Flow Document Overview

.NET Framework 4

3 out of 4 rated this helpful   Rate this topic

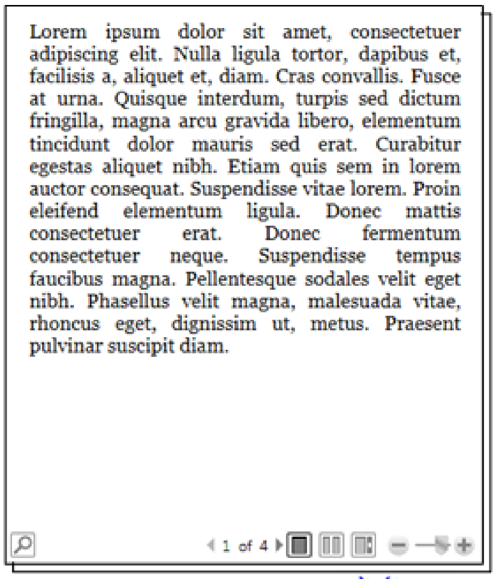
Flow documents are designed to optimize viewing and readability. Rather than being set to one predefined layout, flow documents dynamically adjust and reflow their content based on run-time variables such as window size, device resolution, and optional user preferences. In addition, flow documents offer advanced document features, such as pagination and columns. This topic provides an overview of flow documents and how to create them.

This topic contains the following sections.

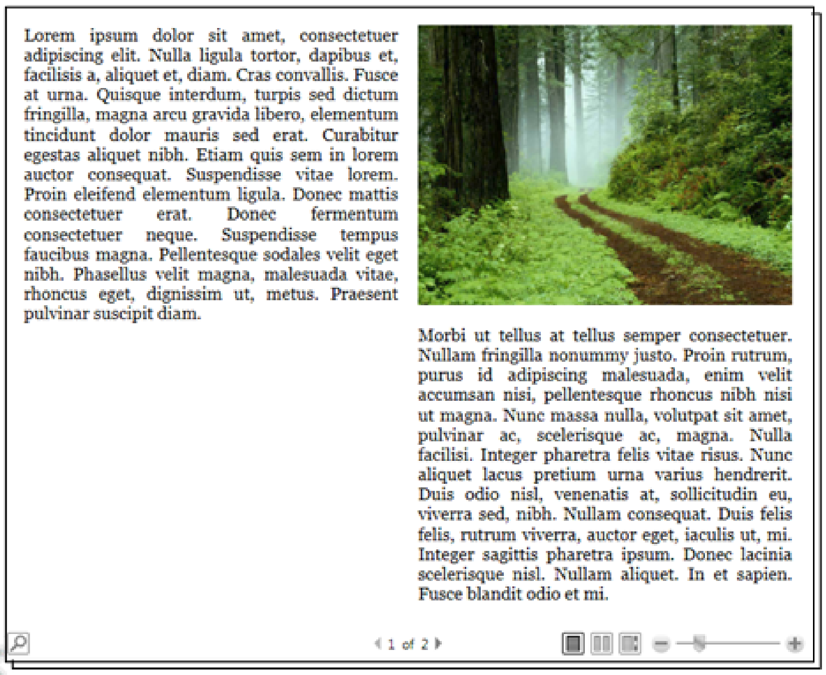
What is a Flow Document

A flow document is designed to "reflow content" depending on window size, device resolution, and other environment variables. In addition, flow documents have a number of built in features including search, viewing modes that optimize readability, and the ability to change the size and appearance of fonts. Flow Documents are best utilized when ease of reading is the primary document consumption scenario. In contrast, Fixed Documents are designed to have a static presentation. Fixed Documents are useful when fidelity of the source content is essential. See Documents in WPF for more information on different types of documents.

The following illustration shows a sample flow document viewed in several windows of different sizes. As the display area changes, the content reflows to make the best use of the available space.

