Applied WPF: Text and Documents

Objectives

After completing this lab, you should understand how to use the following WPF features:

- TextBlock
- Inline text elements
- FlowDocument
- Block text elements
- Document reader controls

Overview

In this lab, you will explore the TextBlock's formatting capabilities. Then you will look FlowDocument and the associated reader controls. Finally, you will modify the lab manual viewer to make use of FlowDocuments.

Part 1 – TextBlock

While TextBlock is the simplest framework element for rendering text, it is still remarkably powerful. In this first section, you will try out a variety of the features it offers.

1. Run XamlPad. Save any content you wish to keep, and then start with a blank Grid:

```
<Grid
   xmlns="http://
schemas.microsoft.com/
winfx/2006/xaml/presentation"
   xmlns:x="http://
schemas.microsoft.com/
winfx/2006/xaml">


</Grid>
```

2. Add a TextBlock element. Set its FontSize to 30. Add some text as the content of the TextBlock element. Type enough that it is too wide for the window. Notice that the line is truncated.

3. Add a TextWrapping="Wrap" attribute. The text now wraps across multiple lines in order to fit.

4. Make sure your text contains some long words. (Antidisestablishmentarianism and floccinaucinihilipification are popular choices.) Make the window narrower, and these words will start to stand out, as they defeat the attempts of the TextBlock to format the text. Set the IsHyphenationEnabled property to True, to alleviate this problem.

5. Add lots more text, so that it wraps over several lines. Notice that the right hand edge is 'jagged' – the line ends are not aligned. Add a TextAlignment="Justify" attribute to the TextBlock. WPF will now pad out the spacing between words in order to make both the lines of equal length. Try the other alignment options: Center, Right, and (the default) Left.

6. Pick some subset of the words you've typed in, and wrap them with an <Italic> element. The relevant text should change to an italic style. Try the other inline elements Bold and Underline. Try making them overlap – you can nest these elements, but you will not be able to use this kind of style: <Bold>This <Italic>is </Bold>not</Italic> legal. XAML must always be well-formed XML.

7. These elements all derive from Span, and are just convenient classes – you can achieve the same things using Span. Replace each of these elements with an equivalently configured Span:
    a. For Italic, set the Span's FontStyle to Italic.
    b. For Bold, set the Span's FontWeight to Bold.
    c. For Underline, set the TextDecorations to Underline. (TextDecorations also supports Overline, StrikeThrough, and Baseline. You can combine these by providing a comma separated list of decorations.)

8. Introduce some line breaks wi the <LineBreak /> element

9. Add the following to see the effect of the BaselineAlignme property.

```
E = mc<Span
BaselineAlignment="Superscr
t" FontSize="18">2</Span>
```

10. Set the Background property o a Span to a brush, such as "Yellow".

11. Set the FontFamily of a Span t a font, such as "Palatino Linotype".

Part 2 – FlowDocument an Block Elements

Powerful as it is, the TextBlock is mainly aimed at presenting fairly sma volumes of text. It can just about stret to multiple paragraphs with its LineBreak element, but it is not well suited to long documents. It does not provide features such as tables, lists, figures. It does not make it straightforward to identify a single paragraph for formatting purposes. It cannot easily be split into pages for printing or paginated on-screen display. It cannot arrange text to flow around multiple features, being limite to a single column of layout.

FlowDocument is able to do solve all these problems. However, unlike TextBlock, it cannot be used in isolati – a FlowDocument is simply a description of a document. Viewer controls (or the printing API) must be used in order to render the document.

In this section, you will become famili

with the block-level features FlowDocument offers, building on the inline elements available in TextBlock. (FlowDocument can use any of the inline elements you've already seen.)

1. In XamlPad, convert the previous work into a FlowDocument-based example:
    a. Wrap the TextBlock in a FlowDocumentScrollViewer element.
    b. Change the TextBlock to a FlowDocument.
    c. Remove the TextWrapping attribute. (FlowDocuments always wrap.)
    d. Wrap the content of the FlowDocument in a Paragraph element

2. XamlPad should be happy enough to display the document at this point. However, it's not quite entering into the spirit of a flow document – new paragraphs are being indicated by LineBreak elements. This works, but the usual way to indicate a paragraph in a FlowDocument is to wrap it in a Paragraph element, rather than to put LineBreak elements in between each paragraph. So any place you have a LineBreak, replace it by closing off the current Paragraph tag and then opening a new Paragraph tag.

3. Next, add a List element. This should come between some paragraphs, or after the final paragraph. Within the List, add a ListItem element. Inside this, add a Paragraph. Inside the paragraph, add some text – as you'd expect, you can use and of the elements you've been using in paragraphs outside of a list. Add some more ListItems, each with paragraphs of their own.

4. By default, List shows a bulleted list. However, by setting the MarkerStyle property, you can change how each item is marked. The available options are Box, Circle, Decimal, Disc, LowerLatin, LowerRoman, Square, UpperLatin, UpperRoman, and None. Try a few of these.

