Hands-On Lab

Building Applications in Silverlight 4
Module 4: User Profile with Webcam, Drag and Drop, and Clipboard
Contents

Introduction ........................................................................................................................................... 3

Exercise 1: Acquiring Pictures with a Webcam...................................................................................... 4
  Get the List of Devices .......................................................................................................................... 4
  Acquire a Capture Source ..................................................................................................................... 5
  Take a Snapshot ..................................................................................................................................... 8

Exercise 2: Drag and Drop..................................................................................................................... 13
  Add Drop Support ............................................................................................................................... 13
  Load the Image from the Database (Optional) ...................................................................................... 15
Introduction

In this lab, you will enable end users to provide pictures of themselves, either by using webcams attached to their machines, or by dragging pictures from Windows Explorer into the Silverlight application.

You can start from where you left off with the previous lab, if you completed it. But if you did not get to the end, or if you would just prefer to work from a fresh starting point, make a copy of the solution in the StartingPoint folder for this lab, which contains the completed solution from the previous lab.
Exercise 1: Acquiring Pictures with a Webcam

In this part of the lab, you will create a new page for adding a picture to a logged-in attendee’s account. This page will show a live preview image from a camera, along with a button to capture a still image.

Get the List of Devices

2. In the SlEventManager project’s Views folder, add a new Silverlight Page called UserPicture.
3. In the Home.xaml file, find the StackPanel that contains the attendee-specific buttons (the Register and Unregister buttons)
4. Add the following before those buttons inside the panel:

   ```xml
   <HyperlinkButton Content="User Picture" NavigateUri="/UserPicture" />
   ```
5. Run the application.

You will need to be logged in each time you run the application in this part of the lab, because the user picture feature should only be available to logged in attendees. So you might find it helpful to check the checkbox to remain logged in. This will set a persistent cookie that should mean you won’t need to log in every single time you run. The cookie has a fairly short lifetime, but it’s better than logging in each time you rerun the application.

When you log in, you should see a User Picture link next to the Register and Unregister buttons, and clicking this will take you to the new (currently blank) UserPicture view.

Silverlight’s webcam API is designed to cope with machines that have multiple webcams attached. From a developer’s perspective, the most straightforward option is to present this as a list of cameras in the UI.

A more sophisticated approach would be to design three different UIs, one for when there are no webcams available, one for when there is exactly one, and one for when there are multiple cameras. The first two cases are far more common than the third, so you’d want them to work well, and you especially wouldn’t want to compromise the ease of use in these two common cases. If supporting the unusual multiple camera case means complicating the UI, it’s probably best only to show the complex version of the UI when you really need to. But since the focus here is working with the webcam API itself we’ll take the simple approach, rather than making you build three versions of the UI.

6. In the UserPicture.xaml file, drag a ListBox onto the page.
7. Position it at the top left, and make it large enough to hold about three lines before needing to scroll.

8. Name this **ListBox cameraList**.

9. In the **UserPicture.xaml.cs (C#)** or **UserPicture.xaml.vb (VB)** code behind, add the following to the constructor after the call to **InitializeComponent**:

   **C#**
   ```csharp
   cameraList.ItemsSource = 
   CaptureDeviceConfiguration.GetAvailableVideoCaptureDevices();
   ```

   **Visual Basic**
   ```vbnet
   cameraList.ItemsSource = 
   CaptureDeviceConfiguration.GetAvailableVideoCaptureDevices()
   ```

10. Run the application.

---

### Acquire a Capture Source

1. If you are not logged in, log in now.

2. Go to the **UserPicture** page. (Any cameras that show up will not be presented in a particularly useful fashion)

3. Add the following inside the **ListBox** element in **UserPicture.xaml**.

   **XAML**
   ```xaml
   <ListBox.ItemTemplate>
   <DataTemplate>
   <TextBlock Text="{Binding Path=FriendlyName}" />
   </DataTemplate>
   </ListBox.ItemTemplate>
   ```

4. Run the application again, and you should see more user-friendly names for any cameras on your system.

   When the user selects a camera, we want to show whatever the camera is looking at, so the user can get into a suitable pose ready for taking the picture. So you will need a UI element to show this preview. The webcam features of Silverlight 4 support this kind of live...
preview with an image source, which lets you use an Image element, and it also supports rendering webcam content through a VideoBrush, meaning you can paint any element that offers Brush properties. We’ll use a Border, painting its Background property (which is of type Brush) with the webcam picture.

