* 1. 

Hands-On Lab

Building Applications in Silverlight 4

09 – Deep Dive into Out of Browser

**Contents**

[Lab 9 - Advanced Silverlight Out of Browser- Introduction 3](#_Toc276632459)

[Exercise 1: Enabling a drop zone 6](#_Toc276632460)

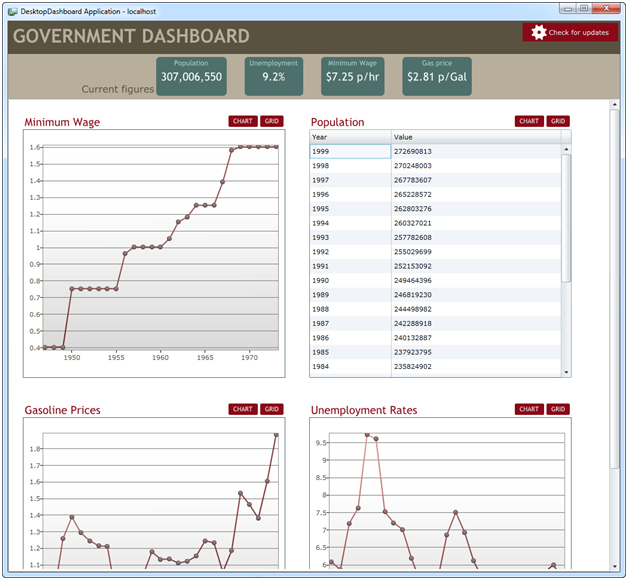
[Exercise 2: Importing Excel data 8](#_Toc276632461)

[Exercise 3: Working with the hard drive 15](#_Toc276632462)

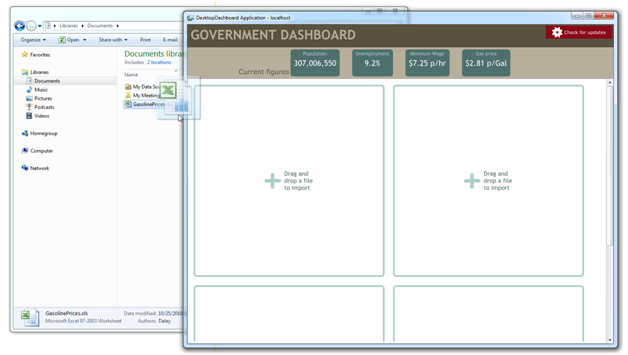
[Exercise 4: Checking for updates 23](#_Toc276632463)

[Summary 27](#_Toc276632464)

Lab 9 - Advanced Silverlight Out of Browser- Introduction

* 1. Silverlight out-of-browser enables you to create desktop applications. Using elevated trust further opens the possibilities of what your application can do. This lab is designed to further explore what can be down with an out-of-browser Silverlight application.
  2. In the lab you'll create an application that imports data directly from Excel. Along the way you'll learn how to interact with the hard drive, check for updates, and work with Excel from Silverlight.
  3. You'll start by dragging and dropping Excel files onto the application. Next, you’ll call Excel through the AutomationFactory and process the data. You’ll create files and directories on the hard drive to store temporary data. Finally, you’ll learn how to check for application updates. The Silverlight application that you'll create is shown next:
     1. 
     2. Figure 1
  4. **You will benefit from this lab if:**
  + You are looking for a deeper understanding on how Silverlight interacts with the hard drive.
  + Want to integrate Silverlight and Excel
  + Looking for guidance on how to deploy updates to applications.
  1. **You will learn:**
  + How to work with the local file system from an Out-of-Browser Trusted application
  + Further explore how to use the AutomationFactory, by importing data from Excel into Silverlight
  + Check for updates to your application
  1. **Business Requirements for the Silverlight application include:**
  + Drop files onto the application to be imported
  + Open a file from anywhere on the hard drive
  + Save a file to a temporary directory and file on the hard drive
  + Loop through an Excel to extract data
  + Enable the user to check for a new version of the application
  1. Estimated Time 45 Minutes

