



Microsoft®
Silverlight™

Hands-On Lab

Silverlight 4 – RichTextBox Control

Contents

Introduction.....	3
Exercise 1 – Getting Started	4
Task 1 – Building a Text Editor UI.....	5
Task 2 – Embedded Elements	14
Task 3 – Persisting content	17
Exercise 2 –Text Services and Printing.....	20
Task 1 – Clipboard Support.....	20
Task 2 – Printing from Silverlight	22
Task 3 – Highlight and XAML functionality	23
Task 4 – Drag and Drop Support	24
Exercise 3 – Multilingual Support	28
Task 1 –BiDi (Bi-Directional) Support	28
Task 2 –Culture based UI with Resources.....	29
Conclusion	31

Introduction

Silverlight 4 offers many new features for text editing including a new RichTextBox control. This new control includes clipboard support, text formatting and BiDi (Bidirectional) text support for input and output. The control also enables applications to show rich text and allows users to input formatted text (Bold, Italic, Underline, Font family, Font size and Font color), highlighting of text, and embedded elements such as Hyperlinks and Images.

Another useful feature is Silverlight's Drop Target API. This feature allows end users to drag & drop word documents directly in to the RichTextBox control. The control also can return Xaml allowing the format of rich text to easily be stored and retrieved.

In this Lab we'll learn how to build a text editor that supports rich text, clipboard copy & paste, and printing. We'll also learn how we can use the new BiDi support to customize the application's UI according to the client's culture. Finally, we will learn how to implement the new Drag & Drop API feature to easily load text or Word document contents into the RichTextBox control.

Estimated completion time for this lab is 75 minutes.

Exercise 1 – Getting Started

The goal of this exercise is to familiarize the student with the existing starter application and add new functionality into the application.

1. Start Visual Studio 2010
 2. On the **File** menu click **Open → Project/Solution...**
 - a. Alternatively, from Visual Studio Start Page click **Open Project...**
 - i. At the Open Project dialog navigate to the Lab installation folder
 - ii. Navigate to RichTextBox\Source\Ex01\Begin folder
 - iii. Click the **RichTextBox.sln** file and then click the **Open** button
 3. The project will not currently build since we'll be adding specific functionality. You can ignore any warnings that may appear at this point.
 4. Set the RichTextBox.Web project as the startup project by right clicking the project in the Solution Explorer and selecting "Set as Startup Project"
-

Task 1 – Building a Text Editor UI

In order to use the rich text formatting offered by the **RichTextBox** control we need a set of buttons that will allow end users to perform text formatting functions. We'll start by building a ribbon user interface much like you've seen in Word 2007 or higher.

1. If you haven't opened the Starter project open it now (see previous section for detailed instructions)
2. Double-click the XAML tab below the designer window to open the XAML code editor in full screen mode
3. Open MainPage.xaml and locate the <!-- RichTextBoxControl --> comment. Replace the existing RichTextBox element with the following:

XAML

```
<RichTextBox x:Name="rtb" SelectionChanged="rtb_SelectionChanged"
  VerticalScrollBarVisibility="Auto" KeyUp="rtb_KeyUp"
  BorderBrush="{x:Null}" Margin="8,10,10,8" FontSize="20" TextWrapping="Wrap"
  MouseRightButtonDown="rtb_MouseRightButtonDown"
  MouseRightButtonUp="rtb_MouseRightButtonUp" MouseMove="rtb_MouseMove"
  DragEnter="rtb_DragEnter" DragLeave="rtb_DragLeave"/>
```

4. In MainPage.xaml file. Scroll to the bottom and locate <!-- TODO Toolbar goes here --> which is located toward the bottom of the file. We will add a Grid after this comment to create containers and placeholders for buttons. Add the following code:

XAML

```
<Grid Margin="3,3,3,1">
  <Grid.RowDefinitions>
    <RowDefinition Height="27"/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="102"/>
    <ColumnDefinition Width="176"/>
    <ColumnDefinition Width="184"/>
    <ColumnDefinition Width="73"/>
    <ColumnDefinition Width="106"/>
    <ColumnDefinition Width="80"/>
    <ColumnDefinition Width="80"/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

  <Rectangle Margin="0,-26,0,2" Grid.RowSpan="1" Fill="{StaticResource
    MenuDividerBrush}" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
  <Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
    Grid.Column="1" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
```

```

<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="2" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="3" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="4" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="5" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="6" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>
<Rectangle Margin="2,-26,0,2" Fill="{StaticResource MenuDividerBrush}"
  Grid.Column="7" Grid.Row="1" RadiusX="2" RadiusY="2" Stroke="#FFA5BED4"/>

<!-- TODO - Font Buttons-->

<!-- TODO - Clipboard Buttons-->

<!-- TODO - Insert Buttons-->

<!-- TODO - Print Button-->

<!-- TODO - Display, Highlight, Xaml Buttons-->

<TextBlock Style="{StaticResource MenuLabel}" Text="Clipboard" Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="Font" Grid.Column="1"
  Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="Insert" Grid.Column="2"
  Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="Print" Grid.Column="3"
  Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="Display" Grid.Column="4"
  Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="Highlight"
  Grid.Column="5" Grid.Row="1"/>
<TextBlock Style="{StaticResource MenuLabel}" Text="XAML" Grid.Column="6"
  Grid.Row="1"/>
</Grid>

```

5. Press **F5** to Run the application. Notice the Grid has columns to act as placeholders for the buttons and functionality that will be added.