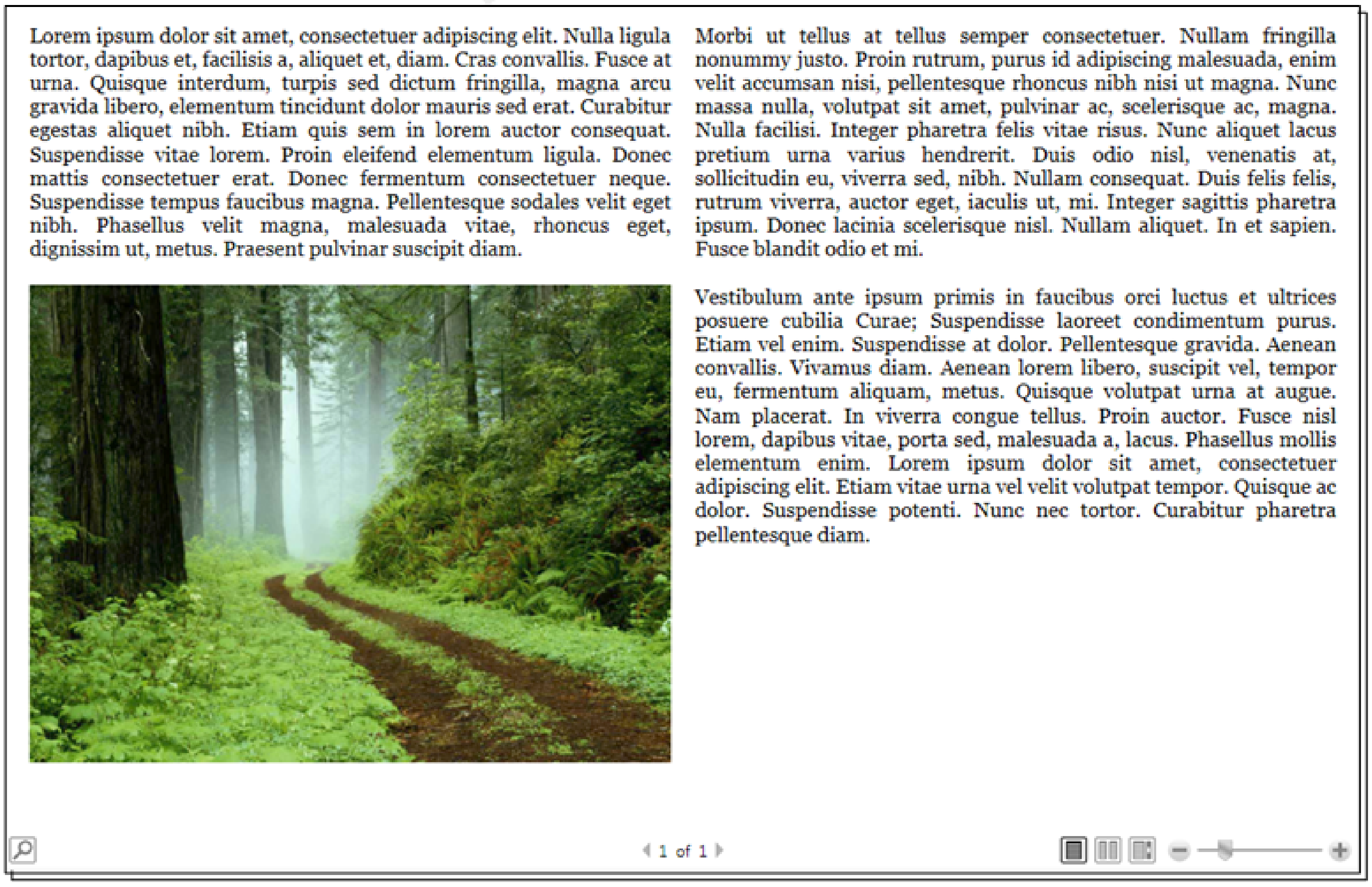


Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetuer erat. Donec fermentum consectetuer neque. Suspendisse tempus faucibus magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetuer erat. Donec fermentum consectetuer neque. Suspendisse tempus faucibus magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.

Morbi ut tellus at tellus semper consectetuer. Nullam fringilla nonummy justo. Proin rutrum, purus id adipiscing malesuada, enim velit accumsan nisi, pellentesque rhoncus nibh nisi ut magna. Nunc massa nulla, volutpat sit amet, pulvinar ac, scelerisque ac, magna. Nulla facilisi. Integer pharetra felis vitae risus. Nunc aliquet lacus pretium urna varius hendrerit. Duis odio nisl, venenatis at, sollicitudin eu, viverra sed, nibh. Nullam consequat. Duis felis felis, rutrum viverra, auctor eget, iaculis ut, mi. Integer sagittis pharetra ipsum. Donec lacinia scelerisque nisl. Nullam aliquet. In et sapien. Fusce blandit odio et mi.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla ligula tortor, dapibus et, facilisis a, aliquet et, diam. Cras convallis. Fusce at urna. Quisque interdum, turpis sed dictum fringilla, magna arcu gravida libero, elementum tincidunt dolor mauris sed erat. Curabitur egestas aliquet nibh. Etiam quis sem in lorem auctor consequat. Suspendisse vitae lorem. Proin eleifend elementum ligula. Donec mattis consectetuer erat. Donec fermentum consectetuer neque. Suspendisse tempus faucibus magna. Pellentesque sodales velit eget nibh. Phasellus velit magna, malesuada vitae, rhoncus eget, dignissim ut, metus. Praesent pulvinar suscipit diam.

Morbi ut tellus at tellus semper consectetuer. Nullam fringilla nonummy justo. Proin rutrum, purus id adipiscing malesuada, enim velit accumsan nisi, pellentesque rhoncus nibh nisi ut magna. Nunc massa nulla, volutpat sit amet, pulvinar ac, scelerisque ac, magna. Nulla facilisi. Integer pharetra felis vitae risus. Nunc aliquet lacus pretium urna varius hendrerit. Duis odio nisl, venenatis at, sollicitudin eu, viverra sed, nibh. Nullam consequat. Duis felis felis, rutrum viverra, auctor eget, iaculis ut, mi. Integer sagittis pharetra ipsum. Donec lacinia scelerisque nisl. Nullam aliquet. In et sapien. Fusce blandit odio et mi.

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Suspendisse laoreet condimentum purus. Etiam vel enim. Suspendisse at dolor. Pellentesque gravida. Aenean convallis. Vivamus diam. Aenean lorem libero, suscipit vel, tempor eu, fermentum aliquam, metus. Quisque volutpat urna at augue. Nam placerat. In viverra congue tellus. Proin auctor. Fusce nisl lorem, dapibus vitae, porta sed, malesuada a, lacus. Phasellus mollis elementum enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam vitae urna vel velit volutpat tempor. Quisque ac dolor. Suspendisse potenti. Nunc nec tortor. Curabitur pharetra pellentesque diam.

As seen in the image above, flow content can include many components including paragraphs, lists, images, and more. These components correspond to elements in markup and objects in procedural code. We will go over these classes in detail later in the Flow Related Classes section of this overview. For now, here is a simple code example that creates a flow document consisting of a paragraph with some bold text and a list.

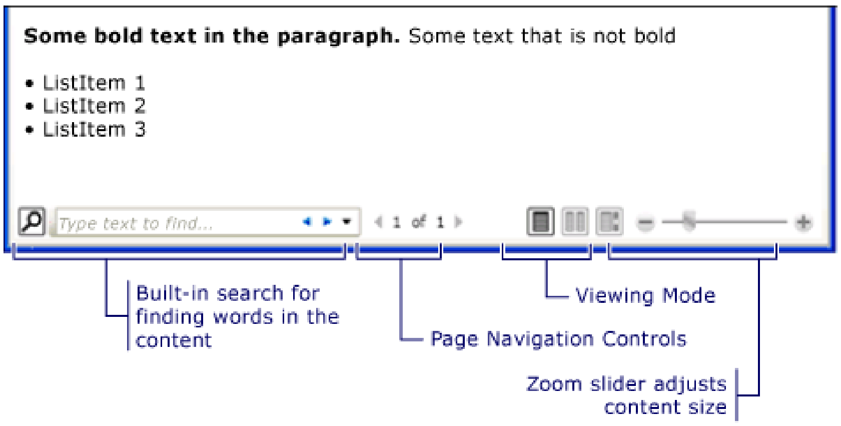
```
<!-- This simple flow document includes a paragraph with some
     bold text in it and a list. -->
<FlowDocumentReader xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <FlowDocument>
    <Paragraph>
      <Bold>Some bold text in the paragraph.</Bold>
      Some text that is not bold.
```

```xml
    </Paragraph>

    <List>
      <ListItem>
        <Paragraph>ListItem 1</Paragraph>
      </ListItem>
      <ListItem>
        <Paragraph>ListItem 2</Paragraph>
      </ListItem>
      <ListItem>
        <Paragraph>ListItem 3</Paragraph>
      </ListItem>
    </List>

  </FlowDocument>
</FlowDocumentReader>
```

