



Microsoft®  
**Silverlight™**

## **Hands-On Lab**

---

*Silverlight 4 – Multi-Touch and Drop Targets*

## Contents

Introduction.....	3
Exercise 1 – Getting Started .....	4
Task 1 – Adding Drag and Drop Support .....	5
Task 2 – Creating Context Menu Custom Control.....	7
Task 3 – Native Right Mouse Click Support .....	15
Task 4 – Printing Support.....	20
Exercise 2 – Multi-touch on Windows 7.....	21
Task 1 – Enabling Multi-touch support.....	21
Conclusion .....	25

# Introduction

---

Rich media support is one of most compelling features provided by Silverlight. It enables you to create virtually any user interface (UI) and provide a rich user experience (UX). Silverlight 4 expands on its rich media experiences and help align it with some new line of business (LOB) oriented functionality.

In this lab you will learn how to create a Silverlight multimedia application:

1. Enriching the standard UX with multi-touch support (for Windows 7 client machines)
2. Add image file Drag and Drop functionality
3. Expose printing support
4. Enable native right mouse click events

In addition, this lab will demonstrate how to create a style-able and template-able Silverlight custom control. The lab application will provide an interface to display, add, remove and manipulate images and will enhance an existing “Silverlight Surface” demo application.

*Estimated completion time for this lab is 60 minutes.*

# Exercise 1 – Getting Started

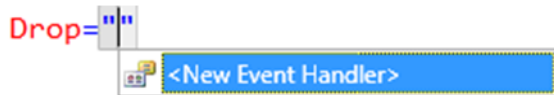
---

The goal of this exercise is to familiarize the student with the existing starter application and enhance it with new features (Drag and Drop, Native Right Mouse Click support, and printing support).

1. Start Visual Studio 2010
  2. On the **File** menu click **Open → Project/Solution...**
    - a. Alternatively, from Visual Studio Start Page click “**Open Project...**”
  3. At “Open Project” dialog navigate to the Lab installation folder
  4. Navigate to “MultiTouchAndDropTargets\Source\Ex01-GettingStarted\begin” folder
  5. Click “ImageGallery.sln” file and the click “**Open**” button
  6. Take some time to familiarize yourself with the Starter application
    - a. Points of interest here are:
      - ◆ Photo.xaml
      - ◆ Photo.xaml.cs
      - ◆ MainPage.xaml.cs
  7. Set the ImageGallery.Web project as the startup project, by right clicking the project in the Solution Explorer and selecting “Set as Startup Project”
  8. Press **F5** or click **Debug → Start Debugging** to Run the application
  9. Use the application to familiarize yourself with it. When finished close the browser window
-

### Task 1 – Adding Drag and Drop Support

1. If you have not opened the Starter project please open it now (see previous section for detailed instructions)
2. Open MainPage.xaml (double click on the filename in the Solution Explorer)
3. Add **AllowDrop="True"** to the UserControl just before the closing of the tag ">"
4. Add a new **"Drop"** event handler (use the default name)



**Figure 1**

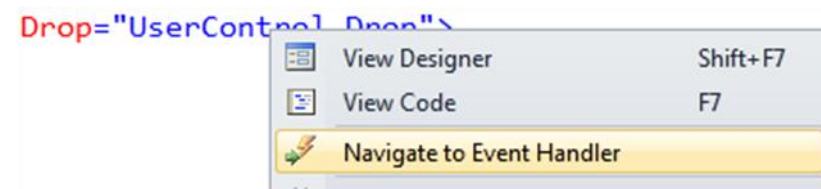
*Event Handler Generation from XAML Editor*

5. The resulting state of UserControl tag should look like follows

#### XAML

```
<UserControl x:Class="ImageGallery.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
AllowDrop="True" Drop="UserControl_Drop">
```

6. Right click on "UserControl\_Drop" and from the context menu choose "Navigate to Event Handler"



**Figure 2**

*Navigate to Event Handler from XAML Editor*

7. The last action will take you to the source code editor, to the event handler function:

#### C#

```
private void UserControl_Drop(object sender, DragEventArgs e)
{
}
}
```

8. To receive a dropped object (that is dragged onto the designer surface of the UIElement from outside the Silverlight application), use the **Data** property of **DragEventArgs**. Drag and Drop is capable of supporting any data file format. In this lab we create an image file Drag and Drop mechanism, so we are interested in Data from the "FileDrop" format. The user could drop more

than one file, in which case we will handle the “FileDrop” data as a list of files. To get the **IDataObject** and check if such data is present, add the following lines inside the function body:

```
C#  
IDataObject theDo = e.Data;  
  
if (theDo.GetDataPresent("FileDrop"))  
{  
    FileInfo[] files = theDo.GetData("FileDrop") as FileInfo[];  
}
```

9. Now, after getting the list of dropped files (a user may drop more than one file), iterate over the list, checking that the file matches the supported extension (in our case we will support JPG and PNG only), then open the file and create a **BitmapImage** instance from it. Paste the following lines inside the “if” block, after the **FileInfo[] files = ...** line:

```
C#  
foreach (var file in files)  
{  
    if (file.Name.ToLower().Contains(".jpg") ||  
        file.Name.ToLower().Contains(".png"))  
    {  
        FileStream reader = file.OpenRead();  
  
        byte[] array = new byte[reader.Length];  
        int count = reader.Read(array, 0, (int)reader.Length);  
        MemoryStream str = new MemoryStream(array);  
        BitmapImage img = new BitmapImage();  
        img.SetSource(str);  
  
        //Initialize new instance of Photo with received image  
    }  
}
```

10. The last thing to do is to initialize a new instance of the Photo class with the received image. The implementation of Photo receives only the filename and loads it from the originating web server. In order to support a dropped image, add an overloaded constructor to the Photo class. Open Photo.xaml.cs and add following code to the class to create the overloaded constructor:

```
C#  
public Photo(MainPage parent, BitmapImage photo, string name)  
{  
    InitializeComponent();  
  
    initializePhoto(parent);  
    _name = name;  
    image.Source = photo;
```

```
// Position and rotate the photo randomly
Translate(random.Next((int)(this._parent.ActualWidth - Width)),
random.Next((int)(this._parent.ActualHeight - Height)));
Rotate(random.Next(-30, 30));

// Add the photo to the parent and play the display animation
_parent.LayoutRoot.Children.Add(this);

display.Begin();
}
```