Figure 1

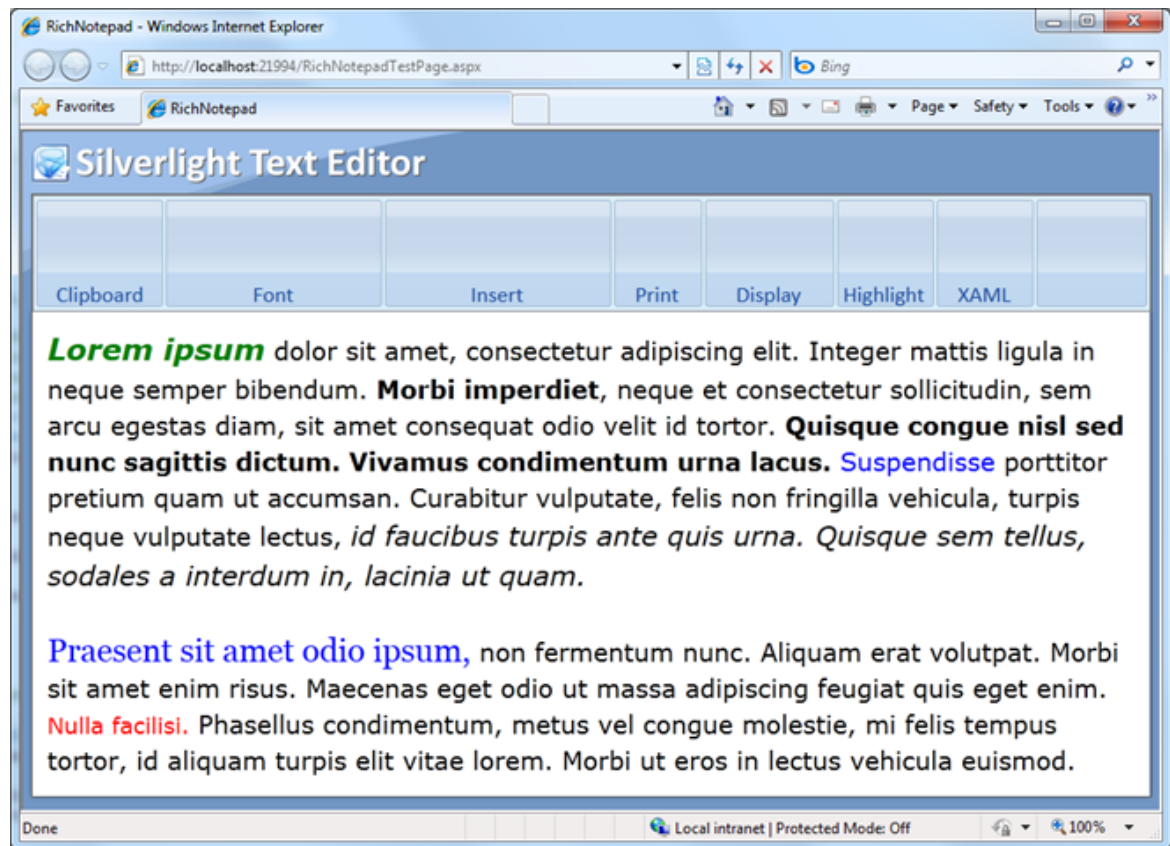
Browser showing grid with placeholder for buttons

- Now we'll load a file to pre-populate the RichTextBox with some formatted Xaml text. Open MainPage.xaml.cs file and near the top of the file locate // TODO Add sample Xaml file here in the MainPage_Loaded event. Add the following code to load from a file named SampleNew.sav and assign to the RichTextBox control's Xaml property:

C#

```
StreamResourceInfo sr = App.GetResourceStream(new
    Uri("/RichNotepad;component/sampleNew.sav", UriKind.Relative));
StreamReader sread = new StreamReader(sr.Stream);
string xaml = sread.ReadToEnd();
rtb.Xaml = xaml;
sread.Close();
```

- Press F5 to run the project and notice that we now have formatted text loaded into the RichTextBox control.

**Figure 2**

Browser showing dynamic text loaded

- Close the browser and return back to Visual Studio
- Go to MainPage.xaml and scroll to the bottom. Locate the <!-- TODO - Font Buttons--> comment. Insert the following code to add the font formatting buttons:

