In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.9 "Silverlight Recipes"

A synopsis of the book’s content

Information on where to buy this book

About the Author

A.P. Rajshkehar, Senior Developer with Red Hat, has more than 7 years of experience in IT, having worked on applications ranging from enterprise-level web applications and game development to android applications. His endeavors include development of Learning Management System, Health Systems, Supply Management Solution, and Xbox-based games. He has extensive knowledge of different technologies (SOA, Portal, Java Persistence, and .NET Persistence) and platforms (Sharepoint and JBoss EAP). He is also the author of Building Dynamic Web 2.0 Websites with Ruby on Rails, Packt Publishing, that was in Amazon's top 50 in Web 2.0 for 6 months. Apart from that he has also contributed to DevShed Portal on topics ranging from server-side development (JEE/.NET/RoR) to mobile (Symbian/Android-based development) and game development (SDL and OpenGL) with a total readership of more than 3 million. He is currently ranked among the top 10 authors on DevShed. You can find out about his interests on his blogs – http://aprajshekhar.wordpress.com and http://sententiasupervicis.wordpress.com.

For More Information:

Authoring a book is not an easy feat. However, the help and guidance from my family and friends helped me to author this book. First, I would like to thank Packt Publishing for providing me an opportunity to work on such an exciting project. I would like to thank my parents for their constant encouragement. Special thanks to my friends Shrikant Khare and Sormita Chakraborty for their support, encouragement, and initial research on the topics to be covered.

For More Information:
.NET is an architecture-neutral programming language and agnostic framework that caters to the varying requirements from desktop application, to business solutions, to multiplayer online three-dimensional games. The Version 4.5 added many new features and enhanced the existing ones that help in the development of robust and user-friendly solutions more easily. .NET Framework 4.5 Expert Programming Cookbook takes a hands-on approach in teaching you how to use the new as well as advanced features of the .NET Framework 4.5. Each topic will teach you how to use a specific feature of .NET to solve a real world problem or scenario.

This is a concise and practical cookbook with recipes which demonstrates advanced concepts with all the new functionality of the .NET Framework 4.5.

What This Book Covers

Chapter 1, Core .NET Recipes, will cover the core concepts in .NET, which include metadata programming, reflection, asynchronous I/O, and dynamic programming.

Chapter 2, Application Events and Windows Forms, covers topics such as event handling, dynamically generating controls, and layouts as well as creating video players using Managed DirectX.

Chapter 3, Threading and Parallel Programming, will cover multi-threading, thread-safety, and the parallel framework extensions to avoid threading pitfalls in your Windows Forms, WPF, and Silverlight applications.

Chapter 4, ASP.NET Recipes – I, explains the new features of ASP.NET applications including strongly-typed controls, HTML 5 controls, and client-side storage as well as passing data between Silverlight and the ASPX page.

Chapter 5, ADO.NET Recipes, covers saving and retrieving files of big size (BLOB) in SQL Server, managing transactions, and using DataSet to operate upon XML data.

For More Information:
Chapter 6, WCF Recipes, explains uploading files using streamed mode, implementing REST services, handling exceptions using FaultContract, implementing custom binding, and securing services using role-based security.

Chapter 7, WPF Recipes, will cover design patterns that include MVVM, repository pattern, and Data Mapper as well as new controls such as Ribbon control and live data shaper. It will also cover creating a video player using WPF Media API.

Chapter 8, ASP.NET Recipes – II, covers the new features of ASP.NET websites such as enabling Google/Facebook, SSO-based authentication, adding unobtrusive validation, embedding maps in websites, and protecting against cross-server scripting attacks.

Chapter 9, Silverlight Recipes, explains the new pivot control, accessing webcams, and client-side storage.

Chapter 10, Entity Framework Recipes, will cover using LINQ to join multiple entities, calling stored procedures using Entity Framework, handling long-running tasks, and using table-valued functions of MS SQL Server.

For More Information:
In this chapter we will cover:

- Using Pivot control to present asset data
- Accessing webcams
- Using client-side storage for saving a draft of the user registration data

Introduction

Silverlight helps developers create rich web clients. This is similar to WPF in the sense that it helps in developing content and feature-rich desktop applications. However, Silverlight applications run in a sandboxed environment within the web browser. In other words, as a developer you will not have access to certain functionalities that are present in WPF. These limitations include restricted access to DirectX for better graphics, limited access to webcams, and so on. With Version 5, new features have been included that help overcome these limitations. In this chapter we will look at some of these new functionalities. The Using Pivot control to present asset data recipe will be about Pivot control introduced in Version 5. Then we will see how to use webcams to capture live images. The Using client-side storage for saving a draft of the user registration data recipe will introduce you to isolated storage APIs, using which you can save data onto the client's system.

