Документ

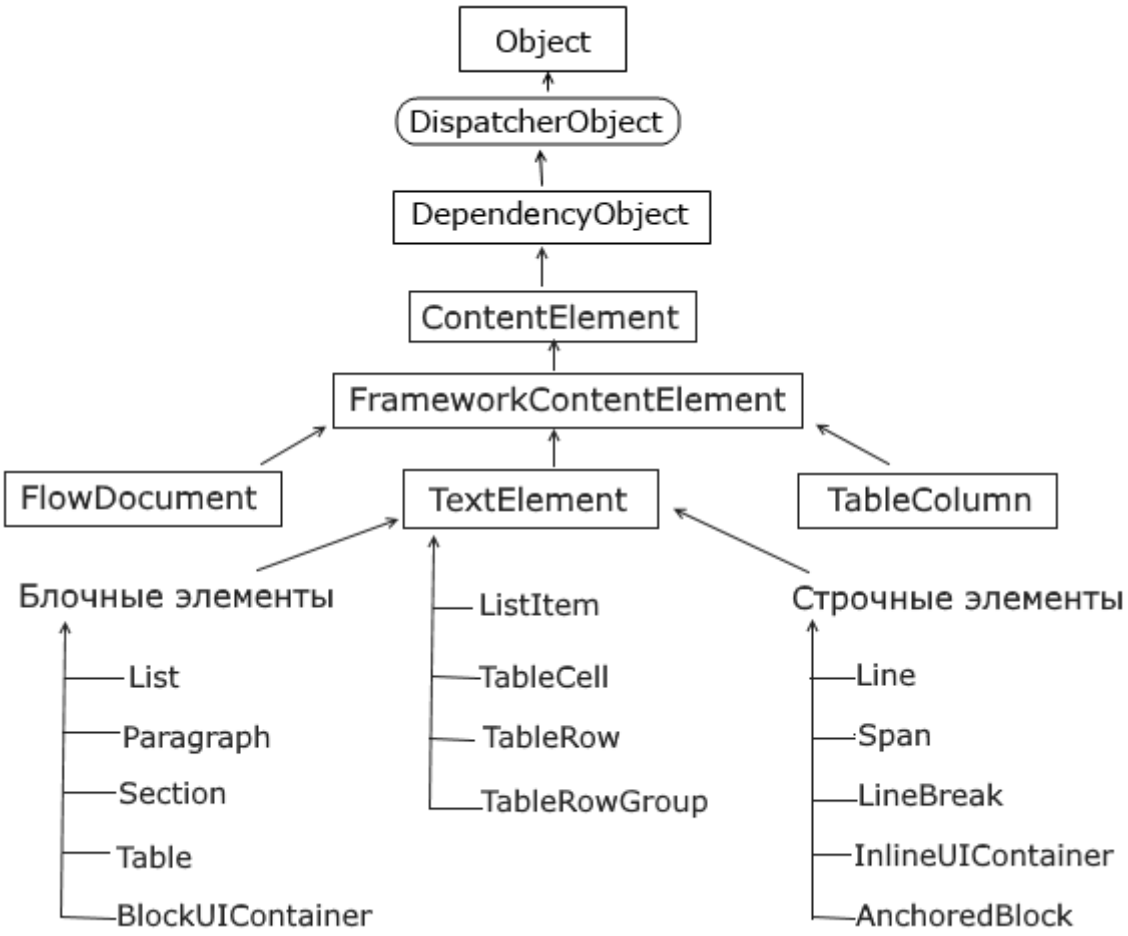
В предыдущих главах для вывода текста использовался элемент TextBlock. Однако этого элемента недостаточно для создания насыщенных приложений с большими объемами текста со сложным форматированием и встроенными изображениями. Для этого нам надо использовать документы.

Итак, все документы в WPF деляться на две группы:

- **Фиксированные документы (fixed documents)**. Формат и расположение содержимого таких документов фиксировано и не может быть изменено. На различных устройствах с различным разрешением экрана содержимое будет выглядеть одинаково и не будет оптимизировано. Такие документы преимущественно предназначены для печати. Для фиксированных документов WPF использует стандарт XPS (XML Paper Specification)
- **Потоковые документы (flow documents)**. Эти документы предназначены для просмотра на экране монитора. А WPF выполняет оптимизацию документа под конкретные параметры среды.