XAML

```
<!--Fonts Face Combo-->
<ComboBox x:Name="cmbFonts" FontFamily="{Binding SelectedItem.FontFamily,
  RelativeSource={RelativeSource Self}}" FontSize="{Binding
  SelectedItem.FontSize, RelativeSource={RelativeSource Self}}" Grid.Column="1"
  Margin="6,0,55,0" Height="22" VerticalAlignment="Bottom" >
  <ComboBoxItem Content="Arial" Tag="Arial" FontFamily="Arial" FontSize="12"/>
  <ComboBoxItem Content="Arial Black" Tag="Arial Black" FontFamily="Arial
  Black" FontSize="12"/>
  <ComboBoxItem Content="Calibri" Tag="Calibri" FontFamily="Calibri"
  IsSelected="True" FontSize="14"/>
  <ComboBoxItem Content="Comic Sans MS" Tag="Comic Sans MS"
  FontFamily="Comic Sans MS" FontSize="12"/>
  <ComboBoxItem Content="Courier New" Tag="Courier New"
  FontFamily="Courier New" FontSize="12"/>
  <ComboBoxItem Content="Georgia" Tag="Georgia" FontFamily="Georgia"
  FontSize="12"/>
  <ComboBoxItem Content="Lucida Sans Unicode" Tag="Lucida Sans Unicode"
  FontFamily="Lucida Sans Unicode" FontSize="12"/>
  <ComboBoxItem Content="Portable User Interface" Tag="{x:Null}"
  FontFamily="Portable User Interface" FontSize="12"/>
  <ComboBoxItem Content="Times New Roman" Tag="Times New Roman"
  FontFamily="Times New Roman" FontSize="12"/>
  <ComboBoxItem Content="Trebuchet MS" Tag="Trebuchet MS"
  FontFamily="Trebuchet MS" FontSize="12"/>
  <ComboBoxItem Content="Verdana" Tag="Verdana" FontFamily="Verdana"
  FontSize="12"/>
  <ComboBoxItem Content="Webdings" Tag="Webdings" FontSize="12"/>
</ComboBox>

<ComboBox x:Name="cmbFontSizes" Width="44" FontSize="14"
HorizontalAlignment="Right" Margin="0,0,6,0" Grid.Column="1" Height="22"
VerticalAlignment="Bottom">
  <ComboBoxItem Content="8" Tag="8"/>
  <ComboBoxItem Content="9" Tag="9"/>
  <ComboBoxItem Content="10" Tag="10"/>
  <ComboBoxItem Content="11" Tag="11"/>
  <ComboBoxItem Content="12" Tag="12"/>
  <ComboBoxItem Content="14" Tag="14"/>
  <ComboBoxItem Content="16" Tag="16" IsSelected="True"/>
  <ComboBoxItem Content="18" Tag="18"/>
  <ComboBoxItem Content="20" Tag="20"/>
  <ComboBoxItem Content="22" Tag="22"/>
  <ComboBoxItem Content="24" Tag="24"/>
  <ComboBoxItem Content="26" Tag="26"/>
  <ComboBoxItem Content="28" Tag="28"/>
  <ComboBoxItem Content="36" Tag="36"/>
  <ComboBoxItem Content="48" Tag="48"/>
  <ComboBoxItem Content="72" Tag="72"/>
</ComboBox>
```

```
<StackPanel Grid.Column="1" Height="22" Margin="6,5,0,0" Orientation="Horizontal"
Grid.Row="1" VerticalAlignment="Top">

  <!--Font Formatting Buttons-->
  <Button x:Name="btnBold" Margin="0,0,1,0">
    <ToolTipService.ToolTip>
      <ToolTip FontSize="16" Content="{Binding tooltip_Bold,
        Source={StaticResource localizedStrings}}"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/Bold.png" d:IsLocked="True"/>
  </Button>

  <Button x:Name="btnItalic" Margin="0,0,1,0">
    <ToolTipService.ToolTip>
      <ToolTip FontSize="16" Content="{Binding tooltip_Italic,
        Source={StaticResource localizedStrings}}"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/Italic.png"/>
  </Button>

  <Button x:Name="btnUnderline" Margin="0,0,13,0">
    <ToolTipService.ToolTip>
      <ToolTip FontSize="16" Content="Underline"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/Underline.png"/>
  </Button>

  <ComboBox x:Name="cmbFontColors" SelectedIndex="0" Margin="39,0,0,0"
HorizontalAlignment="Right">
    <ComboBoxItem Tag="FFFF0000">
      <Rectangle Width="22" Height="14" Fill="Red" Margin="2,0,0,0" />
    </ComboBoxItem>
    <ComboBoxItem Tag="FF008000">
      <Rectangle Width="22" Height="14" Fill="Green" Margin="2,0,0,0" />
    </ComboBoxItem>
    <ComboBoxItem Tag="FF0000FF">
      <Rectangle Width="22" Height="14" Fill="Blue" Margin="2,0,0,0" />
    </ComboBoxItem>
    <ComboBoxItem Tag="FFFFFF00">
      <Rectangle Width="22" Height="14" Fill="Yellow" Margin="2,0,0,0" />
    </ComboBoxItem>
    <ComboBoxItem Tag="FF000000" IsSelected="True">
      <Rectangle Width="22" Height="14" Fill="Black" Margin="2,0,0,0" />
    </ComboBoxItem>
  </ComboBox>
```

```
</StackPanel>
```

- In the code you just inserted into MainPage.xaml, find the button named **btnBold**. Add a new Click event handler (use the default name).

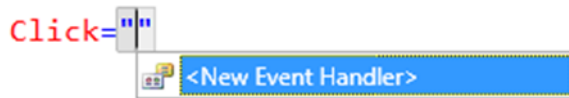


Figure 3

Event Handler Generation from XAML Editor

- Right-click `btnBold_Click` and select `Navigate to Event Handler` from the context menu

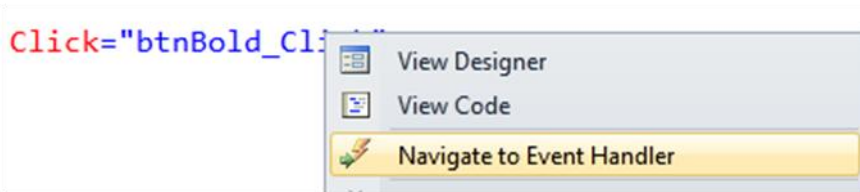


Figure 4

Navigate to Event Handler from XAML Editor

- The last action will take you into the source code editor and to the event handler:

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

- Since we want to set the selected text to bold, we'll add the following **if** statement to check whether the user selected some text prior to pressing the button. Add the following code in the `btnBold_Click` method:

```
C#
if (rtb != null && rtb.Selection.Text.Length > 0)
{
}
ReturnFocus();
```

- Once we have a selection, all that is left to do is to change the font's weight from normal to bold or vice versa according to whether the button is toggled or not. We'll need to check the current font style and apply the opposite (in case the selection is mixed we'll remove the bold style). Add the following code inside the **if** statement to perform the style modification:

```
C#
if (rtb.Selection.GetProperty(Run.FontWeightProperty) is FontWeight &&
```

```
((FontWeight)rtb.Selection.GetPropertyValue(Run.FontWeightProperty))
== FontWeights.Normal)
    rtb.Selection.ApplyPropertyValue(Run.FontWeightProperty,
FontWeights.Bold);
else
    rtb.Selection.ApplyPropertyValue(Run.FontWeightProperty,
FontWeights.Normal);
```