5. Add a Border to your XAML and name it webcamLivePreview:

XAML

```xml
<Border
    x:Name="webcamLivePreview"
    Height="150" Width="150"
    HorizontalAlignment="Left" VerticalAlignment="Top"
    Margin="378,25,0,0"
    BorderBrush="Silver" BorderThickness="2" CornerRadius="10"
/>
```

Feel free to adjust the size and position if this clashes with the ListBox you already added.

6. Double click on the ListBox in the design surface to add a SelectionChanged event handler.

7. Add field of type CaptureSource to your code behind class. This allows you to acquire images from a webcam.

C#

```csharp
CaptureSource cs;
```

Visual Basic

```vbnet
Private cs As CaptureSource
```

8. In the SelectionChanged handler for the ListBox, add the following code

C#

```csharp
if (cs != null)
{
    cs.Stop();
}
```

Visual Basic

```vbnet
If cs IsNot Nothing Then
    cs.Stop()
End If
```

This ensures that if a CaptureSource was already running (perhaps the user had already selected one camera and is now selecting a different one) we stop it before attempting to attach it to the newly-selected camera:

9. Add the following code to check whether the user has selected a device, and to set it up if they have.

C#

```csharp
VideoCaptureDevice videoDevice = cameraList.SelectedItem as VideoCaptureDevice;
```
if (videoDevice != null) {
    // We need user permission to use the webcam.
    if (CaptureDeviceConfiguration.AllowedDeviceAccess ||
        CaptureDeviceConfiguration.RequestDeviceAccess()) {
        if (cs == null) {
            cs = new CaptureSource();
            VideoBrush vb = new VideoBrush();
            vb.SetSource(cs);
            vb.Stretch = Stretch.UniformToFill;
            webcamLivePreview.Background = vb;
        }
        cs.VideoCaptureDevice = videoDevice;
        cs.Start();
    }
}

**Visual Basic**

```vbnet
Dim videoDevice As VideoCaptureDevice = TryCast(cameraList.SelectedItem, VideoCaptureDevice)
If videoDevice IsNot Nothing Then
    ' We need user permission to use the webcam.
    If CaptureDeviceConfiguration.AllowedDeviceAccess OrElse
        CaptureDeviceConfiguration.RequestDeviceAccess() Then
        If cs Is Nothing Then
            cs = New CaptureSource()

            Dim vb As New VideoBrush()
            vb.SetSource(cs)
            vb.Stretch = Stretch.UniformToFill
            webcamLivePreview.Background = vb
        End If
        cs.VideoCaptureDevice = videoDevice
        cs.Start()
    End If
End If
```

This ensures we have the user's permission to use the webcam. It also creates a `CaptureSource` if one hasn’t already been created, along with a `VideoBrush` to paint the video into our preview `Border`. Finally, this code associates the `CaptureSource` with the chosen camera, and starts capturing video.

10. Run the application and go to the **UserPicture** view.

11. Select a video camera in the list, and you should see the permission dialog appear:
12. Click Yes. You should start seeing an image from the camera within a few seconds. There’s a slight issue with the code as it stands. If you navigate away to a different page, the camera doesn’t get released. If your camera has an activity light, you’ll see that it stays on if you navigate back to the Home view. This could cause a problem, as it could prevent your program (or other programs) from using the camera correctly later on, and in any case, it may disturb the user to see the activity light stay on. We need to ensure that the camera is released when users navigate away.

13. Add the code shown below to the OnNavigatedFrom method to stop the webcam if the user navigates away from the page:

```csharp
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    if (cs != null)
    {
        cs.Stop();
        cs = null;
    }
}
```

```visualbasic
    If cs IsNot Nothing Then
        cs.Stop()
        cs = Nothing
    End If
End Sub
```

If the user navigates completely out of your Silverlight application to another web page, or just closes the browser, the Silverlight plug-in will automatically release the camera for you. This code is only to ensure that you release the camera when the user navigates to some other part of your application.