Using Pivot control to present asset data

The Pivot view makes interacting with data easy for the user. Until Version 5 of Silverlight (.NET 4.0 and below), PivotViewer control needed to be installed separately. However, with Silverlight 5 and .NET 4.5, it is supplied as part of the core distribution. In this recipe, we will see how to use PivotViewer control to display asset-related data.
To use Pivot control to present asset data, perform the following steps:

1. Launch Visual Studio 2012. Create a project of the type **Silverlight Application**. Name it **Pivot**.
2. Add a new folder. Name it **Entities**.
3. Add a new class to the **Entities** folder. Name it **Asset**.
4. Add the following properties to the class:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>int</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
</tr>
<tr>
<td>Region</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>double</td>
</tr>
</tbody>
</table>

5. Open **MainWindow.xaml**. Add the following statements in the `<Grid>` section:

   ```xml
   <sdk:PivotViewer x:Name="pvAsset">
     <!--Setting PivotProperties-->
     <sdk:PivotViewer.PivotProperties>
       <sdk:PivotViewerStringProperty Id="PName" Options="CanFilter"
           DisplayName="Name" Binding="{Binding Name}" />
       <sdk:PivotViewerStringProperty Id="PRegion" Options="CanFilter"
           DisplayName="Region" Binding="{Binding Region}" />
     </sdk:PivotViewer.PivotProperties>

     <!--Setting data-->
     <sdk:PivotViewer.ItemTemplates>
       <sdk:PivotViewerItemTemplate>
         <Border Width="200" Height="200" Background="Blue">
           <StackPanel Orientation="Vertical">
             <TextBlock Text="{Binding Name}" Foreground="White" FontSize="16" />
           </StackPanel>
         </Border>
       </sdk:PivotViewerItemTemplate>
     </sdk:PivotViewer.ItemTemplates>
   </sdk:PivotViewer>
   ```

For More Information:
6. Open MainWindow.xaml.cs. Add a private method that returns ObservableCollection<Asset>. Name it GenerateTestData. Its signature will be as follows:

```csharp
private ObservableCollection<Asset> GenerateTestData()
{
    ObservableCollection<Asset> temp = new ObservableCollection<Asset>();
    temp.Add(new Asset() { ID = 1, Name = "ASD", Region = "Mumbai", Value = 1000 });
    temp.Add(new Asset() { ID = 2, Name = "AS Hotel", Region = "Chennai", Value = 11000 });
    temp.Add(new Asset() { ID = 3, Name = "AD Cafe", Region = "Kolkata", Value = 10000 });
    temp.Add(new Asset() { ID = 4, Name = "Landmark", Region = "Mumbai", Value = 50000 });
    temp.Add(new Asset() { ID = 5, Name = "ASD II", Region = "Kolkata", Value = 400 });
    temp.Add(new Asset() { ID = 5, Name = "ASD III", Region = "Kolkata", Value = 1400 });
    return temp;
}
```

7. Add the following code to GenerateTestData:

```csharp
ObservableCollection<Asset> temp = new ObservableCollection<Asset>();
    temp.Add(new Asset() { ID = 1, Name = "ASD", Region = "Mumbai", Value = 1000 });
    temp.Add(new Asset() { ID = 2, Name = "AS Hotel", Region = "Chennai", Value = 11000 });
    temp.Add(new Asset() { ID = 3, Name = "AD Cafe", Region = "Kolkata", Value = 10000 });
    temp.Add(new Asset() { ID = 4, Name = "Landmark", Region = "Mumbai", Value = 50000 });
    temp.Add(new Asset() { ID = 5, Name = "ASD II", Region = "Kolkata", Value = 400 });
    temp.Add(new Asset() { ID = 5, Name = "ASD III", Region = "Kolkata", Value = 1400 });
    return temp;
```

8. Add the following statement to the constructor, after the call to InitializeComponent:

```csharp
pvAsset.ItemsSource = GenerateTestData();
```

For More Information:

9. Run the application. You will see the output as shown in the following screenshot. Test the application by selecting the regions.

How it works...