Exercise 1: Enabling a drop zone

* 1. This first section shows how to enable a drop area where an Excel file will be dropped. The next section shows how to open the dropped Excel file and read its data.
  2. Before getting started, copy and paste the four excel files found in the zip file in to your My Documents folder. Since Silverlight has limited file access you’ll need to put these files in a place where Silverlight can work with them. Later in this lab you’ll learn how to overcome this limitation by creating a temp directory and file from Silverlight.
  3. Open the starter solution, DesktopDashboard.sln, and then open DataWidget.xaml.cs or DataWidget.xaml.vb. This UserControl is where we’re going to handle both the drop off files and rendering of the Excel content. Last bit of set up, ensure the Silverlight project is set as the Startup project.
  4. To handle the dropping of files, the Drop event needs to be handled. Register the Drop event for the Panel called ImportPanel. Additionally set the AllowDrop property to True.
     1. C#
     2. public DataWidget()
     3. {
     4. // Required to initialize variables
     5. InitializeComponent();
     6. this.Data = new ObservableCollection<YearValueData>();
     7. this.ImportPanel.AllowDrop = true;
     8. this.ImportPanel.Drop += new DragEventHandler(ImportPanel\_Drop);
     9. }
     10. Visual Basic
     11. Public Sub New()
     12. ' Required to initialize variables
     13. InitializeComponent()
     14. Me.Data = New ObservableCollection(Of YearValueData)()
     15. AddHandler ImportPanel.Drop, AddressOf ImportPanel\_Drop
     16. End Sub
  5. The DropEventArgs contains file information for dropped files. Copy and paste the below event handler below the DataWidet constructor.
     1. C#
     2. void ImportPanel\_Drop(object sender, DragEventArgs e)
     3. {
     4. if (e.Data != null)
     5. {
     6. FileInfo[] files = e.Data.GetData(DataFormats.FileDrop) as FileInfo[];
     7. }
     8. }
     9. Visual Basic
     10. Private Sub ImportPanel\_Drop(ByVal sender As Object, ByVal e As DragEventArgs)
     11. If e.Data IsNot Nothing Then
     12. Dim files() As FileInfo = TryCast(e.Data.GetData(DataFormats.FileDrop), FileInfo())
     13. End If
     14. End Sub
  6. Run the project. Drag and drop the GasolinePrices.xls, from My Documents, onto the application (as seen in the below figure). Visually nothing will happen, however putting a breakpoint in the Drop event handler will show the files are coming back.
     1. 
     2. Figure
     3. Drag and drop

Exercise 2: Importing Excel data

* 1. As a developer it’s imperative to know how to work with tools that businesses rely on. Microsoft Excel is a tool frequently requested to have applications integrated with. With the out-of-browser feature in Silverlight, you’re able to call Excel methods, which include working with data contained inside.
  2. In this exercise you’ll use the AutomationFactory to open an instance of Excel, open the first Worksheet, and loop through the content. The starter solution is already configured to make use of the AutomationFactory, however if you’re working on your own solution, you must first ensure the application has elevated trusted enabled. You can toggle this setting from the Out-of-browser settings in the Project’s Properties panel.
  3. Add the below code to the Drop event handler in DataWidget.xaml.cs or DataWidget.xaml.vb The code:
     1. gets a handle on the first file, files[0],
     2. opens Excel using AutomationFactory,
     3. loads the Excel document from MyDocuments (notice the use of Environment.SpecialFolder.MyDocuments),
     4. And loads the active sheet.
     5. C#
     6. // Get the first dropped file
     7. var fi = files[0];
     8. // Create the Excel object
     9. dynamic excel = AutomationFactory.CreateObject("Excel.Application");
     10. // Open the excel document. Must be located in "My Documents"
     11. dynamic excelWorkBook = excel.Workbooks.Open(string.Format("{0}\\{1}",
     12. Environment.GetFolderPath(

Environment.SpecialFolder.MyDocuments),fi.Name));

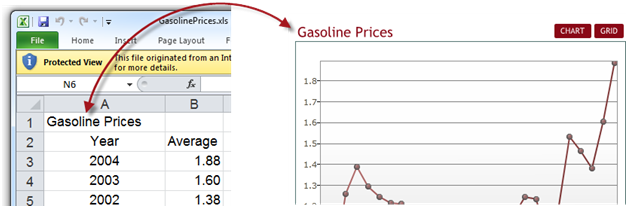
* + 1. // Read the Worksheet
    2. dynamic activeWorkSheet = excelWorkBook.ActiveSheet();
    3. Visual Basic
    4. Dim fi = files(0)
    5. Dim excel As Object
    6. Try
    7. ' Check to see if Excel is already running
    8. excel = AutomationFactory.GetObject("Excel.Application")
    9. Catch
    10. excel = AutomationFactory.CreateObject("Excel.Application")
    11. End Try
    12. ' Open the excel document