11. Go back to the MainPage.xaml.cs and create new instance of Photo and call the overloaded constructor. To do it add the following lines of code right after the “//Initialize new instance...” remark:

```
C#
new Photo(this, img, file.Name);
```

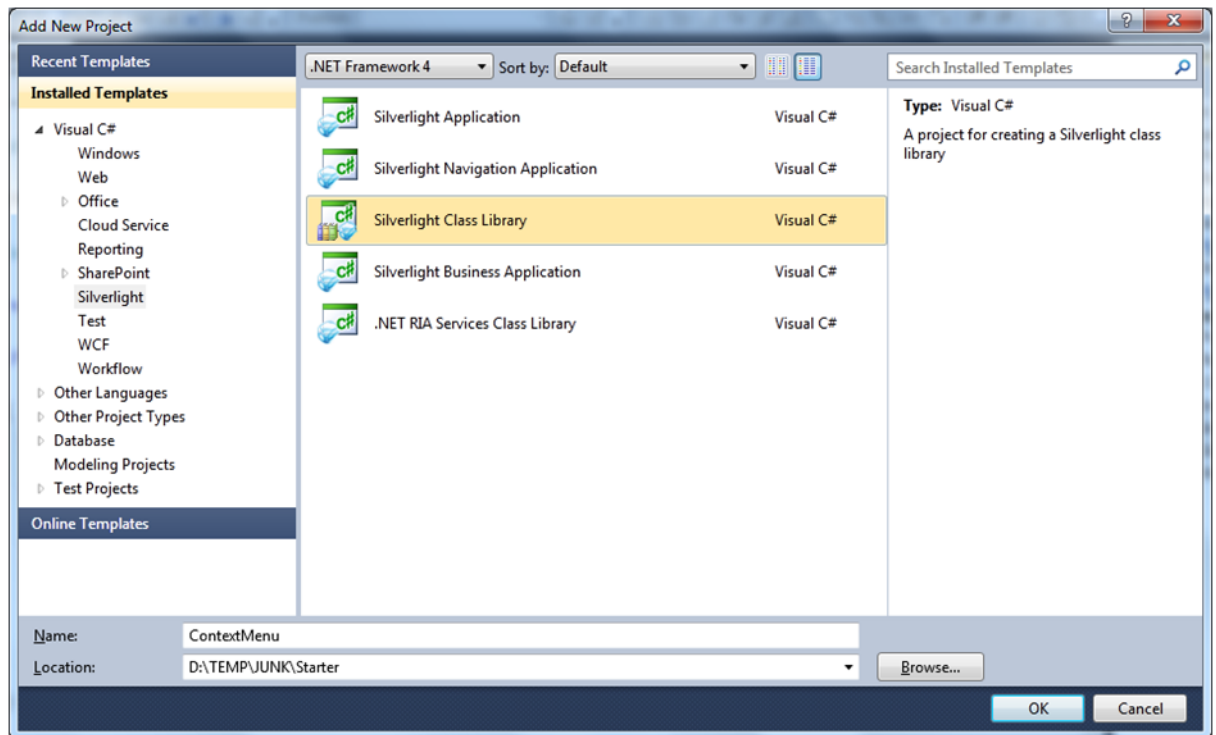
12. Compile and Run the application
13. Open a new Windows Browser, locate and navigate to the “Sample Pictures” folder (or any other folder with pictures). Select one or more pictures from this folder, and Drag and Drop them to the surface of the running Lab application.

---

## Task 2 – Creating Context Menu Custom Control

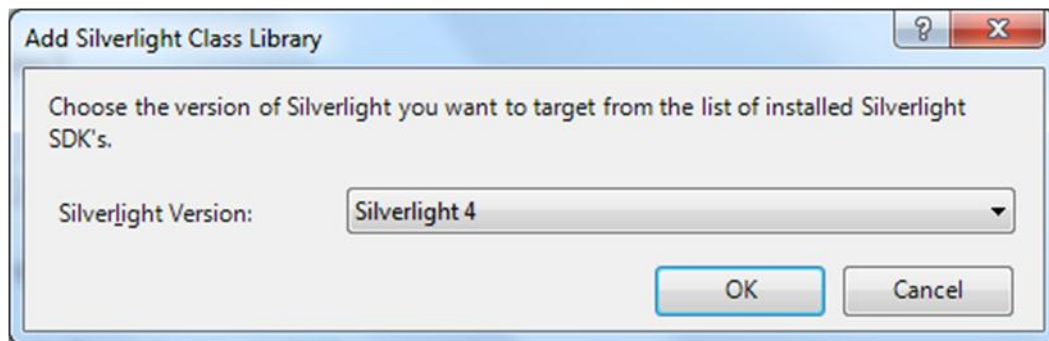
In order to remove the pictures from the application’s surface and print the images collage, we need to provide an image context menu.