To display a pivot using PivotViewer control, two steps are required. First, set the properties that the user can use to filter the data and control the pivot. We did this using the PivotProperties property of PivotViewer. We wanted to set the filter on the Name and Region properties of the Asset class.

```xml
<sdk:PivotViewer.PivotProperties>
  <sdk:PivotViewerStringProperty Id="PName" Options="CanFilter"
      DisplayName="Name" Binding="{Binding Name}" />
  <sdk:PivotViewerStringProperty Id="PRegion" Options="CanFilter"
      DisplayName="Region" Binding="{Binding Region}" />
</sdk:PivotViewer.PivotProperties>
```

For More Information:
In the preceding code, we used `PivotViewerStringProperty` during runtime to signify that the property is of type `string`, it can be used to filter the data, and that it has to be bound to the `Name` and `Region` properties.

The second step is to tell PivotViewer control how to display the data when the filters are applied. That can be done using `PivotViewItemTemplate`. We used `StackPanel` and `TextBlock` within `PivotViewItemTemplate` for representing the data as shown in the following code:

```xml
<sdk:PivotViewerItemTemplate>
    <Border Width="200" Height="200" Background="Blue">
        <StackPanel Orientation="Vertical">
            <TextBlock Text="{Binding Name}" FontSize="16" Foreground="White" />
            <TextBlock Text="{Binding Value}" FontSize="16" Foreground="White" />
        </StackPanel>
    </Border>
</sdk:PivotViewerItemTemplate>
```

`StackPanel` and `TextBlock` are basic ways to display data. Any kind of transformations and controls can be used to display the data.

### Accessing webcams

In certain scenarios, you may have to access the webcam of the client. From Silverlight 4 onwards, the API to access webcams have become simpler to use. In this recipe, we will see how to use webcams to display live images.

#### How to do it...

2. Open `MainPage.xaml`. Add the following code to the `<Grid>` section:

```xml
<Rectangle RadiusX="5" RadiusY="5" x:Name="rectCam" Height="285" HorizontalAlignment="Left" Margin="10,10,0,0" Stroke="Black" StrokeThickness="1" VerticalAlignment="Top" Width="383" Fill="Black" />

<Button Content="Start Webcam" Height="23" HorizontalAlignment="Left" Margin="68,330,0,0" Name="btnStart" VerticalAlignment="Top" Width="94" />
```

For More Information:

3. The page should look similar to the following screenshot:

4. Open MainPage.xaml.cs. Add a private variable of the type CaptureSource. Name it _captureSource.
   
   ```csharp
   private CaptureSource _captureSource = null;
   ```

5. Add a method of the type void. Name it StartWebCam. Its signature will be as follows:

   ```csharp
   private void StartWebCam()
   {
   }
   ```
6. Add the following code to StartWebCam:
   
   ```csharp
   if (_captureSource == null)
   {
       _captureSource = new CaptureSource();
   }
   if (_captureSource.State == CaptureState.Stopped)
   {
       _captureSource.VideoCaptureDevice = 
       CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();
       VideoBrush previewBrush = new VideoBrush();
       previewBrush.SetSource(_captureSource);
       rectCam.Fill = previewBrush;
       _captureSource.Start();
   }
   ```

7. Open MainPage.xaml in the design mode. Double-click on the Start Webcam button to add a Click event handler.

8. Add the following code to the event handler:
   
   ```csharp
   if (!CaptureDeviceConfiguration.AllowedDeviceAccess)
   {
       if (!CaptureDeviceConfiguration.RequestDeviceAccess())
       {
           MessageBox.Show("Cannot access the webcam");
       }
   }
   else
   {
       StartWebCam();
   }
   ```

9. Open MainPage.xaml in the design mode. Double-click on the Stop Webcam button to add a Click event handler.

10. Add the following code to the event handler:
    
    ```csharp
    if (_captureSource != null && _captureSource.State == 
    CaptureState.Started)
    {
        _captureSource.Stop();
    }
    ```

11. Run the application. Test the application by clicking on the Start Webcam and Stop Webcam buttons.

For More Information:  
How it works...