Dim excelWorkBook As Object = excel.Workbooks.Open(String.Format("{0}\\{1}",

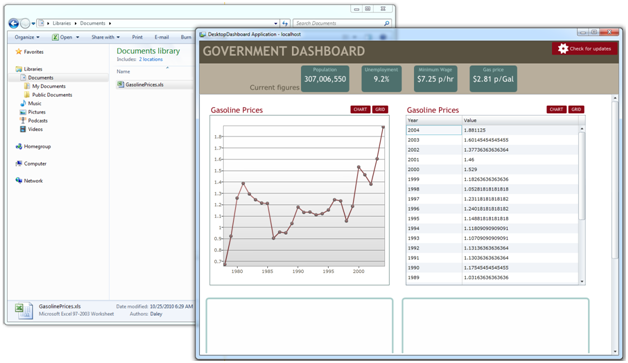
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), fi.Name))

* + 1. ' Read the Worksheet
    2. Dim activeWorkSheet As Object = excelWorkBook.ActiveSheet()
  1. At this point the active worksheet is open and you can access the data. Add the below code immediately following the above block to loop through content of the worksheet.
     1. C#
     2. // Cells to Read
     3. dynamic cell1, cell2;
     4. // Iterate through Cells
     5. for (int count = 3; count < 30; count++)
     6. {
     7. cell1 = activeWorkSheet.Cells[count, 1];
     8. cell2 = activeWorkSheet.Cells[count, 2];

Data.Add(new YearValueData()

* + 1. {
    2. Year = cell1.Value,
    3. Value = cell2.Value
    4. });
    5. }
    6. Visual Basic
    7. ' Cells to Read
    8. Dim cell1, cell2 As Object
    9. ' Iterate through Cells
    10. For count As Integer = 3 To 29
    11. cell1 = activeWorkSheet.Cells(count, 1)
    12. cell2 = activeWorkSheet.Cells(count, 2)
    13. Data.Add(New YearValueData() With {.Year = cell1.Value, .Value = cell2.Value})
    14. Next count
  1. In the Excel sheet, at position A1 (as seen in Figure 2), is the name of the Worksheet. The following code gets that value (Cells[1,1]) and assigns the value to the Text property of Title TextBlock.
     1. C#
     2. // Title is a TextBlock in XAML, this sets the value
     3. Title.Text = activeWorkSheet.Cells[1, 1].Value;
     4. Visual Basic
     5. 'Title is a TextBlock in XAML, this sets the value
     6. Title.Text = activeWorkSheet.Cells(1, 1).Value
     7. 
     8. Figure
  2. Add the below code to clean up the Excel object. This will close the Excel process and prevent any potential file locks
     1. C#
     2. // Close the workbook
     3. excelWorkBook.Close();
     4. // Close the Excel process
     5. excel.Quit();
     6. Visual Basic
     7. ' Close the workbook
     8. excelWorkBook.Close()
     9. ' Close the Excel process
     10. excel.Quit()
  3. Finally, now the data is loaded into the local variable Data, existing DataGrid and Chart’s ItemsSource can be set.
     1. C#
     2. // Populate the DataGrid
     3. ExcelDataGrid.ItemsSource = this.Data;
     4. // Create LineSeries
     5. LineSeries lineSeries = new LineSeries();
     6. lineSeries.ItemsSource = this.Data;
     7. lineSeries.IndependentValueBinding = new Binding("Year");
     8. lineSeries.DependentValueBinding = new Binding("Value");
     9. this.Chart.Series.Add(lineSeries);
     10. Visual Basic
     11. ' Populate the DataGrid
     12. ExcelDataGrid.ItemsSource = Me.Data
     13. ' Create LineSeries
     14. Dim lineSeries As New LineSeries()
     15. lineSeries.ItemsSource = Me.Data
     16. lineSeries.IndependentValueBinding = New Binding("Year")
     17. lineSeries.DependentValueBinding = New Binding("Value")
     18. Me.Chart.Series.Add(lineSeries)
  4. Putting this all together, here is what the ImportPanel’s Dropped event handler should look like.
     1. C#
     2. void ImportPanel\_Drop(object sender, DragEventArgs e)
     3. {
     4. if (e.Data != null)
     5. {
     6. FileInfo[] files = e.Data.GetData(DataFormats.FileDrop) as FileInfo[];
     7. if (AutomationFactory.IsAvailable)
     8. {
     9. // Get the first file
     10. var fi = files[0];
     11. // Create the Excel object
     12. dynamic excel = AutomationFactory.CreateObject("Excel.Application");
     13. // Open the excel document. Must be located in "My Documents"
     14. dynamic excelWorkBook = excel.Workbooks.Open(string.Format("{0}\\{1}",
     15. Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),fi.Name));
     16. // Read the Worksheet
     17. dynamic activeWorkSheet = excelWorkBook.ActiveSheet();
     18. // Cells to Read
     19. dynamic cell1, cell2;
     20. // Iterate through Cells
     21. for (int count = 3; count < 30; count++)
     22. {
     23. cell1 = activeWorkSheet.Cells[count, 1];
     24. cell2 = activeWorkSheet.Cells[count, 2];
     25. Data.Add(new YearValueData()
     26. {
     27. Year = cell1.Value,
     28. Value = cell2.Value
     29. });
     30. }
     31. // Title is a TextBlock in XAML, this sets the value
     32. Title.Text = activeWorkSheet.Cells[1, 1].Value;
     33. // Close the workbook
     34. excelWorkBook.Close();
     35. // Close the Excel process
     36. excel.Quit();
     37. // Populate the DataGrid
     38. ExcelDataGrid.ItemsSource = this.Data;
     39. // Create LineSeries
     40. LineSeries lineSeries = new LineSeries();
     41. lineSeries.ItemsSource = this.Data;
     42. lineSeries.IndependentValueBinding = new Binding("Year");
     43. lineSeries.DependentValueBinding = new Binding("Value");
     44. this.Chart.Series.Add(lineSeries);
     45. }
     46. }
     47. }
     48. Visual Basic
     49. Private Sub ImportPanel\_Drop(ByVal sender As Object, ByVal e As DragEventArgs)
     50. If e.Data IsNot Nothing Then
     51. Dim files() As FileInfo = TryCast(e.Data.GetData(DataFormats.FileDrop), FileInfo())
     52. Dim fi = files(0)
     53. Dim excel As Object
     54. Try
     55. ' Check to see if Excel is already running
     56. excel = AutomationFactory.GetObject("Excel.Application")
     57. Catch
     58. excel = AutomationFactory.CreateObject("Excel.Application")
     59. End Try
     60. ' Open the excel document