15. Add a Click event handler for **btnItalic** (following the same process you followed for btnBold above).
16. Place the following code in the btnItalic_Click event handler and take a moment to look through the code.

```
C#
if (rtb != null && rtb.Selection.Text.Length > 0)
{
    if (rtb.Selection.GetPropertyValue(Run.FontStyleProperty) is FontStyle &&
        ((FontStyle)rtb.Selection.GetPropertyValue(Run.FontStyleProperty))
== FontStyles.Normal)
        rtb.Selection.ApplyPropertyValue(Run.FontStyleProperty,
FontStyles.Italic);
    else
        rtb.Selection.ApplyPropertyValue(Run.FontStyleProperty,
FontStyles.Normal);
}
ReturnFocus();
```

17. Add a Click event handler for **btnUnderline**
18. Place the following code in the btnUnderline_Click event handler:

```
C#
if (rtb != null && rtb.Selection.Text.Length > 0)
{
    if (rtb.Selection.GetPropertyValue(Run.TextDecorationsProperty) == null)
        rtb.Selection.ApplyPropertyValue(Run.TextDecorationsProperty,
TextDecorations.Underline);
    else
        rtb.Selection.ApplyPropertyValue(Run.TextDecorationsProperty, null);
}
ReturnFocus();
```

19. Next, we'll handle font family changes. When the user changes the font in the fonts ComboBox control, we'll change the font family of the current selected text. Find the ComboBox named **cmbFonts** and add an event handler for the SelectionChanged event.
20. Add the following code into the cmbFonts_SelectionChanged event handler:

```
C#
```

```
if (rtb != null && rtb.Selection.Text.Length > 0)
{
    rtb.Selection.ApplyPropertyValue(Run.FontFamilyProperty,
        new FontFamily((cmbFonts.SelectedItem as
ComboBoxItem).Tag.ToString()));
}
ReturnFocus();
```

21. Let's do the same to modify the text size. Find the ComboBox named **cmbFontSizes**, add a default event handler for the SelectionChanged event and add the following code to the event handler:

```
C#
if (rtb != null && rtb.Selection.Text.Length > 0)
{
    rtb.Selection.ApplyPropertyValue(Run.FontSizeProperty,
        double.Parse((cmbFontSizes.SelectedItem as
ComboBoxItem).Tag.ToString()));
}
ReturnFocus();
```

22. Find the **cmbFontColors** ComboBox and examine how each color is represented as a string of HEX values representing the color and stored in the "Tag" property of the ComboBoxItem element. In order to break that string into ARGB (Alpha/Red/Green/Blue) format, we'll need to do some parsing.
23. Create a default event handler for the SelectionChanged event of **cmbFontColors** and add the following code to the event handler:

```
C#
if (rtb != null && rtb.Selection.Text.Length > 0)
{
    string color = (cmbFontColors.SelectedItem as ComboBoxItem).Tag.ToString();

    SolidColorBrush brush = new SolidColorBrush(Color.FromArgb(
        byte.Parse(color.Substring(0, 2),
System.Globalization.NumberStyles.HexNumber),
        byte.Parse(color.Substring(2, 2),
System.Globalization.NumberStyles.HexNumber),
        byte.Parse(color.Substring(4, 2),
System.Globalization.NumberStyles.HexNumber),
        byte.Parse(color.Substring(6, 2),
System.Globalization.NumberStyles.HexNumber)));

    rtb.Selection.ApplyPropertyValue(Run.ForegroundProperty, brush);
}
```

24. Run the application by pressing F5.

25. Enter some text (or use the pre-loaded text), select it and press the bold button. The text has become bold. Press the bold button again and the text will return to normal. If you select mixed text (some characters bold and some normal) and press the bold button, the text will return to normal. Try experimenting with the other text formatting features including fonts and font colors.
-

Task 2 – Embedded Elements

In this exercise we will allow users to add hyperlinks, images, datagrids and calendars into the RichTextBox control.

1. In MainPage.xaml locate the comment `<!-- TODO - Insert Buttons-->` and insert the following code to create the buttons for inserting an image, hyperlink, DataGrid and calendar.

XAML

```
<Button x:Name="btnImage" HorizontalAlignment="Left" Grid.Column="2"
  Margin="13,-22,0,24" Grid.RowSpan="1" Grid.Row="1">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="{Binding tooltip_Image,
      Source={StaticResource localizedStrings}}"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/Image.png" Height="32" Width="30"/>
</Button>

<Button x:Name="btnHyperlink" Grid.Column="2" Margin="55,-22,0,24"
  Grid.RowSpan="1" Grid.Row="1" HorizontalAlignment="Left">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="{Binding tooltip_Hyperlink,
      Source={StaticResource localizedStrings}}"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/Hyperlink_big.png" Width="30" Height="32"/>
</Button>

<Button x:Name="btnDatagrid" Grid.Column="2" Grid.RowSpan="1"
  Margin="97,-22,0,24" HorizontalAlignment="Left" Grid.Row="1">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="Insert Datagrid"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/datagrid.png" Width="29" Height="32"/>
</Button>

<Button x:Name="btnCalendar" Grid.Column="2" Grid.RowSpan="1"
  Margin="138,-22,0,24" HorizontalAlignment="Left" Grid.Row="1">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="Insert Calendar"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/calendar.png" Width="29" Height="32"/>
</Button>
```