1. If the Lab application is still running, stop it now.
2. Add a new project to the solution by right clicking on Solution (in Solution Explorer) and select **Add** → **New Project...**
3. At the “Add New project” dialog, select “Silverlight Class Library” item, and name it ContextMenu



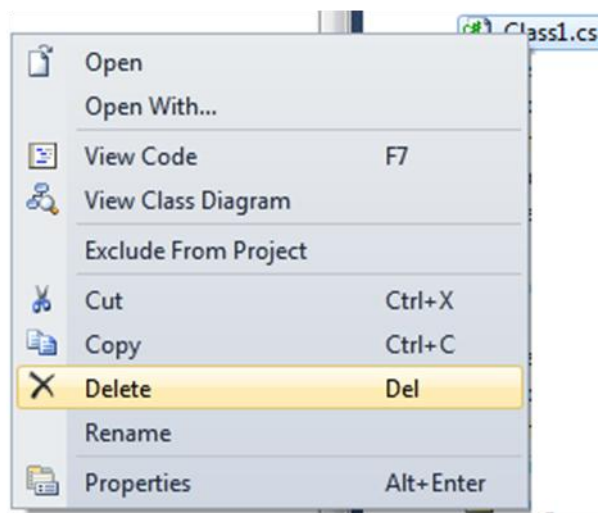
**Figure 3**  
*Add New Project Dialog*

4. Click OK when done
5. Make sure that the correct version of Silverlight is selected at "Add Silverlight Class Library" dialog and click OK



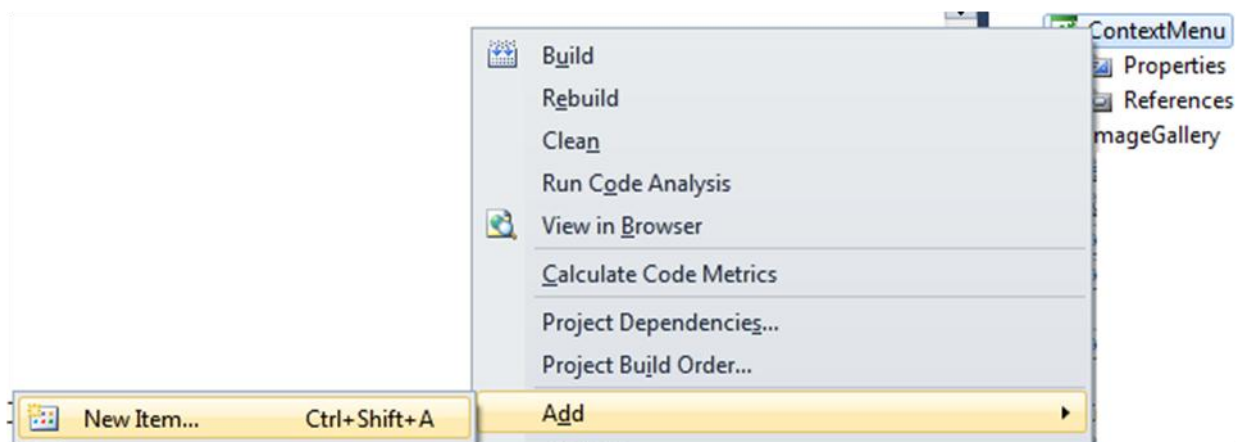
**Figure 4**  
*Silverlight Version Selector*

6. Delete the Class.cs from the project by right clicking on it and selecting **Delete**

**Figure 5**

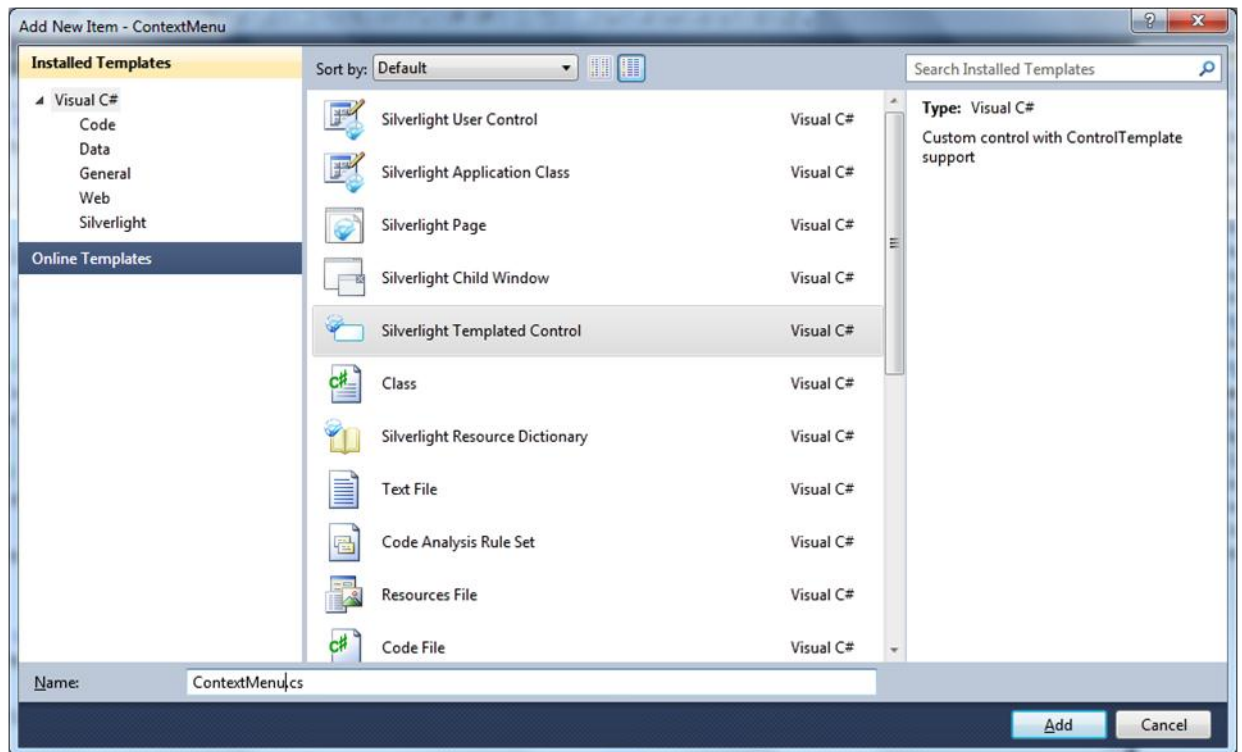
*Delete Default Class*

7. Add a new item into the ContextMenu project item by right clicking on the project and selecting **Add → New Item...**

**Figure 6**

*Add New Item To the Project*

8. From “Add New Item - ContextMenu” dialog select “Silverlight Templated Control”, name it ContextMenu.cs and click “Add”