C#

```csharp
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class SimpleFlowExample : Page
    {
        public SimpleFlowExample()
        {

            Paragraph myParagraph = new Paragraph();

            // Add some Bold text to the paragraph
            myParagraph.Inlines.Add(new Bold(new Run("Some bold text in the paragraph.")));

            // Add some plain text to the paragraph
            myParagraph.Inlines.Add(new Run(" Some text that is not bold."));

            // Create a List and populate with three list items.
            List myList = new List();

            // First create paragraphs to go into the list item.
            Paragraph paragraphListItem1 = new Paragraph(new Run("ListItem 1"));
            Paragraph paragraphListItem2 = new Paragraph(new Run("ListItem 2"));
            Paragraph paragraphListItem3 = new Paragraph(new Run("ListItem 3"));

            // Add ListItems with paragraphs in them.
            myList.ListItems.Add(new ListItem(paragraphListItem1));
            myList.ListItems.Add(new ListItem(paragraphListItem2));
            myList.ListItems.Add(new ListItem(paragraphListItem3));

            // Create a FlowDocument with the paragraph and list.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);
            myFlowDocument.Blocks.Add(myList);

            // Add the FlowDocument to a FlowDocumentReader Control
            FlowDocumentReader myFlowDocumentReader = new FlowDocumentReader();
            myFlowDocumentReader.Document = myFlowDocument;

            this.Content = myFlowDocumentReader;
        }
    }
}
```

The illustration below shows what this code snippet looks like.


Some bold text in the paragraph. Some text that is not bold

- ListItem 1
- ListItem 2
- ListItem 3

Type text to find...
1 of 1
Built-in search for finding words in the content
Page Navigation Controls
Viewing Mode
Zoom slider adjusts content size

In this example, the FlowDocumentReader control is used to host the flow content. See Flow Document Types for more information on flow content hosting controls. Paragraph, List, ListItem, and Bold elements are used to control content formatting, based on their order in markup. For example, the Bold element spans across only part of the text in the paragraph; as a result, only that part of the text is bold. If you have used HTML, this will be familiar to you.

As highlighted in the illustration above, there are several features built into Flow Documents:

- Search: Allows the user to perform a full text search of an entire document.
- Viewing Mode: The user can select their preferred viewing mode including a single-page (page-at-a-time) viewing mode, a two-page-at-a-time (book reading format) viewing mode, and a continuous scrolling (bottomless) viewing mode. For more information about these viewing modes, see FlowDocumentReaderViewingMode.
- Page Navigation Controls: If the viewing mode of the document uses pages, the page navigation controls include a button to jump to the next page (the down arrow) or previous page (the up arrow), as well as indicators for the current page number and total number of pages. Flipping through pages can also be accomplished using the keyboard arrow keys.
- Zoom: The zoom controls enable the user to increase or decrease the zoom level by clicking the plus or minus buttons, respectively. The zoom controls also include a slider for adjusting the zoom level. For more information, see Zoom.

These features can be modified based upon the control used to host the flow content. The next section describes the different controls.

Flow Document Types

Display of flow document content and how it appears is dependent upon what object is used to host the flow content. There are four controls that support viewing of flow content: FlowDocumentReader, FlowDocumentPageViewer, RichTextBox, and FlowDocumentScrollViewer. These controls are briefly described below.

Note: FlowDocument is required to directly host flow content, so all of these viewing controls consume a FlowDocument to enable flow content hosting.

FlowDocumentReader

FlowDocumentReader includes features that enable the user to dynamically choose between various viewing modes, including a single-page (page-at-a-time) viewing mode, a two-page-at-a-time (book reading format) viewing mode, and a continuous scrolling (bottomless) viewing mode. For more information about these viewing modes, see FlowDocumentReaderViewingMode. If you do not need the ability to dynamically switch between different viewing modes, FlowDocumentPageViewer and FlowDocumentScrollViewer provide lighter-weight flow content viewers that are fixed in a particular viewing mode.

FlowDocumentPageViewer and FlowDocumentScrollViewer

FlowDocumentPageViewer shows content in page-at-a-time viewing mode, while FlowDocumentScrollViewer shows content in continuous scrolling mode. Both FlowDocumentPageViewer and FlowDocumentScrollViewer are fixed to a particular viewing mode. Compare to FlowDocumentReader, which includes features that enable the user to dynamically choose between various viewing modes (as provided by the FlowDocumentReaderViewingMode enumeration), at the cost of being more resource intensive than FlowDocumentPageViewer or FlowDocumentScrollViewer.

By default, a vertical scrollbar is always shown, and a horizontal scrollbar becomes visible if needed. The default UI for FlowDocumentScrollViewer does not include a toolbar; however, the IsToolBarVisible property can be used to enable a built-in toolbar.

RichTextBox

You use a RichTextBox when you want to allow the user to edit flow content. For example, if you wanted to create an editor that allowed a user to manipulate things like tables, italic and bold formatting, etc, you would use a RichTextBox. See RichTextBox Overview for more information.

Note: Flow content inside a RichTextBox does not behave exactly like flow content contained in other controls. For example, there are no columns in a RichTextBox and hence no automatic resizing behavior. Also, the typically built in features of flow content like search, viewing mode, page navigation, and zoom are not available within a RichTextBox.

Creating Flow Content

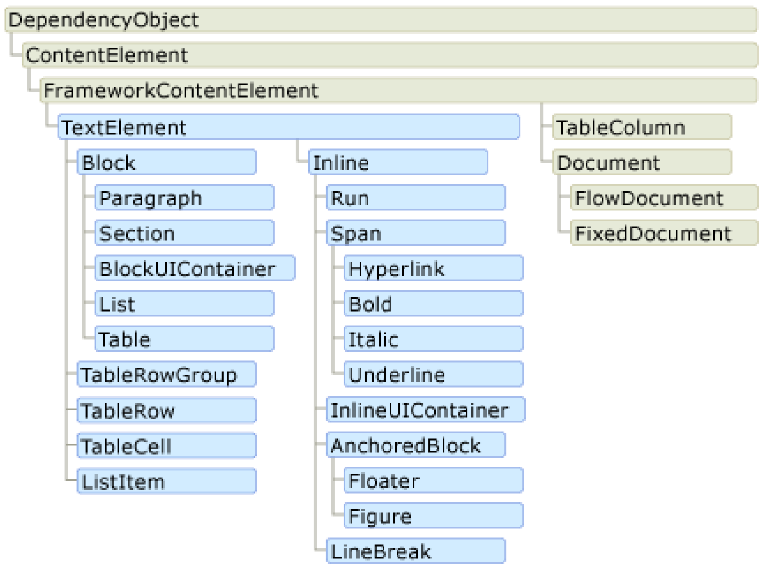
Flow content can be complex, consisting of various elements including text, images, tables, and even UIElement derived classes like controls. To understand how to create complex flow content, the following points are critical:

- **Flow-related Classes**: Each class used in flow content has a specific purpose. In addition, the hierarchical relation between flow classes helps you understand how they are used. For example, classes derived from the Block class are used to contain other objects while classes derived from Inline contain objects that are displayed.
- **Content Schema**: A flow document can require a substantial number of nested elements. The content schema specifies possible parent/child relationships between elements.