2. Locate the **btnImage** button. Add a default event handler for the Click event and navigate to the event handler.
3. In order to add an image to the RichTextBox we need to create an InlineUIContainer object that will contain the image presented by the RichTextBox. Add the following code to the btnImage_Click event handler function:

C#

```
InlineUIContainer container = new InlineUIContainer();
container.Child = getImage();
rtb.Selection.Insert(container);
ReturnFocus();
```

- Now that we have a container, we can load a sample image into it (you can find the image in the ClientBin folder of the Web Project). Add the following code into MainPage.xaml.cs to create a method capable of loading an image and placing it in the container:

C#

```
private Image getImage()
{
    return CreateImageFromUri(new Uri("desert.jpg",
        UriKind.RelativeOrAbsolute), 200, 150);
}
```

- Compile and run the application by pressing F5. Press the insert image button and an image will appear in the RichTextBox control.
- To add a hyperlink to the RichTextBox control we'll use a child window to allow the user to input a hyperlink and a description for it. The ChildWindow class was introduced in Silverlight 3 and is used to implement popup windows you can interact with. Find the file InsertURL.xaml and the file InsertURL.xaml.cs and take a moment to look through their code. Examine the validation code in the InsertURL.xaml.cs class.
- Back in MainPage.xaml, find the **btnHyperlink** button and add an event handler for the Click event. To show the InsertURL child window add the following code to the event handler:

C#

```
InsertURL cw = new InsertURL(rtb.Selection.Text);
cw.HasCloseButton = false;
cw.Show();
```

- The previous code will show the child window, but after confirming the URL we need to fetch it from the child window and place it in the RichTextBox. Add the following code to handle the Close event of the child window (place the code before calling the Show method)

C#

```
cw.Closed += (s, args) =>
{
    if (cw.DialogResult.Value)
    {
    }
};
```

- Next, we'll create a Hyperlink object and place it inside the RichTextBox. Place the following code inside the **if** statement from the last step.

```
C#
Hyperlink hyperlink = new Hyperlink();
hyperlink.TargetName = "_blank";
hyperlink.NavigateUri = new Uri(cw.txtURL.Text);

if (cw.txtURLDesc.Text.Length > 0)
    hyperlink.Inlines.Add(cw.txtURLDesc.Text);
else
    hyperlink.Inlines.Add(cw.txtURL.Text);

rtb.Selection.Insert(hyperlink);
```

10. Locate the **btnDatagrid** button in MainPage.xaml, add a Click event handler and add the following code into the event handler:

```
C#
InlineUIContainer container = new InlineUIContainer();
container.Child = getDataGrid();
rtb.Selection.Insert(container);
ReturnFocus();
```

11. Add the following method into the MainPage.xaml.cs file to create a new DataGrid instance:

```
C#
private DataGrid getDataGrid()
{
    DataGrid dg = new DataGrid();
    dg.AutoGenerateColumns = true;
    dg.Width = 500;
    dg.Height = 150;
    dg.ItemsSource = Customer.GetSampleCustomerList();
    dg.Style = (Style)this.Resources["DataGridStyle1"];

    return dg;
}
```

12. Locate the **btnCalendar** button in MainPage.xaml, add a Click event handler and add the following code into the event handler:

```
C#
InlineUIContainer container = new InlineUIContainer();
container.Child = getCalendar();
rtb.Selection.Insert(container);
ReturnFocus();
```

13. Add the following method into the MainPage.xaml.cs file to create a new Calendar instance:

```
C#
private Calendar getCalendar()
{
```

```

Calendar cal = new Calendar();
cal.Width = 179;
cal.Height = 169;
cal.FontFamily = new FontFamily("Portable User Interface");
cal.Style = (Style)this.Resources["CalendarStyle1"];

return cal;
}

```

- At this point we've seen that InlineUIContainer can act as the container for different types of objects. Compile and run the application by pressing F5. Press the image, hyperlink, datagrid and calendar buttons to test the functionality.

Task 3 – Persisting content

Now that our editor supports inputting text and styling it let's see how we can save documents.

- Open MainPage.xaml, locate the comment remark `<!-- TODO - Open, Save, New Buttons-->`, and add the following stack panel control which will house our New, Open, and Save buttons.

XAML

```

<StackPanel Grid.RowSpan="1" Grid.Column="4" Margin="8" VerticalAlignment="Top"
HorizontalAlignment="Right" Orientation="Horizontal" Grid.Row="0">

    <!--New Button-->
    <Button x:Name="btnNew" HorizontalAlignment="Left" Height="22"
        Margin="0,0,0,0" VerticalAlignment="Top" Width="22">
        <ToolTipService.ToolTip>
            <ToolTip FontSize="16" Content="{Binding tooltip_New,
                Source={StaticResource localizedStrings}}"/>
        </ToolTipService.ToolTip>
        <Image Source="Images/New.png"/>
    </Button>
    <!--Open Existing Button -->
    <Button x:Name="btnOpen" HorizontalAlignment="Left" Height="22"
        Margin="0,0,0,4" VerticalAlignment="Top" Width="22">
        <ToolTipService.ToolTip>
            <ToolTip FontSize="16" Content="{Binding tooltip_Open,
                Source={StaticResource localizedStrings}}"/>
        </ToolTipService.ToolTip>
        <Image Source="Images/Open.png"/>
    </Button>
    <!--Save Existing Button -->
    <Button x:Name="btnSave" HorizontalAlignment="Left" Height="22"
        VerticalAlignment="Top" Width="22">
        <ToolTipService.ToolTip>
            <ToolTip FontSize="16" Content="{Binding tooltip_Save,
                Source={StaticResource localizedStrings}}"/>
        </ToolTipService.ToolTip>