**Figure 7**

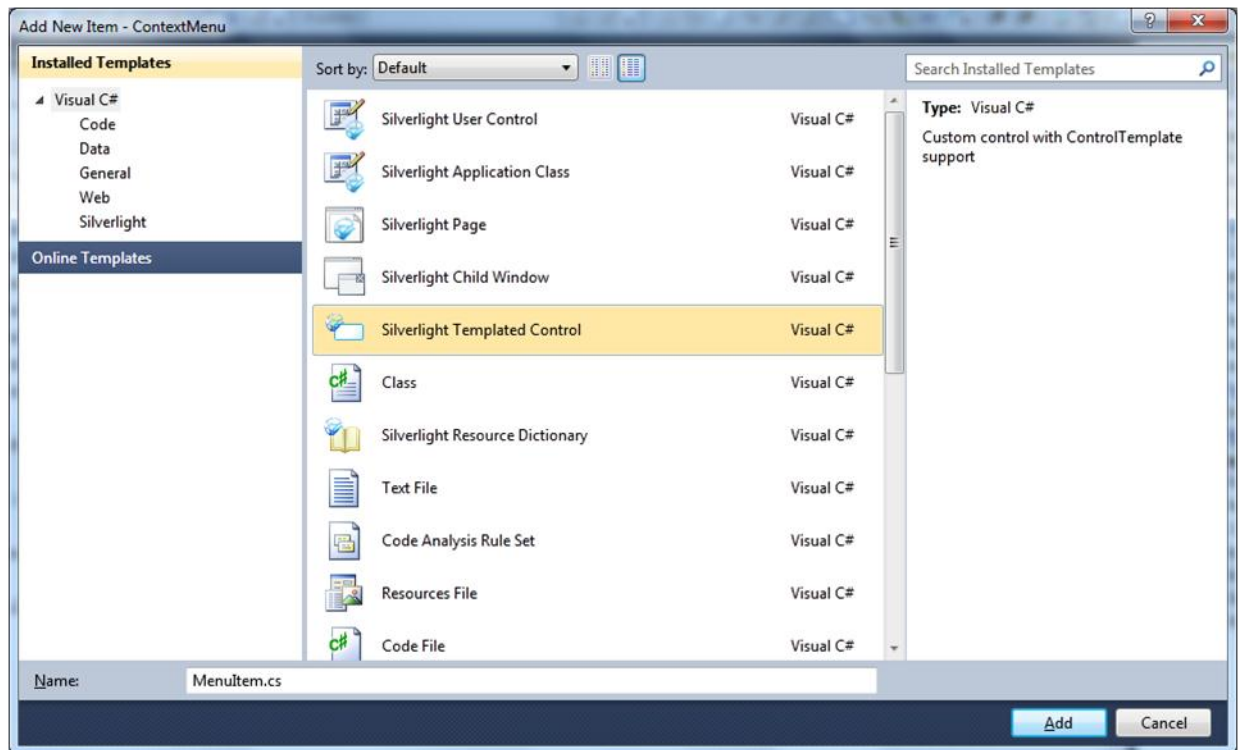
*Add New Item Dialog Box*

9. Change the base class for the ContextMenu class from "Control" to "ItemsControl":

**C#**

```
public class ContextMenu : ItemsControl
```

10. Add additional "Silverlight Templated Control" to the project and name it MenuItem.cs



**Figure 8**

*Add New Item Dialog Box*

11. Declare an additional namespace reference – add the following code after the last “using” statement at the beginning of the class (before the namespace):

```
C#
using System.Windows.Controls.Primitives;
```

12. Change the base class for created MenuItem from “Control” to “ButtonBase”:

```
C#
public class MenuItem : ButtonBase
```

13. Open the Generic.xaml file located in the “Themes” folder (automatically created by Visual Studio while adding new Templated Controls)

14. Locate the Style for ContextMenu, and add the following content inside the ControlTemplate’s **Border** markup (add the markup inside the Border element):

```
XAML
<Border.Effect>
    <DropShadowEffect ShadowDepth=".2" Opacity="1"/>
</Border.Effect>
<ItemsPresenter x:Name="ItemsPanel" Margin="3"/>
```

15. Locate the style for MenuItem and replace the **Border** in ControlTemplate with following markup:

#### XAML

```
<Grid x:Name="LayoutRoot">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="Normal"/>
      <VisualState x:Name="MouseOver">
        <Storyboard>
          <DoubleAnimationUsingKeyFrames
            Storyboard.TargetName="BackgroundAnimation"
            Storyboard.TargetProperty="Opacity">
            <SplineDoubleKeyFrame KeyTime="0" Value="1"/>
          </DoubleAnimationUsingKeyFrames>
          <ColorAnimationUsingKeyFrames
            Storyboard.TargetName="BackgroundGradient"
            Storyboard.TargetProperty="(Rectangle.Fill).
            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
            <SplineColorKeyFrame KeyTime="0" Value="#F2FFFFFF"/>
          </ColorAnimationUsingKeyFrames>
          <ColorAnimationUsingKeyFrames
            Storyboard.TargetName="BackgroundGradient"
            Storyboard.TargetProperty="(Rectangle.Fill).
            (GradientBrush.GradientStops)[2].(GradientStop.Color)">
            <SplineColorKeyFrame KeyTime="0" Value="#CCFFFFFF"/>
          </ColorAnimationUsingKeyFrames>
          <ColorAnimationUsingKeyFrames
            Storyboard.TargetName="BackgroundGradient"
            Storyboard.TargetProperty="(Rectangle.Fill).
            (GradientBrush.GradientStops)[3].(GradientStop.Color)">
            <SplineColorKeyFrame KeyTime="0" Value="#7FFFFFFF"/>
          </ColorAnimationUsingKeyFrames>
        </Storyboard>
      </VisualState>
      <VisualState x:Name="Disabled">
        <Storyboard>
          <DoubleAnimationUsingKeyFrames
            Storyboard.TargetName="DisabledVisualElement"
            Storyboard.TargetProperty="Opacity">
            <SplineDoubleKeyFrame KeyTime="0" Value=".55"/>
          </DoubleAnimationUsingKeyFrames>
        </Storyboard>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
  <Border x:Name="Background" CornerRadius="3"
    Background="White"
    BorderThickness="{TemplateBinding BorderThickness}"
    BorderBrush="{TemplateBinding BorderBrush}">
```

```

<Grid Background="{TemplateBinding Background}" Margin="0">
  <Border Opacity="0" x:Name="BackgroundAnimation"
    Background="#FF448DCA" />
  <Rectangle x:Name="BackgroundGradient" >
    <Rectangle.Fill>
      <LinearGradientBrush StartPoint=".7,0" EndPoint=".7,1">
        <GradientStop Color="#FFFFFF" Offset="0" />
        <GradientStop Color="#F9FFFF" Offset="0.375" />
        <GradientStop Color="#E5FFFF" Offset="0.625" />
        <GradientStop Color="#C6FFFF" Offset="1" />
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
</Grid>
</Border>

<StackPanel Orientation="Horizontal" Margin="2">
  <Image x:Name="MenuItemIcon" Width="20" Height="20"
    Source="{Binding MenuItemImage}" Margin="0,0,2,0"/>
  <ContentPresenter x:Name="MenuItemContent"
    Content="{Binding ItemContent}"
    Margin="0,0,0,1" VerticalAlignment="Center"/>
</StackPanel>
<Rectangle x:Name="DisabledVisualElement" RadiusX="3"
  RadiusY="3" Fill="#FFFFFF" Opacity="0"
  IsHitTestVisible="false" />
</Grid>