The following sections will go over each of these areas in more detail.

Flow Related Classes

The diagram below shows the objects most typically used with flow content:



For the purposes of flow content, there are two important categories:

1. **Block-derived classes**: Also called "Block content elements" or just "Block Elements". Elements that inherit from Block can be used to group elements under a common parent or to apply common attributes to a group.
2. **Inline-derived classes**: Also called "Inline content elements" or just "Inline Elements". Elements that inherit from Inline are either contained within a Block Element or another Inline Element. Inline Elements are often used as the direct container of content that is rendered to the screen. For example, a Paragraph (Block Element) can contain a Run (Inline Element) but the Run actually contains the text that is rendered on the screen.

Each class in these two categories is briefly described below.

Block-derived Classes

Paragraph

Paragraph is typically used to group content into a paragraph. The simplest and most common use of Paragraph is to create a paragraph of text.

```xml
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Paragraph>
    Some paragraph text.
  </Paragraph>
</FlowDocument>
```

C#

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class ParagraphExample : Page
    {
        public ParagraphExample()
        {

            // Create paragraph with some text.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(new Run("Some paragraph text."));

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}
```

However, you can also contain other inline-derived elements as you will see below.

Section

Section is used only to contain other Block-derived elements. It does not apply any default formatting to the elements it contains. However, any property values set on a Section applies to its child elements. A section also enables you to programmatically iterate through its child collection. Section is used in a similar manner to the <DIV> tag in HTML.

In the example below, three paragraphs are defined under one Section. The section has a Background property value of Red, therefore the background color of the paragraphs is also red.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <!-- By default, Section applies no formatting to elements contained
       within it. However, in this example, the section has a Background
       property value of "Red", therefore, the three paragraphs (the block)
       inside the section also have a red background. -->
  <Section Background="Red">
    <Paragraph>
      Paragraph 1
    </Paragraph>
    <Paragraph>
      Paragraph 2
    </Paragraph>
    <Paragraph>
      Paragraph 3
    </Paragraph>
  </Section>
</FlowDocument>
```

C#

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class SectionExample : Page
    {
        public SectionExample()
        {

            // Create three paragraphs
            Paragraph myParagraph1 = new Paragraph(new Run("Paragraph 1"));
            Paragraph myParagraph2 = new Paragraph(new Run("Paragraph 2"));
            Paragraph myParagraph3 = new Paragraph(new Run("Paragraph 3"));

            // Create a Section and add the three paragraphs to it.
            Section mySection = new Section();
            mySection.Background = Brushes.Red;

            mySection.Blocks.Add(myParagraph1);
            mySection.Blocks.Add(myParagraph2);
            mySection.Blocks.Add(myParagraph3);

            // Create a FlowDocument and add the section to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(mySection);

            this.Content = myFlowDocument;
        }
    }
}
```

BlockUIContainer

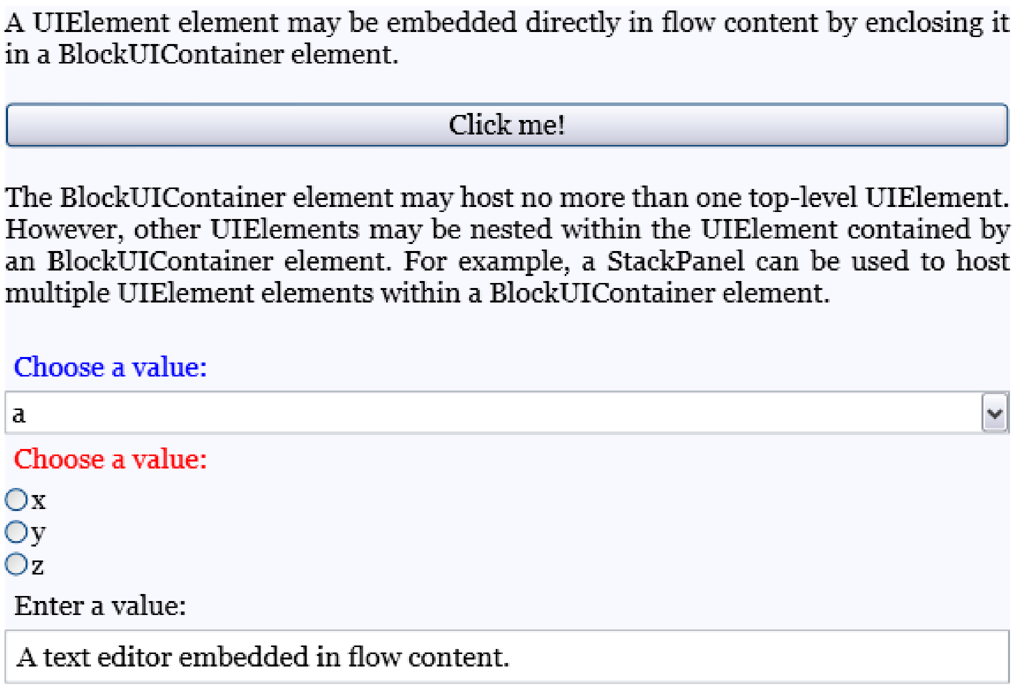
BlockUIContainer enables UIElement elements (i.e. a Button) to be embedded in block-derived flow content. InlineUIContainer (see below) is used to embed UIElement elements in inline-derived flow content. BlockUIContainer and InlineUIContainer are important because there is no other way to use a UIElement in flow content unless it is contained within one of these two elements.

The following example shows how to use the BlockUIContainer element to host UIElement objects within flow content.