```

```
        <Image Source="Images/Save.png"/>
    </Button>

</StackPanel>
```

2. In MainPage.xaml locate the **btnNew** button and add a default event handler for the Click event. Add the following code to the event handler:

```
C#
bool clear = true;
if (IsDirty)
{
    if (MessageBox.Show("All changes will be lost. Continue?",
        "Continue", MessageBoxButton.OKCancel) == MessageBoxResult.Cancel)
    {
        clear = false;
    }
}

if (clear)
{
    rtb.Blocks.Clear();
    IsDirty = false;
}
```

3. Go back to the MainPage.xaml, locate the **btnSave** button and add a default event handler for the Click event. Add the following into the event handler:

```
C#
string xaml = rtb.Xaml;
SaveFileDialog sfd = new SaveFileDialog();
sfd.DefaultExt = ".sav";
sfd.Filter = "Saved Files|*.sav|All Files|*.*";
if (sfd.ShowDialog().Value)
{
}
}
```

4. All that is left to do is take the content, write it into the selected file and reset an IsDirty variable. Add the following code inside the if statement:

```
C#
using (FileStream fs = (FileStream)sfd.OpenFile())
{
    System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();
    byte[] buffer = enc.GetBytes(xaml);
    fs.Write(buffer, 0, buffer.Length);
    fs.Close();
    IsDirty = false;
}
```

```
}
```

5. Before we can test this code, we need to be able to open the saved file. Go to MainPage.xaml and locate the **btnOpen** button. Add a default event handler for the Click event and navigate to it.
6. Next we'll use the OpenFileDialog class to ask the user to select which document to open. Place the following code after calling the btnOpen_Click method:

```
C#
```

```
OpenFileDialog ofd = new OpenFileDialog();  
ofd.Multiselect = false;  
ofd.Filter = "Saved Files|*.sav|All Files|*.*";  
  
if (ofd.ShowDialog().Value)  
{  
  
}
```

7. Add the following code into the **if** statement:

```
C#
```

```
FileInfo fi = ofd.File;  
using (StreamReader reader = fi.OpenText())  
{  
    rtb.Xaml = reader.ReadToEnd();  
}
```

8. Compile and run the application by pressing F5. Enter some text and test the save and load features.
-

Exercise 2 –Text Services and Printing

In this exercise we'll learn how we can perform operations such as using the clipboard to copy and paste text into the rich text area and print our rich text content.

Task 1 – Clipboard Support

Silverlight 4 provides support for clipboard operations. In task we'll add functionality to support cut, copy and paste.

1. Open the MainPage.xaml file, and locate the comment remark <!-- TODO - Clipboard Buttons-->. Add the following code for the Cut, Copy and Paste buttons.

XAML

```
<Button x:Name="btnPaste" Margin="17,-22,33,24" Grid.Row="1"
  HorizontalAlignment="Left">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="Paste"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/Paste_big.png" Height="32" Width="29"/>
</Button>
<Button x:Name="btnCut" Margin="0,0,17,0" HorizontalAlignment="Right" Width="22"
  VerticalAlignment="Bottom">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="Cut"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/Cut.png" d:IsLocked="True"/>
</Button>
<Button x:Name="btnCopy" Margin="0,5,17,0" VerticalAlignment="Top" Grid.Row="1"
  HorizontalAlignment="Right" Width="22">
  <ToolTipService.ToolTip>
    <ToolTip FontSize="16" Content="Copy"/>
  </ToolTipService.ToolTip>
  <Image Source="Images/Copy.png" d:IsLocked="True"/>
</Button>
```

2. Add a default event handler for the Click event in each of the buttons above.
3. In order to access the clipboard, we'll use the Clipboard class introduced in Silverlight 4 which supports saving and loading strings in the clipboard. Add the following code to the event handler for **btnCopy** to allow saving strings in the clipboard:

C#

```
Clipboard.SetText(rtb.Selection.Text);
ReturnFocus();
```

4. Find the Click event handler for the **btnPaste** button and add the following code to read strings from the clipboard

C#

```
rtb.Selection.Text = Clipboard.GetText();  
ReturnFocus();
```

5. Find the Click event handler for the **btnCut** button and add the following code:

C#

```
Clipboard.SetText(rtb.Selection.Text);  
rtb.Selection.Text = "";  
ReturnFocus();
```

6. Compile and run the application by pressing F5. Add some text in the RichTextBox and press the Copy button. Open Notepad and paste the string (CTRL+V).
 7. Return to the application, press the Create New document button (click OK if prompted) and then press the Paste button. The copied text will be pasted in the RichTextBox. The Text Editor application now has clipboard support.
-

Task 2 – Printing from Silverlight

Silverlight 4 enables developers to use installed printers in order to print application content. In this exercise we will use this functionality and print the document content.