**Take a Snapshot**

1. Add a button called `takePictureButton`
2. Set the button’s caption to **Take Picture**

3. Add another **Border** with the same dimensions and appearance as the existing one.

4. Name the new **Border** `userPictureBorder` and position it below the list of cameras.

5. Double click on the **Take Picture** button to add a **Click** event handler.

6. Put the following code in the **Client** event handler:

```csharp
if (cs == null) return;

cs.CaptureImageCompleted += (s, pe) =>
{
    var bmp = pe.Result;
    ImageBrush brush = new ImageBrush();
    brush.ImageSource = bmp;
    brush.Stretch = Stretch.UniformToFill;
    userPictureBorder.Background = brush;
};

cs.CaptureImageAsync();
```

```visualbasic
If cs Is Nothing Then
    Return
End If

AddHandler cs.CaptureImageCompleted, Sub(s, pe)
    Dim bmp = pe.Result
    Dim brush As New ImageBrush()
    brush.ImageSource = bmp
    brush.Stretch = Stretch.UniformToFill
    userPictureBorder.Background = brush
End Sub

cs.CaptureImageAsync()
```

7. Run the application.

8. Go to the **User Picture** view.

9. Select a camera

10. Click the **Take Picture** button. A still image from the camera should appear in the new **Border** element.

The next question is: how do we get this still image out of the UI and into the database? This is not entirely straightforward because Silverlight’s image handling only knows how to load bitmaps from streams, not how to create new bitmap streams from existing images. Fortunately, the .NET Image Tools project on codeplex (available from [http://imagetools.codeplex.com/](http://imagetools.codeplex.com/)) offers a solution, licensed under the Ms-PL (Microsoft...
11. In your **SlEventManager** project, add references to the DLLs in the ImageTools folder.

12. Add the following using declarations to your UserPicture.xaml.cs(C#) or UserPicture.xaml.vb(VB) code behind:

```csharp
using System.IO;
using ImageTools;
using ImageTools.IO.Png;
```

```vbnet
Imports System.IO
Imports ImageTools
Imports ImageTools.IO.Png
```

13. Add the following field to the code behind. This will hold the encoded bytes of the image:

```csharp
byte[] imageBytes;
```

```vbnet
Private imageBytes() As Byte
```

14. Go to the Take Picture button’s Click event handler

15. Add the following code after the line that sets the userPictureBorder.Background property

```csharp
PngEncoder encoder = new PngEncoder();
using (MemoryStream ms = new MemoryStream())
{
    var itImage = bmp.ToImage();
    encoder.Encode(itImage, ms);
    imageBytes = ms.ToArray();
}
```

```vbnet
Dim encoder As New PngEncoder()
Using ms As New MemoryStream()
    Dim itImage = bmp.ToImage()
    encoder.Encode(itImage, ms)
    imageBytes = ms.ToArray()
End Using
```

16. In the **SlEventManager.Web** project, open the EventManagerDomainService.cs(C#) or EventManagerDomainService.vb(VB) file in the Services folder.

17. And add the following method:
C#  
[Invoke]  
public void SetCurrentUserImage(byte[] imageBytes)  
{  
    Attendee att = GetOrCreateAttendeeForCurrentUser();  
    att.UserPicture = imageBytes;  
    ObjectContext.SaveChanges();  
}  

Visual Basic  
<Invoke()>  
Public Sub SetCurrentUserImage(ByVal imageBytes() As Byte)  
    Dim att As Attendee = GetOrCreateAttendeeForCurrentUser()  
    att.UserPicture = imageBytes  
    ObjectContext.SaveChanges()  
End Sub  

18. Add one more button to the user interface called savePictureButton.

19. Set the caption to Save Picture.

20. Add a Click event handler.

21. Add the following using declaration to the code behind:

C#  
using SlEventManager.Web.Services;

Visual Basic  
Imports SlEventManager.Web.Services

22. Implement the Click handler:

C#  
if (imageBytes != null)  
{  
    savePictureButton.IsEnabled = false;  
    var context = newEventManagerDomainContext();  
    context.SetCurrentUserImage(imageBytes, operation =>  
    {  
        savePictureButton.IsEnabled = true;  
    }, null);  
}

Visual Basic  
If imageBytes IsNot Nothing Then  
    savePictureButton.IsEnabled = False  
    Dim context = NewEventManagerDomainContext()  
    context.SetCurrentUserImage(imageBytes, Sub(operation)  
    savePictureButton.IsEnabled = True, Nothing)  
End If

23. Run the application.
25. Save the image.
26. In Visual Studio, go to the **Server Explorer** panel, expand the **Data Connections**
27. Expand the **Tables** item.
28. Right-click on the **Attendee** table and select **Show Table Data**. This should show that the **UserPicture** column for the attendee is no longer null.
Exercise 2: Drag and Drop

Not all users will have webcams, and even those who do may already have a picture they’d like to use instead of taking a new one. In this part of the lab, you’ll make it possible for users to drag a PNG or JPEG file from their computer into the application.