5. Add the following markup to put a table into the text:

```
<Table Background="#eef"
BorderBrush="Black"
BorderThickness="1">
  <TableRowGroup>
    <TableRow
FontWeight="Bold">

<TableCell><Paragraph>Foo</
Paragraph></TableCell>

<TableCell><Paragraph>Bar</
Paragraph></TableCell>

<TableCell><Paragraph>Quux</
Paragraph></TableCell>
    </TableRow>
    <TableRow>

<TableCell><Paragraph>One</
Paragraph></TableCell>

<TableCell><Paragraph>Two</
Paragraph></TableCell>

<TableCell><Paragraph>Three</
Paragraph></TableCell>
    </TableRow>
  </TableRowGroup>
</Table>
```

6. Experiment with the table layout – try adding extra cells to a row, and see how the column count adapts dynamically. Add extra TableRowGroup elements – these allow you to apply a consistent set of formatting to a set of rows. Try using the TableCell's RowSpan and ColumnSpan properties – these work in a similar way to the equivalent Grid properties.

7. The Floater element allows you to anchor some content to a particular point in the document, but have the content itself rendered outside of the main flow. The main document will then flow around the floater. Try adding this *inside* one of your paragraphs:

```
<Floater BorderBrush="Black"
BorderThickness="1"
Width="200">
  <Table>
    <TableRowGroup>
      <TableRow>
        <TableCell
TextAlignment="Center"
Background="#EEE">
          <Paragraph
Margin="4">Hello, world!</
Paragraph>
        </TableCell>
      </TableRow>
      <TableRow>
        <TableCell
TextAlignment="Center">
          <Paragraph
FontStyle="Italic">Figure 1
A software
              development
classic</Paragraph>
        </TableCell>
      </TableRow>
    </TableRowGroup>
  </Table>
</Floater>
```

The Floater is technically an Inline element because it is placed within a paragraph, at t point at which you would like the floater to be anchored. While the floater appears outside of the main flow, its position is influenced by when it appears. If you have a particularly long paragraph, tr changing the position at which you insert the floater, to see ho this changes its position on screen.

Despite being an Inline, Floate only makes sense in the contex of a FlowDocument, so you cannot usefully use it in a TextBlock.

8. Finally, change the Floater to Figure. This allows you to change the placement of the figure – rather than having to l next to its anchored paragraph you can indicate that it should appear on a particular position on the page containing the anchor. This is controlled with the VerticalAnchor and HorizontalAnchor properties. However, these only function fully with paginated display. F this we need to use a different document reader control.

Part 3 – Document Reader Controls

In the previous section, you used the FlowDocumentScrollViewer control. This provides the simplest display – a linear scrolling top-to-bottom view of the document. However, it doesn't necessarily make the most effective use of space, particularly if a lot of width is available. Moreover, it cannot use page-oriented document features such as figure placement.

In this step you will look at the alternative ways of presenting a FlowDocument.

1. Change the FlowDocumentScrollViewer to a FlowDocumentPageViewer. This will change to a paginating display, and it will break the text into columns if space is available.

2. Experiment with the anchoring of your figure now that you have a paginated columnar view. Set both VerticalAnchor and HorizontalAnchor to PageCenter – the figure should appear in the middle of the page, with columns flowing around it.

3. The reader control offers zooming and paging controls. Try changing the zoom to see how it reformats the document to suit. Also try resizing the window to see how it adapts to make use of the space.

4. Change the reader to a FlowDocumentReader. This lets the user select which of the viewing modes to use – it offers buttons to select between the scrolling and the paged mode. It also offers a two-page view, which feels similar to how the content would look if printed out double sided and read as a booklet.

Part 4 – Using FlowDocuments in LabManualViewer

The LabManualViewer currently uses TextBlock for all of its pages. It would be better to use FlowDocument, as this allows for much more flexible content. So you will now modify the application to use FlowDocuments.

1. Open your LabManualViewer project. (Or use the one provided as a starting point in this lab's Before directory.)

2. Remove all the pages except for the Quiz.xaml from the Pages folder. Leave Quiz.xaml and Quiz.xaml.cs in place.

3. Add in the replacement versions in the Pages directory for this lab.

4. Open one of these files up – see how these are now FlowDocuments instead of Pages.

5. Run the application. Note that we're very nearly done – it turns out that the Frame class is as happy to display FlowDocuments as it is Pages. (It uses the FlowDocumentReader, allowing the user to choose their reading style.)

6. The main problem is that we now have two zoom controls – the one we added earlier, and the one built into the document reader control. Remove the Slider in Window1 that provides zoom facilities. And remove both the corresponding event handler code, and the code that sets the zoomSlider.DataContext in the codebehind.

7. Now, your zoom setting handling code won't be working. Add the following helper in Window1.xaml.cs to locate the FlowDocumentReader:

```
FlowDocumentReader FindReader
(DependencyObject elem)
{
    int c =
VisualTreeHelper.GetChildre
ount(elem);
    for (int i = 0; i < c;
+i)
    {
        DependencyObject
child =
VisualTreeHelper.GetChild
(elem, i);
        FlowDocumentReader
= child as
FlowDocumentReader;
        if (r == null)
        {
            r = FindReader
(child);
        }
        if (r != null)
        {
            return r;
        }
    }
    return null;
}
```

8. Use this in the 100% zoom menu item click handler:

```
void zoom100Menu_Click(obje
sender, RoutedEventArgs e)
{
    FlowDocumentReader rdr
FindReader(contentArea);
    if (rdr != null)
    {
        rdr.Zoom = 1;
    }
}
```

Conclusion

In this lab, you used the TextBlock's formatting capabilities. Then you looked at how FlowDocument builds these, adding paragraphs, lists, and tables. Finally, you modified the lab manual viewer to make use of FlowDocuments.