1. To print the contents of the RichTextBox we will use the PrintDocument class introduced in Silverlight 4. Open the MainPage.xaml file, and locate the comment remark <!-- TODO - Print Buttons-->. Add the following code for the Print button:

```
C#
<Button x:Name="btnPrint" Grid.Column="3" Margin="20,-22,0,24"
    HorizontalAlignment="Left" Grid.Row="1">
    <ToolTipService.ToolTip>
        <ToolTip FontSize="16" Content="{Binding tooltip_Print,
            Source={StaticResource localizedStrings}}"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/Print_big.png" Width="30"/>
</Button>
```

2. Add a default event handler for the Click event for the **Print** button.
3. Navigate to the event handler function and add the following code

```
C#
PrintDocument theDoc = new PrintDocument();
string DocumentName = "Silverlight 4 Text Editor - Opened Document";
theDoc.Print(DocumentName);
```

4. As you can see, we've called the Print method, but we haven't told it what to print. To set the printed area we'll need to handle the PrintDocument.PrintPage event. Add the following code to handle the event (place the code before calling the "Print" method)

```
C#
theDoc.PrintPage += (s, args) =>
{
    args.PageVisual = rtb;
    args.HasMorePages = false;
};
```

5. We'll also add a MessageBox to notify the user the printing was successful. Place the following code before calling the Print method:

```
C#
theDoc.EndPrint += (s, args) =>
{
    MessageBox.Show("The document printed successfully", "Text Editor",
        MessageBoxButton.OK);
};
```

6. Compile and run the application by pressing F5. Enter text and add an image to the content, then press the Print button.
7. In the printing dialog, select one of the installed printers or print to the "Microsoft XPS Document Writer" printer. Save the file and after the printing is finished, open the .xps file. Examine how the printed content is similar to the content displayed in the rich text control.

Task 3 – Highlight and XAML functionality

In this exercise we will build upon the controls added thus far. We will add highlight functionality to the RichTextBox. We will also add the ability to toggle between traditional rich text view and the Xaml markup generated for the contents of the RichTextBox.

1. Open MainPage.xaml and locate the comment remark `<!-- TODO - Display, Highlight, Xaml Buttons-->`. Add the following code for the Highlight and Xaml buttons:

XAML

```
<!-- Highlight Button-->
<ToggleButton x:Name="btnHighlight" Checked="btnHighlight_Checked"
    Unchecked="btnHighlight_Checked" Margin="17,-22,15,24"
    Grid.Column="5" Grid.Row="1" HorizontalAlignment="Left">
    <ToolTipService.ToolTip>
        <ToolTip FontSize="16" Content="Show Highlight"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/Annotation_New.png"/>
</ToggleButton>

<!-- Xaml Buttons-->
<ToggleButton x:Name="btnMarkUp" Checked="btnMarkUp_Checked"
    Unchecked="btnMarkUp_Checked" Margin="10,-22,10,24" Grid.Column="6"
    Grid.Row="1" HorizontalAlignment="Left">
    <ToolTipService.ToolTip>
        <ToolTip FontSize="16" Content="Show XAML"/>
    </ToolTipService.ToolTip>
    <TextBlock FontSize="25" Text="&lt;&lt;>>"/>
</ToggleButton>
```

2. Examine the code and above and note that the event handlers for Checked and Unchecked have already been assigned to each Toggle Button control.
3. Right-click on the Checked event handler for the Xaml Markup Button and navigate to the corresponding code in MainPage.xaml.cs. Place the following code into the event:

C#

```
if (btnMarkUp.IsChecked.Value)
{
    xamlTb.Visibility = System.Windows.Visibility.Visible;
```

```
xamlTb.IsTabStop = true;
xamlTb.Text = rtb.Xaml;
}
else
{
    rtb.Xaml = xamlTb.Text;
    xamlTb.Visibility = System.Windows.Visibility.Collapsed;
    xamlTb.IsTabStop = false;
}
```

4. Locate the Highlight Button's btnHighlight_Checked method in the code. As you study the code you'll see how the TextPointer tracks the current position of the caret to gain a starting point. The struct Rect is used to find the populated area of the rectangle that the user (using Mouse Events) placed over the text.
5. Compile and run the application by pressing F5. Enter text, click the Highlight Button, and as you enter the RichTextBox your mouse will highlight the current row you are on. Left mouse click and drag across some text and you can specifically highlight a certain part of the text. Clicking the Highlight Button again will turn off this feature.
6. Now click the Xaml Button. You will see Xaml markup code used to display the presentation version of the text within the RichTextBox.

Task 4 – Drag and Drop Support

New to Silverlight 4 is the Drop Target API. In this exercise we will demonstrate the use of this API by adding drag and drop functionality of Word documents and text files right into the RichTextBox.

1. Open MainPage.xaml file and locate the RichTextBox control named **rtb**. Add the following attributes to it:

XAML

2. `Drop="rtb_Drop" AllowDrop="True"`
3. Open MainPage.xaml.cs and add the following code:

C#

```
private void rtb_Drop(object sender, System.Windows.DragEventArgs e)
{
    VisualStateManager.GoToState(this, "Normal", true);
    if (e.Data != null)
    {
        IDataObject f = e.Data as IDataObject;

        if (f != null)
        {
            object data = f.GetData(DataFormats.FileDrop);
            FileInfo[] files = data as FileInfo[];
        }
    }
}
```

```
if (files != null)
{
    foreach (FileInfo file in files)
    {
        if (file != null)
        {
            if (file.Extension.Equals(".txt"))
            {
                try
                {
                    Stream sr = file.OpenRead();
                    string contents;
                    using (StreamReader reader =
                        new StreamReader(sr))
                    {
                        contents = reader.ReadToEnd();
                    }
                    rtb.Selection.Text = contents;
                }
                catch (IOException ex)
                {
                    //check if message is for a File IO
                    docOpenedError();
                }
            }
            else if (file.Extension.Equals(".docx"))
            {
                try
                {
                    Stream sr = file.OpenRead();
                    string contents;

                    StreamResourceInfo zipInfo =
                        new StreamResourceInfo(sr, null);
                    StreamResourceInfo wordInfo =
                        Application.GetResourceStream(zipInfo,
                            new Uri("word/document.xml", UriKind.Relative));

                    using (StreamReader reader =
                        new StreamReader(wordInfo.Stream))
                    {
                        contents = reader.ReadToEnd();
                    }
                    XDocument xmlFile = XDocument.Parse(contents);
                    XNamespace w =
                        "http://schemas.openxmlformats.org/wordprocessingml/2006/main";

                    var query = from xp in xmlFile.Descendants(w +
                        "p")
```



```
}
```

4. Compile and run the application by pressing F5. Locate the "Helpers" folder contained within this Lab's folders (ie:\RichTextBox\Source\Helpers). Drag and drop the file SampleTextFile.txt into the Rich Text Box control. Now do the same for the file Hamlet.docx.
-

Exercise 3 – Multilingual Support

Bi-Directional (BiDi) support is new in Silverlight 4 and allows applications to display controls in both left-to-right flow and in right-to-left (for languages such as Arabic, Persian or Hebrew). Silverlight 4 also supports Bi-Directional input to allow users to input text written from right to left (for example, Hebrew text).