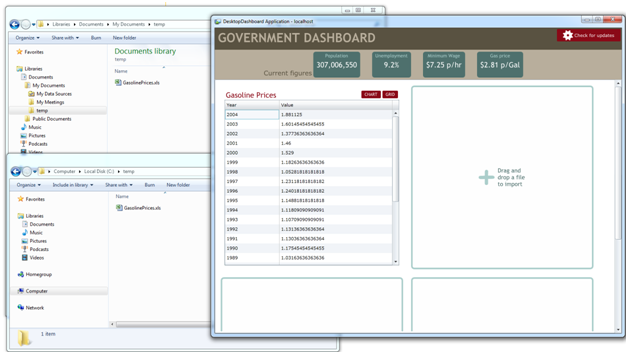
Dim excelWorkBook As Object = excel.Workbooks.Open(String.Format("{0}\\{1}", Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), fi.Name))

* + 1. ' Read the Worksheet
    2. Dim activeWorkSheet As Object = excelWorkBook.ActiveSheet()
    3. ' Cells to Read
    4. Dim cell1, cell2 As Object
    5. ' Iterate through Cells
    6. For count As Integer = 3 To 29
    7. cell1 = activeWorkSheet.Cells(count, 1)
    8. cell2 = activeWorkSheet.Cells(count, 2)
    9. Data.Add(New YearValueData() With {.Year = cell1.Value, .Value = cell2.Value})
    10. Next count
    11. 'Title is a TextBlock in XAML, this sets the value
    12. Title.Text = activeWorkSheet.Cells(1, 1).Value
    13. ' Close the workbook
    14. excelWorkBook.Close()
    15. ' Close the Excel process
    16. excel.Quit()
    17. ' Populate the DataGrid
    18. ExcelDataGrid.ItemsSource = Me.Data
    19. ' Create LineSeries
    20. Dim lineSeries As New LineSeries()
    21. lineSeries.ItemsSource = Me.Data
    22. lineSeries.IndependentValueBinding = New Binding("Year")
    23. lineSeries.DependentValueBinding = New Binding("Value")
    24. Me.Chart.Series.Add(lineSeries)
    25. End If
    26. End Sub
    27. 
    28. Figure
    29. **Note:** Working with object via AutomationFactory, can be challenging. Unfortunately there is no intellisence helping out, so you must rely on documentation. Here are the coding documents that will help when working with Excel.
    30. Excel 2007 Developer Reference <http://msdn.microsoft.com/en-us/library/bb149067(v=office.12).aspx>
    31. Excel 2010 Developer Reference <http://msdn.microsoft.com/en-us/library/ff846370.aspx>