```
<FlowDocument ColumnWidth="400">
  <Section Background="GhostWhite">
    <Paragraph>
      A UIElement element may be embedded directly in flow content
      by enclosing it in a BlockUIContainer element.
    </Paragraph>
    <BlockUIContainer>
      <Button>Click me!</Button>
    </BlockUIContainer>
    <Paragraph>
      The BlockUIContainer element may host no more than one top-level
      UIElement.  However, other UIElements may be nested within the
      UIElement contained by an BlockUIContainer element.  For example,
      a StackPanel can be used to host multiple UIElement elements within
      a BlockUIContainer element.
    </Paragraph>
    <BlockUIContainer>
      <StackPanel>
        <Label Foreground="Blue">Choose a value:</Label>
        <ComboBox>
          <ComboBoxItem IsSelected="True">a</ComboBoxItem>
          <ComboBoxItem>b</ComboBoxItem>
          <ComboBoxItem>c</ComboBoxItem>
        </ComboBox>
        <Label Foreground ="Red">Choose a value:</Label>
        <StackPanel>
          <RadioButton>x</RadioButton>
          <RadioButton>y</RadioButton>
          <RadioButton>z</RadioButton>
        </StackPanel>
        <Label>Enter a value:</Label>
        <TextBox>
          A text editor embedded in flow content.
        </TextBox>
      </StackPanel>
    </BlockUIContainer>
  </Section>
```

```
</FlowDocument>
```

The following figure shows how this example renders.



A UIElement element may be embedded directly in flow content by enclosing it in a BlockUIContainer element.

The BlockUIContainer element may host no more than one top-level UIElement. However, other UIElements may be nested within the UIElement contained by an BlockUIContainer element. For example, a StackPanel can be used to host multiple UIElement elements within a BlockUIContainer element.

List

List is used to create a bulleted or numeric list. Set the MarkerStyle property to a TextMarkerStyle enumeration value to determine the style of the list. The example below shows how to create a simple list.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <List>
    <ListItem>
      <Paragraph>
        List Item 1
      </Paragraph>
    </ListItem>
    <ListItem>
      <Paragraph>
        List Item 2
      </Paragraph>
    </ListItem>
    <ListItem>
      <Paragraph>
        List Item 3
      </Paragraph>
    </ListItem>
  </List>
</FlowDocument>
```

C#

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class ListExample : Page
    {
        public ListExample()
        {
            // Create three paragraphs
            Paragraph myParagraph1 = new Paragraph(new Run("List Item 1"));
            Paragraph myParagraph2 = new Paragraph(new Run("List Item 2"));
            Paragraph myParagraph3 = new Paragraph(new Run("List Item 3"));

            // Create the ListItem elements for the List and add the
            // paragraphs to them.
            ListItem myListItem1 = new ListItem();
            myListItem1.Blocks.Add(myParagraph1);
            ListItem myListItem2 = new ListItem();
            myListItem2.Blocks.Add(myParagraph2);
            ListItem myListItem3 = new ListItem();
            myListItem3.Blocks.Add(myParagraph3);

            // Create a List and add the three ListItems to it.
            List myList = new List();

            myList.ListItems.Add(myListItem1);
            myList.ListItems.Add(myListItem2);
            myList.ListItems.Add(myListItem3);

            // Create a FlowDocument and add the section to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myList);

            this.Content = myFlowDocument;
        }
    }
}
```

Note: List is the only flow element that uses the ListItemCollection to manage child elements.

Table

Table is used to create a table. Table is similar to the Grid element but it has more capabilities and, therefore, requires greater resource overhead. Because Grid is a UIElement, it cannot be used in flow content unless it is contained in a BlockUIContainer or InlineUIContainer. For more information on Table, see Table Overview.

Inline-derived Classes

Run

Run is used to contain unformatted text. You might expect Run objects to be used extensively in flow content. However, in markup, Run elements are not required to be used explicitly. Run is required to be used when creating or manipulating flow documents by using code. For example, in the markup below, the first Paragraph specifies the Run element explicitly while the second does not. Both paragraphs generate identical output.

```
<Paragraph>
  <Run>Paragraph that explicitly uses the Run element.</Run>
</Paragraph>

<Paragraph>
  This Paragraph omits the the Run element in markup. It renders
  the same as a Paragraph with Run used explicitly.
</Paragraph>
```

Note: Starting in the .NET Framework 4, the Text property of the Run object is a dependency property. You can bind the Text property to a data source, such as a TextBlock. The Text property fully supports one-way binding. The Text property also supports two-way binding, except for RichTextBox. For an example, see Run.Text.

Span

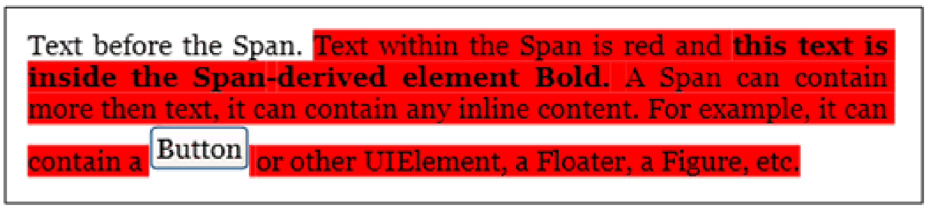
Span groups other inline content elements together. No inherent rendering is applied to content within a Span element. However, elements that inherit from Span including Hyperlink, Bold, Italic and Underline do apply formatting to text.

Below is an example of a Span being used to contain inline content including text, a Bold element, and a Button.

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

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

  <Paragraph>
    Text before the Span. <Span Background="Red">Text within the Span is
    red and <Bold>this text is inside the Span-derived element Bold.</Bold>
    A Span can contain more then text, it can contain any inline content. For
    example, it can contain a
    <InlineUIContainer>
      <Button>Button</Button>
    </InlineUIContainer>
    or other UIElement, a Floater, a Figure, etc.</Span>
  </Paragraph>

</FlowDocument>
```

The following screenshot shows how this example renders.



Text before the Span. Text within the Span is red and **this text is inside the Span-derived element Bold.** A Span can contain more then text, it can contain any inline content. For example, it can contain a Button or other UIElement, a Floater, a Figure, etc.

InlineUIContainer

InlineUIContainer enables UIElement elements (i.e. a control like Button) to be embedded in an Inline content element. This element is the inline equivalent to BlockUIContainer described above. Below is an example that uses InlineUIContainer to insert a Button inline in a Paragraph.

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

  <Paragraph>
    Text to precede the button...