```

16. Open MenuItem.cs
17. Add the following **TemplatePart** class attributes to the class definition (just before the class declaration):

```

C#
[TemplatePart(Name = "LayoutRoot", Type = typeof(Grid)),
TemplatePart(Name = "MenuItemIcon", Type = typeof(Image)),
TemplatePart(Name = "MenuItemContent", Type = typeof(ContentPresenter))]

```

18. Create private variables to hold the instances of template parts. Add following code in the class (before the constructor code):

```

C#
Grid layoutRoot;
Image itemImage;
ContentPresenter itemContent;

```

19. Set the **DataContext** of the control to enable self binding. Add the code to the Constructor, right after the first (and only) line:

```

C#

```

```
this.DataContext = this;
```

20. Create a Dependency Property “ItemContent” of type string. This will be used to bind to the content for the menu item. Add the following code to the class, after the constructor:

**C#**

```
public string ItemContent
{
    get { return (string)GetValue(ItemContentProperty); }
    set { SetValue(ItemContentProperty, value); }
}

public static readonly DependencyProperty ItemContentProperty =
```

21. `DependencyProperty.Register("ItemContent", typeof(string), typeof(MenuItem), null);`

22. Create an additional Dependency Property “MenuItemImage” of type ImageSource. This will be used to bind to the image for the menu item. Add the following code after the previously added lines:

**C#**

```
public ImageSource MenuItemImage
{
    get { return (ImageSource)GetValue(MenuItemImageProperty); }
    set { SetValue(MenuItemImageProperty, value); }
}

public static readonly DependencyProperty MenuItemImageProperty =
```

23. `DependencyProperty.Register("MenuItemImage", typeof(ImageSource), typeof(MenuItem), null);`

24. Create a helper function to change the Visual State of the control when the IsEnabled property changes. Add the following code after the created dependency properties:

**C#**

```
private void HandleEnabledVisualAid()
{
    if (!this.IsEnabled)
        VisualStateManager.GoToState(this, "Disabled", true);
    else
        VisualStateManager.GoToState(this, "Normal", true);
}
```

25. Override the “OnApplyTemplate” function. This function will initialize template part variables and subscribe and handle the control events like MouseEnter and MouseLeave by applying visual states to them. Add the following code at the bottom of the class:

**C#**

```
public override void OnApplyTemplate()
{
    base.OnApplyTemplate();

    layoutRoot = GetTemplateChild("LayoutRoot") as Grid;
    itemImage = GetTemplateChild("MenuItemIcon") as Image;
    itemContent = GetTemplateChild("MenuItemContent")
        as ContentPresenter;

    layoutRoot.MouseEnter += (s, e) =>
    {
        VisualStateManager.GoToState(this, "MouseOver", true);
    };

    layoutRoot.MouseLeave += (s, e) =>
    {
        VisualStateManager.GoToState(this, "Normal", true);
    };

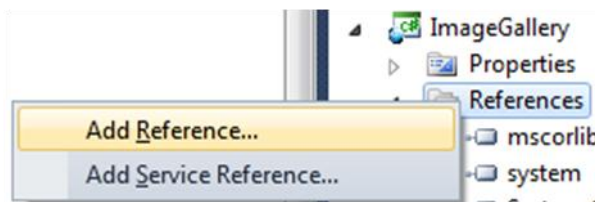
    this.IsEnabledChanged += (s, e) =>
    {
        HandleEnabledVisualAid();
    };

    HandleEnabledVisualAid();
}
```