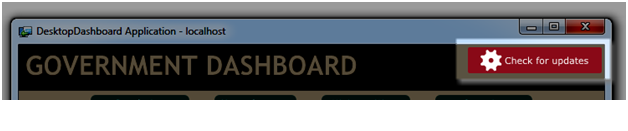
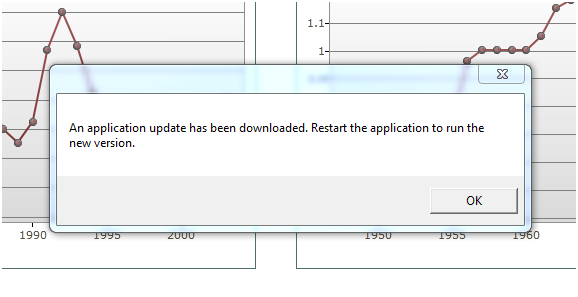
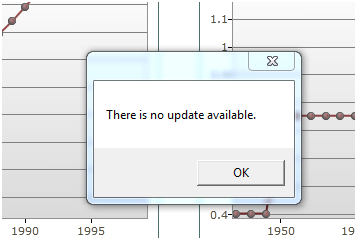
Exercise 3: Working with the hard drive

* 1. When running in a trusted environment, you can access only files in user folders, specifically the MyDocuments, MyMusic, MyPictures, and MyVideos folders. Although this makes sense from a security point of view, it’s limiting. You want to enable the user to drag their data from any location. As it stands right now, if you try to drop a file from a location other than stated above, Silverlight will throw a security error.
  2. In this section we’re going to cover how to import an Excel file from anywhere on the hard drive, how to create and delete directories, and how to create and modify files. Before you begin it’s important to lay out exactly what needs to be done.
  + Read the bytes of the dropped file
  + Create a temporary directory in My Documents
  + Create a temporary file with the byte contents of the dropped file
  + Open the temporary file in Excel
  + Read the data
  + Finally delete both the temporary file and directory
  1. Create a new method, called CopyFileToTempDirectory. This method will copy the dropped file to into a temporary directory in My Documents.
     1. C#
     2. private bool CopyFileToTempDirectory(FileInfo fi, string tempDirectory, string fileName)
     3. {
     4. using (Stream stream = fi.OpenRead())
     5. {
     6. try
     7. {
     8. byte[] buffer = new byte[Convert.ToInt32(stream.Length)];
     9. stream.Read(buffer, 0, Convert.ToInt32(stream.Length));
     10. stream.Close();
     11. // Create a temporary directory in My Documents
     12. Directory.CreateDirectory(tempDirectory);
     13. // Write a new file to the temp directory
     14. File.WriteAllBytes(string.Format("{0}\\{1}", tempDirectory, fileName), buffer);
     15. return true;
     16. }
     17. catch (Exception e)
     18. {
     19. return false;
     20. }
     21. }
     22. }
     23. Visual Basic
     24. Private Function CopyFileToTempDirectory(ByVal fi As FileInfo, ByVal tempDirectory As String, ByVal fileName As String) As Boolean
     25. Using stream\_Renamed As Stream = fi.OpenRead()
     26. Try
     27. ' Copy file to My Documents. This ensures we can open it up in Excel
     28. Dim buffer(Convert.ToInt32(stream\_Renamed.Length) - 1) As Byte
     29. stream\_Renamed.Read(buffer, 0, Convert.ToInt32(stream\_Renamed.Length))
     30. stream\_Renamed.Close()
     31. ' Create a temporary directory in My Documents
     32. Directory.CreateDirectory(tempDirectory)
     33. ' Write a new file to the temp directory
     34. File.WriteAllBytes(String.Format("{0}\{1}", tempDirectory, fileName), buffer)
     35. Return True
     36. Catch e As Exception
     37. Return False
     38. End Try
     39. End Using
     40. End Function
  2. Back in the Drop event handler, call the newly created method, CopyFileToTempDirectory , and tell the Worksheet to open the temporary folder instead of the calling My Documents.
     1. C#
     2. void ImportPanel\_Drop(object sender, DragEventArgs e)
     3. {
     4. // ...