In this exercise we will learn how to use the application's culture to determine the direction of the controls and how we can use culture based resources.

Task 1 –BiDi (Bi-Directional) Support

Silverlight 4 adds support for bi-directional text. In this exercise we will add support for Right-To-Left languages such as Hebrew and Arabic.

1. Close the existing solution in Visual Studio if one is currently open
2. On the **File** menu click **Open → Project/Solution...**
3. At the Open Project dialog navigate to the Lab installation folder
4. Navigate to **RichTextBox\Source\Ex03\Begin** folder
5. Click the **RichTextBox.sln** file and then click the **Open** button
6. Open MainPage.xaml and search for a Border control that contains the FlowDirection attribute – a new property which belongs to the FrameworkElement class. The attribute is bound to a property called IsRTL.
7. Open MainPage.xaml.cs and examine the property code:

```
C#
#region Properties
public bool IsRTL
{
    get { return (bool)GetValue(IsRTLProperty); }
    set { SetValue(IsRTLProperty, value); }
}

// Using a DependencyProperty as the backing store for IsRTL. This enables
// animation, styling, binding, etc...
public static readonly DependencyProperty IsRTLProperty =
    DependencyProperty.Register("IsRTL", typeof(bool),
        typeof(MainPage),
        null);
```

- Next, we'll add a ToggleButton that will be responsible for changing the property's value. Open MainPage.xaml and locate the comment <!-- TODO - Display, Highlight, Xaml Buttons-->. Add the following code below the comment:

XAML

```
<ToggleButton x:Name="btnRTL" Checked="btnRTL_Checked"
    Unchecked="btnRTL_Checked" Grid.Column="4" Margin="15, -22, 0, 24"
    HorizontalAlignment="Center" Grid.Row="1">
    <ToolTipService.ToolTip>
        <ToolTip FontSize="16" Content="Text Direction"/>
    </ToolTipService.ToolTip>
    <Image Source="Images/ltr.png" Width="30"/>
</ToggleButton>
```

- Right-click on the Checked event handler name and select **Navigate to Event Handler** to go to it in MainPage.xaml.cs. Add the following code into the event handler to handle showing different images based upon the value of the IsRTL property:

C#

```
IsRTL = !IsRTL;
if (IsRTL)
    btnRTL.Content = CreateImageFromUri(
        new Uri("/RichNotepad;component/images/rtl.png",
            UriKind.RelativeOrAbsolute), 30, 32);
else
    btnRTL.Content = CreateImageFromUri(
        new Uri("/RichNotepad;component/images/ltr.png",
            UriKind.RelativeOrAbsolute), 30, 32);
ReturnFocus();
```

- Compile and run the application by pressing F5. Enter some text and then press the LTR toggle button – The entire RichTextBox control's content is now right aligned – buttons, controls, and grids are all aligned to the right. Now our application is BiDi aware and takes in account the direction of the text and changes to the FlowDirection property.

Task 2 –Culture based UI with Resources

In this task, we'll use 3 cultures – en-US (English), ar-SA (Arabic) and he-IL (Hebrew) and will dynamically change the UI culture of our application according to the user's language setting.

- The first thing we'll do is decide whether the flow direction of the application should be LTR (left-to-right) or RTL (right-to-left), according to the selected culture. Open MainPage.xaml.cs and observe the following in the constructor:

C#

```
IsRTL =
Thread.CurrentThread.CurrentUICulture.Parent.Name.ToLower() == "he" ||
```

```
Thread.CurrentThread.CurrentUICulture.Parent.Name.ToLower() == "ar";
```

2. In the Solution Explorer, expand the content of the RichTextBox.Web project, right-click the SelectCulture.htm file and select Set As Start Page.
3. Open SelectCulture.htm file in the code editor window. You'll see that the html page contains a ComboBox and a Button – pressing the button makes the browser navigate to the Silverlight page with the selected culture as part of the URL.
4. Open RichTextBoxTestPage.aspx and look at the culture parameter passed to the Silverlight object (bottom of the page). This is how we pass the selected culture from the URL to the Silverlight Control.
5. Compile and run the application by pressing F5. You'll see the following page



Figure 5

Language Selection Page in Browser

6. Select English and press the Open Editor Button. The application will open with LTR mode enabled.
 7. Run the application again, this time select either Hebrew or Arabic. Notice how the application changes its flow to RTL.
-

Conclusion

In this lab we learned about new features Silverlight 4 offers in the area of text editing, bi-directional text and layout and culture support. We've used the new RichTextBox control and seen how it allows the user to input rich text and embed hyperlinks and images into a document and to save or print the rich text.

We've also seen the Clipboard capabilities of Silverlight 4 and how we can copy and paste text in and out of Silverlight. In addition, we've used Silverlight 4's new bi-directional (BiDi) feature that allows us to change the flow direction of our Silverlight applications to either left-to-right (LTR) or right-to-left (RTL).