26. Compile the solution.

### Task 3 – Native Right Mouse Click Support

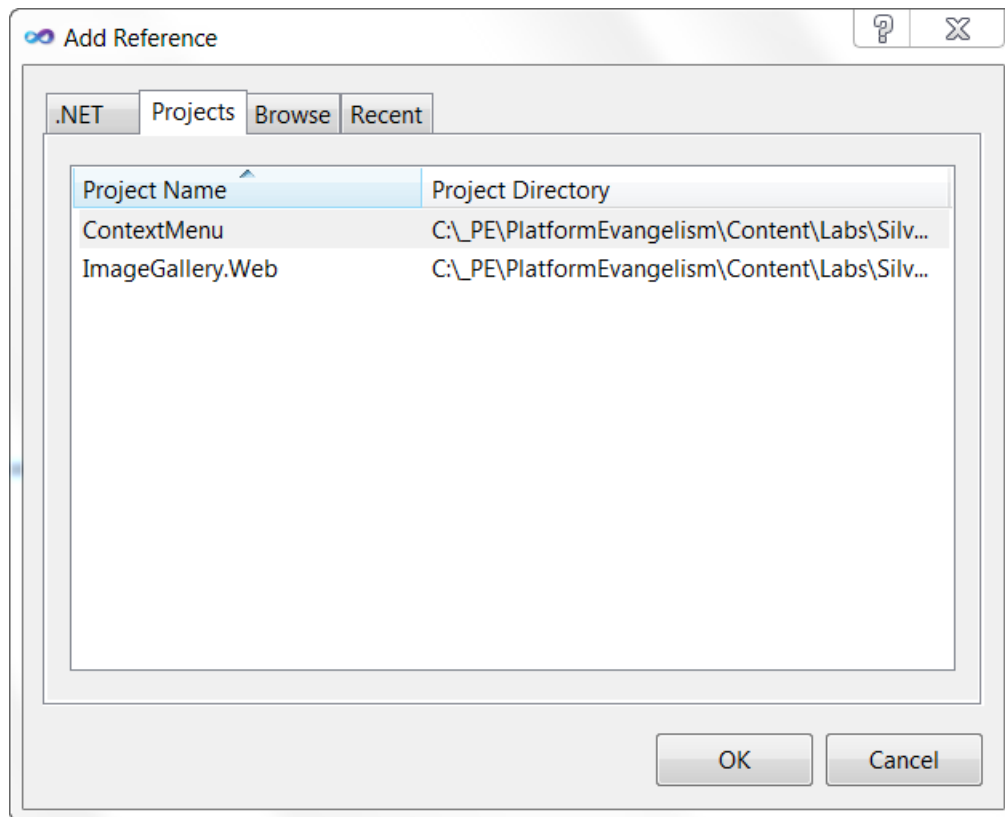
1. Back to the ImageGallery project - add the reference to the created custom control to the ImageGallery project by right clicking the References → Add Reference...



**Figure 9**

*Add Reference to the Project*

2. At "Add Reference" dialog click on the "Projects" tab and select ContextMenu



**Figure 10**

*Reference Selection Dialog Box*

- Open MainPage.xaml and add a **“MouseRightButtonDown”** event handler to the canvas named “LayoutRoot”. Accept the default name of the handler function. Resulting markup should look as follows:

#### XAML

```
<Canvas x:Name="LayoutRoot" Background="Gray"
MouseRightButtonDown="LayoutRoot_MouseRightButtonDown">

</Canvas>
```

- Add a **“MouseLeftButtonDown”** event handler to the UserControl. Accept the default name for the handler function. Resulting markup should look as follows:

#### XAML

```
<UserControl x:Class="ImageGallery.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
AllowDrop="True" Drop="UserControl_Drop"
MouseLeftButtonDown="UserControl_MouseLeftButtonDown">
```

5. Switch to the code (MainMenu.xaml.cs)
6. Import the control's namespace – add the following line after the last “using” statement in the class (before the namespace):

```
C#  
using ContextMenu;
```

7. Add a class level variable of type ContextMenu and name it contextMenu:

```
C#  
ContextMenu.ContextMenu contextMenu;
```

8. Navigate to the created event handlers. In the body of “UserControl\_MouseLeftButtonDown” function, add the following code to close the context menu when user clicks away from it:

```
C#  
if (null != contextMenu)
```

9.     LayoutRoot.Children.Remove(contextMenu);
10. Create helper function to generate and show the context menu by adding following code to the class:

```
C#  
private void GenerateContextMenu(Point pos)  
{  
    contextMenu = new ContextMenu.ContextMenu();  
    contextMenu.SetValue(Canvas.LeftProperty, pos.X);  
    contextMenu.SetValue(Canvas.TopProperty, pos.Y);  
    contextMenu.Visibility = Visibility.Visible;  
  
    MenuItem menuItem1 = new MenuItem();  
    menuItem1.ItemContent = "Delete";  
    menuItem1.MenuItemImage = new BitmapImage(new  
Uri("Images/Delete.png", UriKind.RelativeOrAbsolute));  
    menuItem1.Click += DeleteMenuItem_Click;  
    if (null == _lastActivePhoto)  
        menuItem1.IsEnabled = false;  
    contextMenu.Items.Add(menuItem1);  
  
    MenuItem menuItem2 = new MenuItem();  
    menuItem2.ItemContent = "Print";  
    menuItem2.MenuItemImage = new BitmapImage(new  
Uri("Images/Print.png", UriKind.RelativeOrAbsolute));  
    menuItem2.Click += PrintMenuItem_Click;  
    contextMenu.Items.Add(menuItem2);  
}
```

```
LayoutRoot.Children.Add(contextMenu);
}
```

11. Create an event handler function for the Delete command – add the following lines to the class:

```
C#
private void DeleteMenuItem_Click(object sender, RoutedEventArgs e)
{
    LayoutRoot.Children.Remove(contextMenu);
    LayoutRoot.Children.Remove(_lastActivePhoto);
}
```

12. Create an event handler function for the Print command – add the following lines to the class:

```
C#
private void PrintMenuItem_Click(object sender, RoutedEventArgs e)
{
}
```

13. Add the following line to the “LayoutRoot\_MouseRightButtonDown” function body:

```
C#
e.Handled = true;
Point pos = e.GetPosition(null);

if (null != contextMenu)
    LayoutRoot.Children.Remove(contextMenu);

GenerateContextMenu(pos);
```

14. Create a new folder in the ImageGallery project and name it **Images** – right click on the ImageGallery project, and select **Add** → **New Folder** from the context menu.