string myDocuments = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

* + 1. string tempDirectory = string.Format("{0}\\temp", myDocuments);
    2. string tempFullPath = string.Format("{0}\\{1}", tempDirectory, fi.Name);
    3. // Copy the file to a temp directory in My Documents
    4. CopyFileToTempDirectory(fi, tempDirectory, fi.Name);
    5. // Create the Excel object
    6. dynamic excel = AutomationFactory.CreateObject("Excel.Application");
    8. dynamic excelWorkBook = excel.Workbooks.Open(tempFullPath);
    10. // ...
    11. }
    12. Visual Basic
    13. Private Sub ImportPanel\_Drop(ByVal sender As Object, ByVal e As DragEventArgs)
    14. '...
    15. Dim myDocuments As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    16. Dim tempDirectory As String = String.Format("{0}\temp", myDocuments)
    17. Dim tempFullPath As String = String.Format("{0}\{1}", tempDirectory, fi.Name)
    18. ' Copy the file to a temp directory in My Documents
    19. CopyFileToTempDirectory(fi, tempDirectory, fi.Name)
    20. 'Dim result = New ObservableCollection(Of YearValueData)()
    21. Dim excel As Object
    22. Try
    23. ' Check to see if Excel is already running
    24. excel = AutomationFactory.GetObject("Excel.Application")
    25. Catch
    26. excel = AutomationFactory.CreateObject("Excel.Application")
    27. End Try
    28. ' Open the excel document
    29. Dim excelWorkBook As Object = excel.Workbooks.Open(tempFullPath)
    30. '...
    31. End Sub
  1. This screen shot shows the GasolinePrices.xls being dragged from C:\temp and the temporary directory created in My Document.
     1. 
     2. Figure
  2. Remove the temporary directory and file. Create a new method to delete both the temporary directory and temporary file.
     1. C#
     2. private void CleanUpFileSystem(string tempDirectory, string tempFullPath)
     3. {
     4. File.Delete(tempFullPath);
     5. // Remove temp directory
     6. Directory.Delete(tempDirectory + "\\", true);
     7. }
     8. Visual Basic
     9. Private Sub CleanUpFileSystem(ByVal tempDirectory As String, ByVal tempFullPath As String)
     10. File.Delete(tempFullPath)
     11. ' Remove temp directory
     12. Directory.Delete(tempDirectory & "\", True)
     13. End Sub
  3. Finally add the call to this method in the Drop event handler. CleanUpFileSystem(tempDirectory, tempFullPath); The final Drop event handler looks like this:
     1. C#
     2. void ImportPanel\_Drop(object sender, DragEventArgs e)
     3. {
     4. if (e.Data != null)
     5. {
     6. FileInfo[] files = e.Data.GetData(DataFormats.FileDrop) as FileInfo[];
     7. if (AutomationFactory.IsAvailable)
     8. {
     9. // Get the first file
     10. var fi = files[0];
     11. string myDocuments = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
     12. string tempDirectory = string.Format("{0}\\temp", myDocuments);
     13. string tempFullPath = string.Format("{0}\\{1}", tempDirectory, fi.Name);
     14. // Copy the file to a temp directory in My Documents
     15. CopyFileToTempDirectory(fi, tempDirectory, fi.Name);
     16. // Create the Excel object
     17. dynamic excel = AutomationFactory.CreateObject("Excel.Application");
     18. // Open the excel document. Must be located in "My Documents"
     19. dynamic excelWorkBook = excel.Workbooks.Open(tempFullPath);
     20. // Read the Worksheet
     21. dynamic activeWorkSheet = excelWorkBook.ActiveSheet();
     22. // Cells to Read
     23. dynamic cell1, cell2;
     24. // Iterate through Cells
     25. for (int count = 3; count < 30; count++)
     26. {
     27. cell1 = activeWorkSheet.Cells[count, 1];