Don’t forget that the **Completed** folder in each lab contains a completed solution for every part of the lab. If you did not do the previous part (e.g., because you don’t have a webcam) you can use the solution in the Part 1 folder as your starting point for this part.

Add Drop Support

1. Open the SlEventManager solution. In the SlEventManager project, open the UserPicture.xaml file in the Views folder.

2. Find the Grid element named LayoutRoot.

3. Set its AllowDrop property to true. This enables it to act as a drop target.

4. Add a Drop event handler to the LayoutRoot element.

The Grid is currently invisible. It performs layout but has no appearance of its own. This presents a problem for drag and drop because invisible elements cannot directly receive mouse input. They can receive mouse events as they bubble out of child elements, but as far as Silverlight is concerned, any empty space in the UI really is empty, even though it’s contained within the bounding box of the grid. (Mouse events will end up being delivered to the element behind the Grid.)

To ensure that the whole area occupied by the Grid works as a drop target, you need to set its Background property. In fact you can set it to Transparent (although this is also invisible). Silverlight treats anything that has been painted with a brush as being visible to the mouse, even if it’s not visible to the eyes.

5. Set the background of the LayoutRoot to White.

6. Add the following using directive to the UserPicture.xaml.cs(C#) or UserPicture.xaml.vb(VB) code behind.

C#  

Visual Basic
Imports System.Windows.Media.Imaging

7. Add the following code to the **Drop** event handler:

C#  
if (e.Data.GetDataPresent(DataFormats.FileDrop))
{

 FileInfo[] fi = (FileInfo[]) e.Data.GetData(DataFormats.FileDrop);

BitmapImage image = new BitmapImage();
using (Stream fileStream = fi[0].OpenRead())
{
    image.SetSource(fileStream);
}

ImageBrush brush = new ImageBrush();
brush.ImageSource = image;
brush.Stretch = Stretch.UniformToFill;
userPictureBorder.Background = brush;

---

Visual Basic

If e.Data.GetDataPresent(DataFormats.FileDrop) Then
    Dim fi() As FileInfo = CType(e.Data.GetData(DataFormats.FileDrop), FileInfo())

    Dim image As New BitmapImage()
    Using fileStream As Stream = fi(0).OpenRead()
        image.SetSource(fileStream)
    End Using

    Dim brush As New ImageBrush()
    brush.ImageSource = image
    brush.Stretch = Stretch.UniformToFill
    userPictureBorder.Background = brush
End If

8. Run the application.

9. Navigate to the **UserPicture** View.

10. Drag a bitmap onto the UI. (There’s a JPEG file in the lab folder if you don’t have a suitable image to hand.) You should see the image appear.

    The **Save Picture** button won’t be working yet, because it relies on saving a copy of the bytes of the image. We’re not yet retrieving those bytes.

11. Inside the **using** statement you just added, after the call to **image.SetSource**, add this code:

    ```csharp
    fileStream.Seek(0, SeekOrigin.Begin);
    MemoryStream targetStream = new MemoryStream();
    fileStream.CopyTo(targetStream);
    imageBytes = targetStream.ToArray();
    ```

    ```visualbasic
    fileStream.Seek(0, SeekOrigin.Begin)
    Dim targetStream As New MemoryStream()
    fileStream.CopyTo(targetStream)
    ```
imageBytes = targetStream.ToArray()

12. Run the application.

13. Verify that you can drag an image and save it to the database.

**Load the Image from the Database (Optional)**

As an additional exercise, you might want to try writing code that loads the images saved in this lab to display them. One way to do this would be to add a method to the domain service that retrieves an `Attendee` entity for the currently logged in user. This has a `UserPicture` property containing the image bytes. Once you’ve got that on the client side, you can wrap it in a `MemoryStream`, and then pass that to a `BitmapImage` object’s `SetSource` method, just like you did with the file stream in the `Drop` handler. (Or if you want a more advanced approach, you could write an ASP.NET HTTP handler that makes the user pictures available via HTTP.)

The solution for this optional task can be found in the completed lab for Exercise 2.