    <!-- Set the BaselineAlignment property to "Bottom" 
         so that the Button aligns properly with the text. -->
    <InlineUIContainer BaselineAlignment="Bottom">
      <Button>Button</Button>
    </InlineUIContainer>
    Text to follow the button...
  </Paragraph>

</FlowDocument>
```

C#

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class InlineUIContainerExample : Page
    {
        public InlineUIContainerExample()
        {
            Run run1 = new Run(" Text to precede the button... ");
            Run run2 = new Run(" Text to follow the button... ");

            // Create a new button to be hosted in the paragraph.
            Button myButton = new Button();
            myButton.Content = "Click me!";

            // Create a new InlineUIContainer to contain the Button.
            InlineUIContainer myInlineUIContainer = new InlineUIContainer();
            // Set the BaselineAlignment property to "Bottom" so that the
            // Button aligns properly with the text.
            myInlineUIContainer.BaselineAlignment = BaselineAlignment.Bottom;

            // Asign the button as the UI container's child.
            myInlineUIContainer.Child = myButton;

            // Create the paragraph and add content to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(run1);
            myParagraph.Inlines.Add(myInlineUIContainer);
            myParagraph.Inlines.Add(run2);

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}
```

Note: InlineUIContainer does not need to be used explicitly in markup. If you omit it, an InlineUIContainer will be created anyway when the code is compiled.

Figure and Floater

Figure and Floater are used to embed content in Flow Documents with placement properties that can be customized independent of the primary content flow. Figure or Floater elements are often used to highlight or accentuate portions of content, to host supporting images or other content within the main content flow, or to inject loosely related content such as advertisements.

The following example shows how to embed a Figure into a paragraph of text.

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

  <Paragraph>
    <Figure
      Width="300" Height="100"
      Background="GhostWhite" HorizontalAnchor="PageLeft" >
      <Paragraph FontStyle="Italic" Background="Beige" Foreground="DarkGreen" >
        A Figure embeds content into flow content with placement properties
        that can be customized independently from the primary content flow
      </Paragraph>
    </Figure>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy
    nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi
    enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis
    nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
  </Paragraph>

</FlowDocument>
```

C#

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class FigureExample : Page
    {
        public FigureExample()
        {

            // Create strings to use as content.
            string strFigure = "A Figure embeds content into flow content with" +
                               " placement properties that can be customized" +
                               " independently from the primary content flow";
            string strOther = "Lorem ipsum dolor sit amet, consectetuer adipiscing" +
                              " elit, sed diam nonummy nibh euismod tincidunt ut laoreet" +
                              " dolore magna aliquam erat volutpat. Ut wisi enim ad" +
                              " minim veniam, quis nostrud exerci tation ullamcorper" +
                              " suscipit lobortis nisl ut aliquip ex ea commodo consequat." +
                              " Duis autem vel eum iriure.";

            // Create a Figure and assign content and layout properties to it.
            Figure myFigure = new Figure();
            myFigure.Width = new FigureLength(300);
            myFigure.Height = new FigureLength(100);
            myFigure.Background = Brushes.GhostWhite;
            myFigure.HorizontalAnchor = FigureHorizontalAnchor.PageLeft;
            Paragraph myFigureParagraph = new Paragraph(new Run(strFigure));
            myFigureParagraph.FontStyle = FontStyles.Italic;
            myFigureParagraph.Background = Brushes.Beige;
            myFigureParagraph.Foreground = Brushes.DarkGreen;
            myFigure.Blocks.Add(myFigureParagraph);

            // Create the paragraph and add content to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(myFigure);
            myParagraph.Inlines.Add(new Run(strOther));

            // Create a FlowDocument and add the paragraph to it.
            FlowDocument myFlowDocument = new FlowDocument();
            myFlowDocument.Blocks.Add(myParagraph);

            this.Content = myFlowDocument;
        }
    }
}
```

The following illustration shows how this example renders.

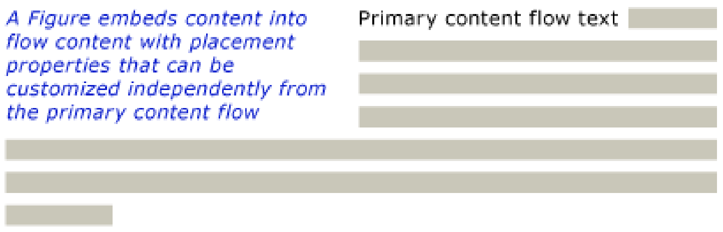

A Figure embeds content into flow content with placement properties that can be customized independently from the primary content flow
Primary content flow text

Figure and Floater differ in several ways and are used for different scenarios.

Figure:

- Can be positioned: You can set its horizontal and vertical anchors to dock it relative to the page, content, column or paragraph. You can also use its HorizontalOffset and VerticalOffset properties to specify arbitrary offsets.
- Is sizable to more than one column: You can set Figure height and width to multiples of page, content or column height or width. Note that in the case of page and content, multiples greater than 1 are not allowed. For example, you can set the width of a Figure to be "0.5 page" or "0.25 content" or "2 Column". You can also set height and width to absolute pixel values.
- Does not paginate: If the content inside a Figure does not fit inside the Figure, it will render whatever content does fit and the remaining content is lost

Floater:

- Cannot be positioned and will render wherever space can be made available for it. You cannot set the offset or anchor a Floater.
- Cannot be sized to more than one column: By default, Floater sizes at one column. It has a Width property that can be set to an absolute pixel value, but if this value is greater than one column width it is ignored and the floater is sized at one column. You can size it to less than one column by setting the correct pixel width, but sizing is not column-relative, so "0.5Column" is not a valid expression for Floater width. Floater has no height property and it's height cannot be set, it’s height depends on the content
- Floater paginates: If its content at its specified width extends to more than 1 column height, floater breaks and paginates to the next column, the next page, etc.

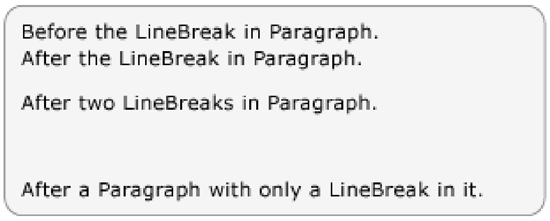
Figure is a good place to put standalone content where you want to control the size and positioning, and are confident that the content will fit in the specified size. Floater is a good place to put more free-flowing content that flows similar to the main page content, but is separated from it.