     28. cell2 = activeWorkSheet.Cells[count, 2];
     29. Data.Add(new YearValueData(){
     30. Year = cell1.Value,
     31. Value = cell2.Value
     32. });
     33. }
     34. // Title is a TextBlock in XAML, this sets the value
     35. Title.Text = activeWorkSheet.Cells[1, 1].Value;
     36. // Close the workbook
     37. excelWorkBook.Close();
     38. // Close the Excel process
     39. excel.Quit();
     40. CleanUpFileSystem(tempDirectory, tempFullPath);
     41. // Populate the DataGrid
     42. ExcelDataGrid.ItemsSource = this.Data;
     43. // Create LineSeries
     44. LineSeries lineSeries = new LineSeries();
     45. lineSeries.ItemsSource = this.Data;
     46. lineSeries.IndependentValueBinding = new Binding("Year");
     47. lineSeries.DependentValueBinding = new Binding("Value");
     48. this.Chart.Series.Add(lineSeries);
     49. // Hide the ImportPanel
     50. this.ImportPanel.Visibility = Visibility.Collapsed;
     51. }
     52. }
     53. }
     54. Visual Basic
     55. Private Sub ImportPanel\_Drop(ByVal sender As Object, ByVal e As DragEventArgs)
     56. If e.Data IsNot Nothing Then
     57. Dim files() As FileInfo = TryCast(e.Data.GetData(DataFormats.FileDrop), FileInfo())
     58. If files.Length > 0 Then
     59. If AutomationFactory.IsAvailable Then
     60. Dim fi = files(0)
     61. If fi.Name.ToLower().Contains(".xls") OrElse fi.Name.ToLower().Contains(".xlsx") Then
     62. VisualStateManager.GoToState(Me, "Loading", True)
     63. Dim myDocuments As String = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
     64. Dim tempDirectory As String = String.Format("{0}\temp", myDocuments)
     65. Dim tempFullPath As String = String.Format("{0}\{1}", tempDirectory, fi.Name)
     66. ' Copy the file to a temp directory in My Documents
     67. CopyFileToTempDirectory(fi, tempDirectory, fi.Name)
     68. 'Dim result = New ObservableCollection(Of YearValueData)()
     69. Dim excel As Object
     70. Try
     71. ' Check to see if Excel is already running
     72. excel = AutomationFactory.GetObject("Excel.Application")
     73. Catch
     74. excel = AutomationFactory.CreateObject("Excel.Application")
     75. End Try
     76. ' Open the excel document
     77. Dim excelWorkBook As Object = excel.Workbooks.Open(tempFullPath)
     78. ' Read the Worksheet
     79. Dim activeWorkSheet As Object = excelWorkBook.ActiveSheet()
     80. ' Cells to Read
     81. Dim cell1, cell2 As Object
     82. Title.Text = activeWorkSheet.Cells(1, 1).Value
     83. ' Iterate through Cells
     84. For count As Integer = 3 To 29
     85. cell1 = activeWorkSheet.Cells(count, 1)
     86. cell2 = activeWorkSheet.Cells(count, 2)
     87. Me.Data.Add(New YearValueData() With {.Year = cell1.Value, .Value = cell2.Value})
     88. Next count
     89. ' Close the workbook
     90. excelWorkBook.Close()
     91. ' Close the Excel process
     92. excel.Quit()
     93. ' Clean up temp file
     94. CleanUpFileSystem(tempDirectory, tempFullPath)
     95. ' Populate the DataGrid
     96. ExcelDataGrid.ItemsSource = Me.Data
     97. ' Create LineSeries
     98. Dim lineSeries As New LineSeries()
     99. lineSeries.ItemsSource = Me.Data
     100. lineSeries.IndependentValueBinding = New Binding("Year")
     101. lineSeries.DependentValueBinding = New Binding("Value")
     102. Me.Chart.Series.Add(lineSeries)
     103. VisualStateManager.GoToState(Me, "DetailsState", True)
     104. Else
     105. ' Display error: "Hey this isn't an Excel file, please select valid excel file"
     106. End If
     107. End If
     108. End If
     109. End If
     110. End Sub