The first step in accessing a webcam is finding out whether we have permission to access it or not. To do so, we used the AllowDeviceAccess property and the RequestDeviceAccess() method of the CaptureDeviceConfiguration class in the Click event handler of btnStart.

```csharp
if (!CaptureDeviceConfiguration.AllowedDeviceAccess)
{
    if (!CaptureDeviceConfiguration.RequestDeviceAccess())
    {
        MessageBox.Show("Cannot access the webcam");
    }
}
```

If we get a false value for any of them, we show a message to the user and stop the process. If we get a true value for both, we need to initialize the CaptureSource variable. That's what we did in the StartWebCam method.

```csharp
_captureSource = new CaptureSource();
```

Then set the VideoCapture property of _capture using the GetDefaultVideoCaptureDevice() method of CaptureDeviceConfiguration.

```csharp
_captureSource.VideoCaptureDevice =
    CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();
```

Next, we need a brush using which we can draw (the output of webcam) on the Rectangle control. Hence, we instantiated VideoBrush, set its source to _captureSource, and then set the Fill property of the Rectangle control to the instance of VideoBrush.

```csharp
VideoBrush previewBrush = new VideoBrush();
previewBrush.SetSource(_captureSource);
rectCam.Fill = previewBrush;
```

Once we have the Brush instance, we can start capturing the image using the webcam.

```csharp
_captureSource.Start();
```

To stop the capturing of images, we first checked whether _captureSource has been instantiated or not. If it is instantiated, we need to check whether the capturing is going on. If both are true, we call the Stop() method of _captureSource to stop the capturing.

```csharp
if (_captureSource != null && _captureSource.State == CaptureState.Started)
{
    _captureSource.Stop();
}
```
Using client-side storage for saving a draft of the user registration data

Isolated storage APIs provide a way to store limited amount of data at the client side. Using this functionality, we can save data such as search strings and usernames. In this recipe we will use the isolated storage API so that the user can save the data entered in the registration form on the client's system.

How to do it...

2. Open MainPage.xaml. Design it so that it looks similar to the following screenshot:

![Image of the registration form with save and clear buttons]

For More Information:
3. Name the controls as detailed in the following table:

<table>
<thead>
<tr>
<th>Control</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textbox</td>
<td>txtUserName</td>
<td>To enter the username</td>
</tr>
<tr>
<td>Textbox</td>
<td>txtFullName</td>
<td>To enter the full name of the user</td>
</tr>
<tr>
<td>Textbox</td>
<td>txtEmail</td>
<td>To enter the e-mail address of the user</td>
</tr>
<tr>
<td>Button</td>
<td>btnDraft</td>
<td>To save the entered values in the user's system</td>
</tr>
<tr>
<td>Button</td>
<td>btnClear</td>
<td>To clear the values</td>
</tr>
<tr>
<td>Button</td>
<td>btnLoad</td>
<td>To load and display the saved values</td>
</tr>
</tbody>
</table>

4. Open MainPage.xaml.cs. Add a private variable of the type IsolatedStorageSettings. Name it appSettings.

5. Instantiate it using the ApplicationSettings property of IsolatedStorageSettings as shown in the following code snippet:

```csharp
private IsolatedStorageSettings appSettings = IsolatedStorageSettings.ApplicationSettings;
```

6. Add a Click event handler for btnDraft. Add the following code in the event handler:

```csharp
appSettings.Add("username", txtUserName.Text);
appSettings.Add("fullname", txtFullName.Text);
appSettings.Add("email", txtEmail.Text);
```

7. Next, add a Click event handler for btnClear. Add the following code in the event handler:

```csharp
txtEmail.Text = "";
txtFullName.Text = "";
txtUserName.Text = "";
```

8. Then, add a Click event handler for btnLoad. Add the following code in the event handler:

```csharp
txtUserName.Text = appSettings["username"]!=null?appSettings["username"].ToString() : "";
txtFullName.Text = appSettings["fullname"]!=null?appSettings["fullname"].ToString() : "";
txtEmail.Text = appSettings["email"]!=null?appSettings["email"].ToString() : "";
```

9. Run the application. Enter values in the textboxes. Click on the Save Draft button.
10. Then click on Clear. The values will be cleared.
11. Now click on the Load button. The saved values will be displayed.
How it works...

The `IsolatedStorageSettings` class acts as a collection. So we can save our data using the `Add` method. The `Add` method takes `key` and `value` as parameters. To save our data, we called the `Add` method with a key and the text from the textbox as the value.

```csharp
appSettings.Add("username", txtUserName.Text);
appSettings.Add("fullname", txtFullName.Text);
appSettings.Add("email", txtEmail.Text);
```

Similarly, we can access the saved value by using the key as an index. That's what we did in the `Click` event of the event handler of `btnLoad`.

```csharp
txtUserName.Text = appSettings["username"]!=null? appSettings["username"].ToString():"";
txtFullName.Text = appSettings["fullname"]!=null? appSettings["fullname"].ToString():"";
txtEmail.Text = appSettings["email"]!=null? appSettings["email"].ToString():"";
```

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.