Потоковые документы

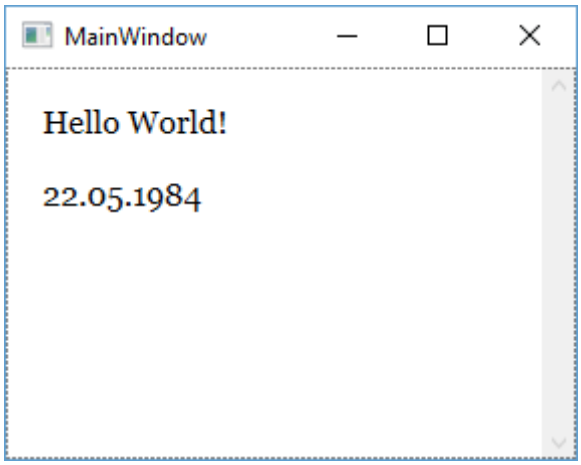
Потоковые документы в WPF представлены классом FlowDocument, который может включать в себя разлиные потоковые элементы (flow elements). Все эти элементы не являются стандартными элементами управления, как, например, Button или TextBlock, а наследуются от базового класса **FrameworkContentElement** и поэтому поддерживают такие механизмы, как привязка, анимация и другие, но не используют компоновку. В итоге всю иерархию потоковых элементов можно представить следующим образом:



Создание потоковых документов

Для использования объекта FlowDocument мы должны поместить его в один из контейнеров - FlowDocumentReader, FlowDocumentPageViewer или FlowDocumentScrollViewer. Например:

```
<FlowDocumentScrollViewer>
    <FlowDocument>
        <Paragraph>Hello World!</Paragraph>
        <Paragraph>22.05.1984</Paragraph>
    </FlowDocument>
</FlowDocumentScrollViewer>
```



Содержимое потоковых документов

В качестве содержимого FlowDocument принимает один или несколько потоковых элементов. Все эти элементы являются наследниками класса **TextElement** и могут быть блочными (block) и строчными (inline).

Блочные элементы

К блочным элементам относят следующие : **Paragraph**, **List**, **Table**, **BlockUIContainer** и **Section**.

Элемент Paragraph

Элемент Paragraph содержит коллекцию **Inlines**, которая в свою очередь включает строковые элементы, причем не только текст. Чтобы параграф отображал текст, надо использовать строчный элемент **Run**:

```
<Paragraph  x:Name="p1" TextIndent="20">
        <Run>Hello World!</Run>
</Paragraph>
```

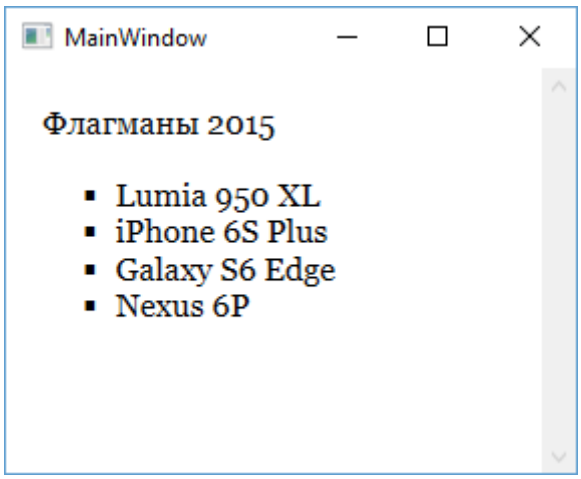
Хотя мы можем не использовать **Run** и напрямую писать текст в содержимое параграфа, однако в этом случае элемент **Run** все равно будет создан, только неявно. Поэтому чтобы в данном случае получить в коде содержимое параграфа, нам надо получить текст элемента Run:

```
string s = ((Run)p1.Inlines.FirstInline).Text;
```

