```
<Extended:C1ColorPicker x:Name="C1ColorPicker2" HorizontalAlignment="Left"
 VerticalAlignment="Top" Margin="143,101,0,0" Width="179" Height="84"
ShowRecentColors="False"/>
```

In Code

In code view, you can hide Recent Colors tab by adding the following code in the MainPage constructor.

Visual Basic

```
Me.C1ColorPicker1.ShowRecentColors = False
```

C#

```
this.C1ColorPicker2.ShowRecentColors = false;
```

Run the application and notice that **Recent Colors** tab is no longer visible in the drop-down window.