LineBreak

LineBreak causes a line break to occur in flow content. The following example demonstrates the use of LineBreak.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Paragraph>
    Before the LineBreak in Paragraph.
    <LineBreak />
    After the LineBreak in Paragraph.
    <LineBreak/><LineBreak/>
    After two LineBreaks in Paragraph.
  </Paragraph>

  <Paragraph>
    <LineBreak/>
  </Paragraph>

  <Paragraph>
    After a Paragraph with only a LineBreak in it.
  </Paragraph>
</FlowDocument>
```

The following screenshot shows how this example renders.


Before the LineBreak in Paragraph.
After the LineBreak in Paragraph.

After two LineBreaks in Paragraph.

After a Paragraph with only a LineBreak in it.

Flow Collection Elements

In many of the examples above, the BlockCollection and InlineCollection are used to construct flow content programmatically. For example, to add elements to a Paragraph, you can use the syntax:

…

```
myParagraph.Inlines.Add(new Run("Some text"));
```

…

This adds a Run to the InlineCollection of the Paragraph. This is the same as the implicit Run found inside a Paragraph in markup:

…

```
<Paragraph>

Some Text

</Paragraph>
```

…

As an example of using the BlockCollection, the following example creates a new Section and then uses the **Add** method to add a new Paragraph to the Section contents.

C#

```
Section secx = new Section();
secx.Blocks.Add(new Paragraph(new Run("A bit of text content...")));
```

In addition to adding items to a flow collection, you can remove items as well. The following example deletes the last Inline element in the Span.

C#

```
spanx.Inlines.Remove(spanx.Inlines.LastInline);
```

The following example clears all of the contents (Inline elements) from the Span.

C#

```
spanx.Inlines.Clear();
```

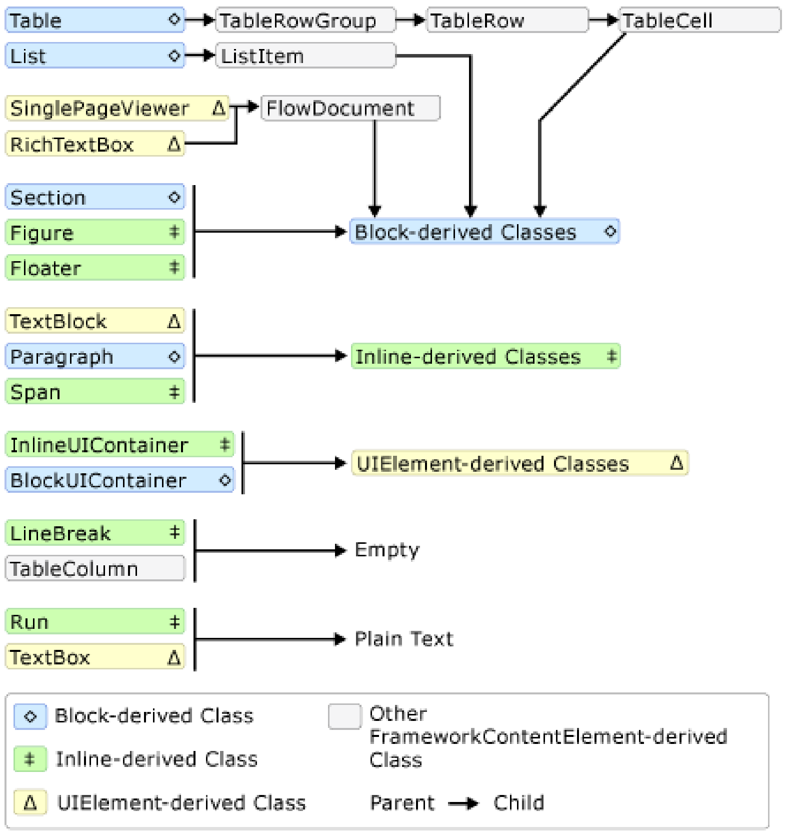
When working with flow content programmatically, you will likely make extensive use of these collections.

Whether a flow element uses an InlineCollection (Inlines) or BlockCollection (Blocks) to contain its child elements depends on what type of child elements (Block or Inline) can be contained by the parent. Containment rules for flow content elements are summarized in the content schema in the next section.

Note: There is a third type of collection used with flow content, the ListItemCollection, but this collection is only used with a List. In addition, there are several collections used with Table. See Table Overview for more information.

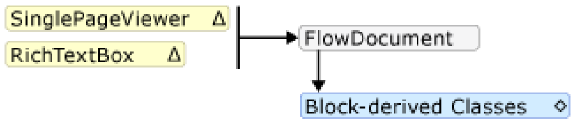
Content Schema

Given the number of different flow content elements, it can be overwhelming to keep track of what type of child elements an element can contain. The diagram below summarizes the containment rules for flow elements. The arrows represent the possible parent/child relationships.



As can be seen from the diagram above, the children allowed for an element are not necessarily determined by whether it is a Block element or an Inline element. For example, a Span (an Inline element) can only have Inline child elements while a Figure (also an Inline element) can only have Block child elements. Therefore, a diagram is useful for quickly determining what element can be contained in another. As an example, let's use the diagram to determine how to construct the flow content of a RichTextBox.

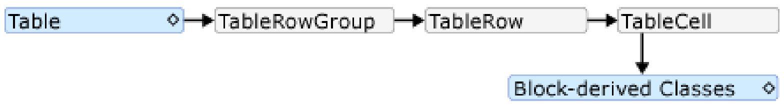
1. A RichTextBox must contain a FlowDocument which in turn must contain a Block-derived object. Below is the corresponding segment from the diagram above.



Thus far, this is what the markup might look like.

```
<RichTextBox>
  <FlowDocument>
    <!-- One or more Block-derived object… -->
  </FlowDocument>
</RichTextBox>
```

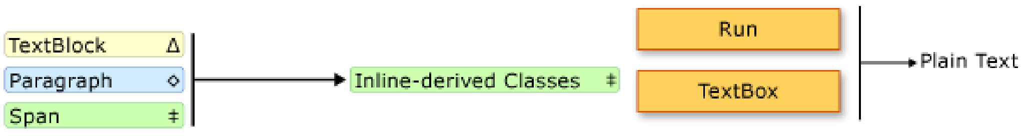
2. According to the diagram, there are several Block elements to choose from including Paragraph, Section, Table, List, and BlockUIContainer (see Block-derived classes above). Let's say we want a Table. According to the diagram above, a Table contains a TableRowGroup containing TableRow elements, which contain TableCell elements which contain a Block-derived object. Below is the corresponding segment for Table taken from the diagram above.



Below is the corresponding markup.