Exercise 4: Checking for updates

* 1. At this point you’ve learned how to read Excel data, open files from anywhere in the hard drive, and work with the hard drive. Moving away working with data, a situation that needs consideration is how do you check for updates in an out-of-browser scenario. This exercise covers how to check if there are updates.
  2. In the starter solution is a button labeled “Check for update”, as seen in the below figure. This is located in MainPage.xaml.
     1. 
     2. Figure
  3. Open MainPage.xaml.cs and add a Click event handler for the CheckForUpdatesButton and for the Application’s CheckAndDownloadUpdateCompleted (this even gets fired after checking if there are updates).
     1. C#
     2. public MainPage()
     3. {
     4. InitializeComponent();
     5. this.CheckForUpdatesButton.Click += new RoutedEventHandler(CheckForUpdatesButton\_Click);
     6. }
     7. Visual Basic
     8. Public Sub New()
     9. InitializeComponent()
     10. AddHandler CheckForUpdatesButton.Click, AddressOf CheckForUpdatesButton\_Click
     11. End Sub
  4. Checking to see if there’s an update is fairly simple. In the CheckForUpdatesButton’s Click event call Application.Current.CheckAndDownloadUpdateAsync(). This method checks for and downloads an updated version of your application.
     1. C#
     2. void CheckForUpdatesButton\_Click(object sender, RoutedEventArgs e)
     3. {
     4. Application.Current.CheckAndDownloadUpdateAsync();
     5. Application.Current.CheckAndDownloadUpdateCompleted += new CheckAndDownloadUpdateCompletedEventHandler(Current\_CheckAndDownloadUpdateCompleted);
     6. }
     7. Visual Basic
     8. Private Sub CheckForUpdatesButton\_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
     9. Application.Current.CheckAndDownloadUpdateAsync()
     10. AddHandler Application.Current.CheckAndDownloadUpdateCompleted, AddressOf Current\_CheckAndDownloadUpdateCompleted
     11. End Sub
  5. After the application has finished checking if there was an update the CheckAndDownloadUpdateCompleted event is fired. Add the below event handler.
     1. C#
     2. void Current\_CheckAndDownloadUpdateCompleted(object sender, CheckAndDownloadUpdateCompletedEventArgs e)
     3. {
     4. if (e.UpdateAvailable)
     5. {
     6. MessageBox.Show("An application update has been downloaded. " +
     7. "Restart the application to run the new version.");
     8. }
     9. else if (e.Error != null)
     10. {
     11. MessageBox.Show(
     12. "An application update is available, but an error has occurred.\n" +
     13. "This can happen, for example, when the update requires\n" +
     14. "a new version of Silverlight or requires elevated trust.\n" +
     15. "To install the update, visit the application home page.");
     16. // LogErrorToServer(e.Error);
     17. }
     18. else
     19. {
     20. MessageBox.Show("There is no update available.");
     21. }
     22. }
     23. Visual Basic
     24. Private Sub Current\_CheckAndDownloadUpdateCompleted(ByVal sender As Object, ByVal e As CheckAndDownloadUpdateCompletedEventArgs)
     25. If e.UpdateAvailable Then
     26. MessageBox.Show("An application update has been downloaded. " & "Restart the application to run the new version.")
     27. ElseIf e.Error IsNot Nothing Then
     28. MessageBox.Show("An application update is available, but an error has occurred." & vbLf & "This can happen, for example, when the update
     29. requires" & vbLf & "a new version of Silverlight or requires elevated trust."
     30. & vbLf & "To install the update, visit the application home page.")
     31. 'LogErrorToServer(e.Error);
     32. Else
     33. MessageBox.Show("There is no update available.")
     34. End If
     35. End Sub
  6. This Figure shows a message box that’s displayed if there is an update from the above CheckAndDownloadUpdateCompleted event.
     1. 
     2. Figure
  7. This Figure shows a message box that’s displayed if there is not an update from the above CheckAndDownloadUpdateCompleted event.
     1. 
     2. Figure

Summary

In this exercise you examined advanced features of Silverlight’s out-of-browser feature. You learned how to open an Excel file from anywhere on the hard drive by writing its content to a temporary directory. From there, Silverlight read the data from the worksheet and displayed it to a grid and chart. The application satisfied the following requirements:

* + Drop files onto the application to be imported
  + Open a file from anywhere on the hard drive
  + Save a file to a temporary directory and file on the hard drive
  + Loop through an Excel to extract data
  + Enable the user to check for a new version of the application

This lab provided the building blocks on how to out-of-browser in a business setting. By applying skills learned from previous labs, you should now be able to read in the data, create a WCF service, and send the imported data to be stored on the server.