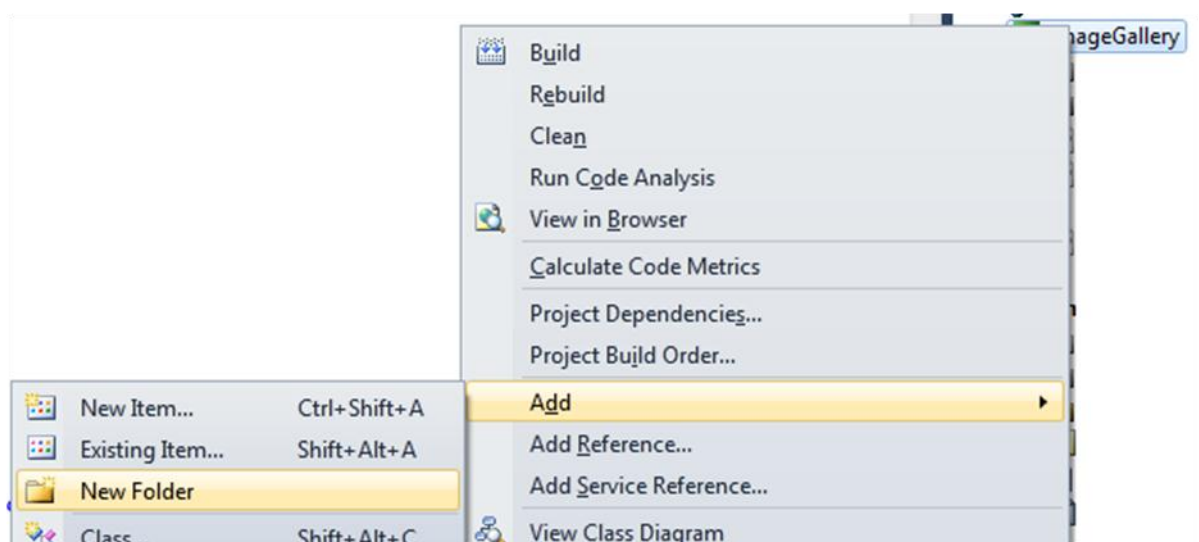
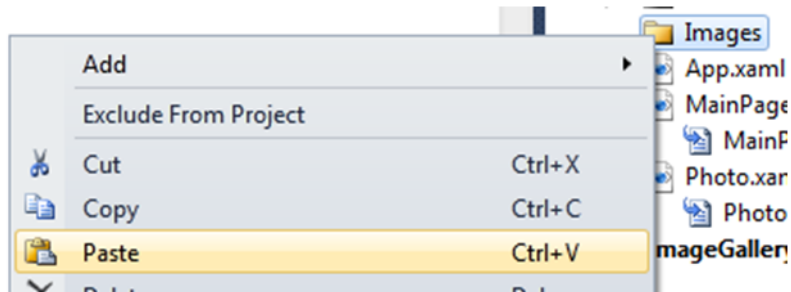


Figure 11

*Add New Folder to the Project*

15. Open Windows Explorer and navigate to the Lab installation folder. Navigate to “\Helpers\Images” folder and select all images there. Press Ctrl-C (or right click and choose **Copy** command)
16. Get back to the Visual Studio, right click on **Images** folder in ImageBrowser project and click **Paste**



**Figure 12**

*Paste Selection to the Project*

17. Compile and run the application. Select an image. Then place your cursor over one of the images and right-click. The context menu will appear. Next, click the **Delete** menu option and the image disappears.
-

#### Task 4 – Printing Support

1. Close the application if it's still running
2. Navigate to the “PrintMenuItem\_Click” handler function in MainPage.xaml.cs
3. In order to provide printing functionality, Silverlight 4 adds new a class called **PrintDocument**. It works by issuing events such as **StartPrint**, **EndPrint**, and **PrintPage**. We will use the **PrintPage** event in order to provide the visuals to print. Add the following code to the “PrintMenuItem\_Click” function:

```
C#

//First, close the context menu
LayoutRoot.Children.Remove(contextMenu);

PrintDocument pd = new PrintDocument();
    //Subscribe to the PrintPage event
pd.PrintPage += (s, args) =>
{
    //Handle printing event
};

    //Begin print - will show standard Windows Print Dialog
pd.Print("PrintDocument");
```

4. Every time Silverlight is ready to send a new page to the printer it will fire PrintPage event. The event arguments enable the developer to specify the size of the **PrintableArea**, define which UIElement will be printed and identify whether there will be additional pages.
5. Add the following code after the “Handle printing event” comment:

```
C#

args.PageVisual = LayoutRoot;
args.HasMorePages = false;
```

6. Compile and run the application. Check that printing function works by right clicking the control and choosing “Print”.
-

## Exercise 2 – Multi-touch on Windows 7

---

The goal of this exercise is to familiarize you with Multi-touch support as provided by Silverlight 4 under Windows 7. Multi-touch denotes a set of interaction techniques which allow Silverlight users to control the graphical user interface with more than one finger. Whilst a normal touch screen application allows you to drag or select visual elements with your finger, a multi-touch application allows you to do something such as resize a visual element, by stretching or squeezing it with two fingers. This functionality leverages Windows 7 multi-touch APIs and requires supported hardware. It works by subscribing to the frame reported event of the static **Touch** class – the event argument provides information about the collection of detected touch points. Within this collection, the **API will** have marked the most pressed touch point as the primary touch point in the collection. Each touch point exposes various properties, such as timestamp (for applying comparative temporal logic between detected touch points), an action (which can be move, down, up), a source device and others.

### Task 1 – Enabling Multi-touch support

In order to provide multi-touch support, the application must be aware of touch events reported by the Silverlight engine. We will add those events at two levels: at the level of a single photo (to make the photo active, much like clicking with the mouse) and at the level of the application surface (to move/rotate the photo objects).