```
<RichTextBox>
  <FlowDocument>
    <Table>
      <TableRowGroup>
        <TableRow>
          <TableCell>
            <!-- One or more Block-derived object… -->
          </TableCell>
        </TableRow>
      </TableRowGroup>
    </Table>
  </FlowDocument>
</RichTextBox>
```

3. Again, one or more Block elements are required underneath a TableCell. To make it simple, let's place some text inside the cell. We can do this using a Paragraph with a Run element. Below is the corresponding segments from the diagram showing that a Paragraph can take an Inline element and that a Run (an Inline element) can only take plain text.



Below is the entire example in markup.

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

      <!-- Normally a table would have multiple rows and multiple
           cells but this code is for demonstration purposes.-->
      <Table>
        <TableRowGroup>
          <TableRow>
            <TableCell>
              <Paragraph>

                <!-- The schema does not actually require
                     explicit use of the Run tag in markup. It
                     is only included here for clarity. -->
                <Run>Paragraph in a Table Cell.</Run>
              </Paragraph>
            </TableCell>
          </TableRow>
        </TableRowGroup>
      </Table>
    </FlowDocument>
  </RichTextBox>
</Page>
```

Customizing Text

Usually text is the most prevalent type of content in a flow document. Although the objects introduced above can be used to control most aspects of how text is rendered, there are some other methods for customizing text that is covered in this section.

Text Decorations

Text decorations allow you to apply the underline, overline, baseline, and strikethrough effects to text (see pictures below). These decorations are added using the TextDecorations property that is exposed by a number of objects including Inline, Paragraph, TextBlock, and TextBox.

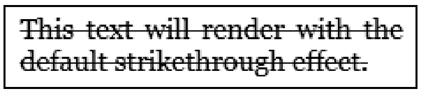
The following example shows how to set the TextDecorations property of a Paragraph.

```
<FlowDocument ColumnWidth="200">
  <Paragraph TextDecorations="Strikethrough">
    This text will render with the strikethrough effect.
  </Paragraph>
</FlowDocument>
```

C#

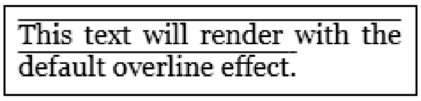
```
Paragraph parx = new Paragraph(new Run("This text will render with the strikethrough effect."));
parx.TextDecorations = TextDecorations.Strikethrough;
```

The following figure shows how this example renders.

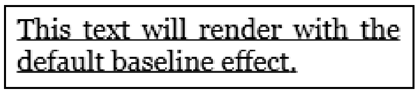


~~This text will render with the default strikethrough effect.~~

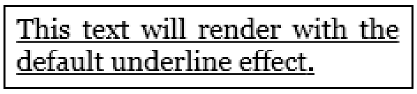
The following figures show how the **Overline**, **Baseline**, and **Underline** decorations render, respectively.



This text will render with the default overline effect.



This text will render with the default baseline effect.



This text will render with the default underline effect.

Typography

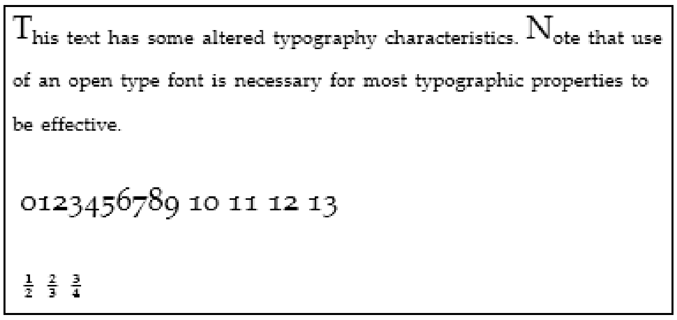
The Typography property is exposed by most flow-related content including TextElement, FlowDocument, TextBlock, and TextBox. This property is used to control typographical characteristics/variations of text (i.e. small or large caps, making superscripts and subscripts, etc).

The following example shows how to set the Typography attribute, using Paragraph as the example element.

```
<Paragraph
  TextAlignment="Left"
  FontSize="18"
  FontFamily="Palatino Linotype"
  Typography.NumeralStyle="OldStyle"
  Typography.Fraction="Stacked"
  Typography.Variants="Inferior"
>
  <Run>
    This text has some altered typography characteristics.  Note
    that use of an open type font is necessary for most typographic
    properties to be effective.
  </Run>
  <LineBreak/><LineBreak/>
  <Run>
```

```
    0123456789 10 11 12 13
  </Run>
  <LineBreak/><LineBreak/>
  <Run>
    1/2 2/3 3/4
  </Run>
</Paragraph>
```

The following figure shows how this example renders.

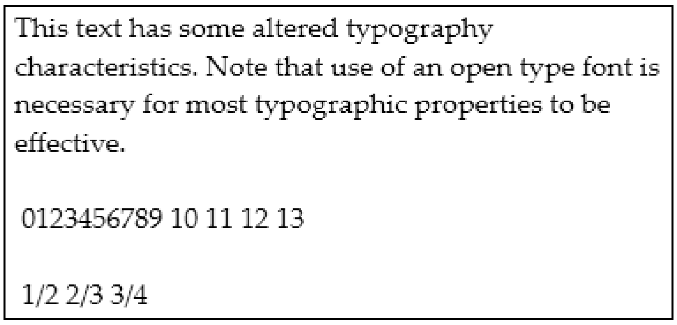


This text has some altered typography characteristics. Note that use of an open type font is necessary for most typographic properties to be effective.

0123456789 10 11 12 13

$\frac{1}{2}$ $\frac{2}{3}$ $\frac{3}{4}$

In contrast, the following figure shows how a similar example with default typographic properties renders.



This text has some altered typography characteristics. Note that use of an open type font is necessary for most typographic properties to be effective.

0123456789 10 11 12 13

1/2 2/3 3/4

The following example shows how to set the Typography property programmatically.

C#

```
Paragraph par = new Paragraph();

Run runText = new Run(
    "This text has some altered typography characteristics.  Note" +
    "that use of an open type font is necessary for most typographic" +
    "properties to be effective.");
Run runNumerals = new Run("0123456789 10 11 12 13");
Run runFractions = new Run("1/2 2/3 3/4");

par.Inlines.Add(runText);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runNumerals);
par.Inlines.Add(new LineBreak());
par.Inlines.Add(new LineBreak());
par.Inlines.Add(runFractions);

par.TextAlignment = TextAlignment.Left;
par.FontSize = 18;
par.FontFamily = new FontFamily("Palatino Linotype");

par.Typography.NumeralStyle = FontNumeralStyle.OldStyle;
par.Typography.Fraction = FontFraction.Stacked;
par.Typography.Variants = FontVariants.Inferior;
```

See Typography in WPF for more information on typography.

See Also

Concepts

Optimizing Performance: Text

Typography in WPF
TextElement Content Model Overview
RichTextBox Overview
Documents in WPF
Table Overview
Annotations Overview

Other Resources

Flow Content Elements How-to Topics

Did you find this helpful? ◯ Yes ◯ No