1. Close the browser window hosting the Silverlight application if it is still running, and go back to the Visual Studio development environment.
2. First we will add touch support to the Photo object. Open Photo.xaml.cs and locate the “IntializePhoto” function.
3. Add subscription to touch events by adding the following code at the end of the InitializePhoto function body:

```
C#  
Touch.FrameReported += new TouchFrameEventHandler(Touch_FrameReported);
```

4. Create the event handler function:

```
C#  
void Touch_FrameReported(object sender, TouchFrameEventArgs e)  
{  
}  
}
```

5. A Touch event is a bubbled Routed Event (thus it will travel up the visual tree from its originating element to the root of the control). This means that events will bubble up and be received at the level of Canvases and Rectangles in the control itself, and not at the level of the Photo control. In order to ease on operation with the photo control we need to know which Photo was touched rather than which element of its visual tree. In addition, this event will arrive to all photo objects

under the pressed point, an effect which is not desired, requiring a work around to suppress it. To detect the control instance, use a helper function to detect touch events emanating from the real Photo control from the provided child. Add the following code to the Photo class:

```
C#
private Photo GetContainer(DependencyObject theObj)
{
    DependencyObject obj = VisualTreeHelper.GetParent(theObj);

    if (obj is Photo)
        return obj as Photo;
    else
        if (null != obj)
            return GetContainer(obj);
        else
            return null;
}
```

6. Get back to the “Touch\_FrameReported” function. The touch-points reported by the function arguments are provided in multiple ways. We will use a helper function to extract the primary touch point from all the touch points (in a collection). *Note: it is possible to understand from the selected element in a touch point’s collection if it is a primary point or not, but in this lab we will not iterate over the collection in order to do this.*
7. Add the following code at the beginning of “Touch\_FrameReported” function body:

```
C#
TouchPointCollection points = e.GetTouchPoints(null);
TouchPoint primaryPoint = e.GetPrimaryTouchPoint(null);
```

8. In order to deactivate mouse events while touching the screen (mouse events are not needed in a touch scenario), add the following lines after the previous code snippet in the “Touch\_FrameReported” method:

```
C#
if (null != primaryPoint && primaryPoint.Action == TouchAction.Down)
```

9. `e.SuspendMousePromotionUntilTouchUp();`
10. Remember that a touch event is a routed event? Add the following code snippet at the end of the “Touch\_FrameReported” function to get the Photo control from the primary touch point location, and apply some logic only if the event reported from the same instance of the control:

```
C#
Photo photo = null;
if (null != primaryPoint)
    photo = GetContainer(primaryPoint.TouchDevice.DirectlyOver);
else
    photo = GetContainer(points[0].TouchDevice.DirectlyOver);
```

```
if (this == photo)
{
}
}
```

11. In the case of the Photo instance, we are interested only on the “touchdown” action in order to “select” the photo instance. Add the following code, which will report the selection of the current Photo to the parent surface (much like in case of a mouse click event in a mouse-only scenario). Add the code snippet to the “if” block of the previous snippet:

```
C#
TouchPoint thePoint = null != primaryPoint ? primaryPoint : points[0];

switch (thePoint.Action)
{
    case TouchAction.Down:
        _parent.SetActivePhoto(this, ActionType.Touching,
            new Point(translateTransform.X +
rotateTransform.CenterX,
translateTransform.Y +
rotateTransform.CenterY),
                thePoint.Position);
        break;

    case TouchAction.Up:
        break;

    case TouchAction.Move:
        break;
}
```

12. Open MainPage.xaml.cs. Locate the MainPage constructor and subscribe to the Touch.FrameReported event – add the following code snippet at the end of the constructor code:

```
C#
Touch.FrameReported += new TouchFrameEventHandler(Touch_FrameReported);
```

13. Create the event handler function and collect touch points and the primary point (using the same code as in Photo class) – add the following code to the MainPage.xaml.cs:

```
C#
void Touch_FrameReported(object sender, TouchFrameEventArgs e)
{
    TouchPointCollection points = e.GetTouchPoints(null);
    TouchPoint primaryPoint = e.GetPrimaryTouchPoint(null);

    if (null != primaryPoint && primaryPoint.Action == TouchAction.Down)
        e.SuspendMousePromotionUntilTouchUp();
}
```

```
}

```

14. Now add the following code to move and rotate the selected photo. Add following snippet into the handler function body after the “if” in the previous snippet:

**C#**

```
if (null != _activePhoto && null != primaryPoint)
{
    // Perform the appropriate transform on the active photo
    var position = primaryPoint.Position;
    switch (_actionType)
    {
        case ActionType.Touching:
            if (points.Count == 1 && primaryPoint.Action == TouchAction.Move)
            {
                // Move it by the amount of the mouse move
                _activePhoto.Translate(position.X - _lastPosition.X,
                    position.Y - _lastPosition.Y);
            }
            else if (points.Count > 1 &&
                primaryPoint.Action == TouchAction.Move)
            {
                // Rotate it according to the angle the mouse moved
                // around the photo's center
                var radiansToDegrees = 360 / (2 * Math.PI);
                var lastAngle = Math.Atan2(_lastPosition.Y -
                    _photoCenter.Y, _lastPosition.X - _photoCenter.X) *
                    radiansToDegrees;
                var currentAngle = Math.Atan2(position.Y - _photoCenter.Y,
                    position.X - _photoCenter.X) * radiansToDegrees;
                _activePhoto.Rotate(currentAngle - lastAngle);

                // Scale it according to the distance the mouse
                // moved relative to the photo's center
                var lastLength = Math.Sqrt(Math.Pow(_lastPosition.Y -
                    _photoCenter.Y, 2) + Math.Pow(_lastPosition.X -
                    _photoCenter.X, 2));
                var currentLength = Math.Sqrt(Math.Pow(position.Y -
                    _photoCenter.Y, 2) + Math.Pow(position.X -
                    _photoCenter.X, 2));
                _activePhoto.Scale(currentLength / lastLength);
            }
            break;
    }
    _lastPosition = position;
}
```

15. Compile and run the application. Check the touch features of the application if you're on a touch-enabled device.



**Figure 9**  
*Finished Application*

---

## Conclusion

---

In this lab you learned how to build and enrich a Silverlight RIA application that utilizes multimedia support, how to enrich the standard UX with multi-touch support (for Windows 7 client machines), how to add image file Drag and Drop capabilities and printing support and how to enable native right mouse click events.