Элемент List

Блочный элемент List представляет собой список. Он содержит коллекцию элементов **ListItem**, которые и представляют элементы списка. Каждый из элементов **ListItem**, в свою очередь, может включать другие блочные элементы, например, **Paragraph**:

```
<FlowDocumentScrollViewer >
    <FlowDocument>
        <Paragraph>Флагманы 2015</Paragraph>
        <List MarkerStyle="Box">
            <ListItem>
                <Paragraph>Lumia 950 XL</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>iPhone 6S Plus</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>Galaxy S6 Edge</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>Nexus 6P</Paragraph>
            </ListItem>
        </List>
    </FlowDocument>
</FlowDocumentScrollViewer>
```



С помощью свойства **MarkerStyle** можно задать формат списка:

- Disc: стандартный черный кружочек. Значение по умолчанию
- Box: черный квадратик, как в примере выше
- Circle: кружок без наполнения
- Square: квадратик без наполнения
- Decimal: десятичные цифры от 1, то есть обычный нумерованный список
- LowerLatin: строчные латинские буквы (a, b, c)
- UpperLatin: заглавные латинские буквы (A, B, C)

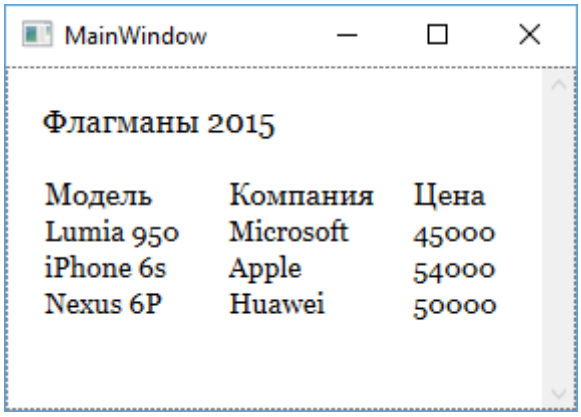
- LowerRoman: латинские цифры в нижнем регистре (i, iv, x)
- UpperRoman: латинские цифры в верхнем регистре (I, IV, X)
- None: отсутствие маркера списка

Элемент Table

Элемент Table организует вывод содержимого в виде таблицы. Он имеет вложенный элемент **TableRowGroup**. Этот элемент позволяет задать однообразный вид таблицы и содержит коллекцию строк - элементов **TableRow** (строку таблицы). А каждый элемент TableRow содержит несколько элементов **TableCell** (ячейка таблицы). В элементе TableCell затем уже размещаются блочные элементы с содержимым, например, элементы Paragraph:

```
<FlowDocumentScrollViewer >
    <FlowDocument>
        <Paragraph>Флагманы 2015</Paragraph>
        <Table>
            <Table.Columns>
                <TableColumn Width="2*" />
                <TableColumn Width="2*" />
                <TableColumn Width="*" />
            </Table.Columns>
            <TableRowGroup FontSize="14">
                <TableRow FontSize="15">
                    <TableCell>
                        <Paragraph>Модель</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>Компания</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>Цена</Paragraph>
                    </TableCell>
                </TableRow>
                <TableRow>
                    <TableCell>
                        <Paragraph>Lumia 950</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>Microsoft</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>45000</Paragraph>
                    </TableCell>
                </TableRow>
                <TableRow>
                    <TableCell>
                        <Paragraph>iPhone 6s</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>Apple</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>54000</Paragraph>
                    </TableCell>
                </TableRow>
                <TableRow>
                    <TableCell>
                        <Paragraph>Nexus 6P</Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph>Huawei</Paragraph>
                    </TableCell>
```

```
                    <TableCell>
                        <Paragraph>50000</Paragraph>
                    </TableCell>
                </TableRow>
            </TableRowGroup>
        </Table>
    </FlowDocument>
</FlowDocumentScrollViewer>
```



Для определения столбцов в элементе применяется коллекция Table.Columns. Каждый столбец представляет элемент TableColumn, у которого мы можем задать ширину.

Элемент Section

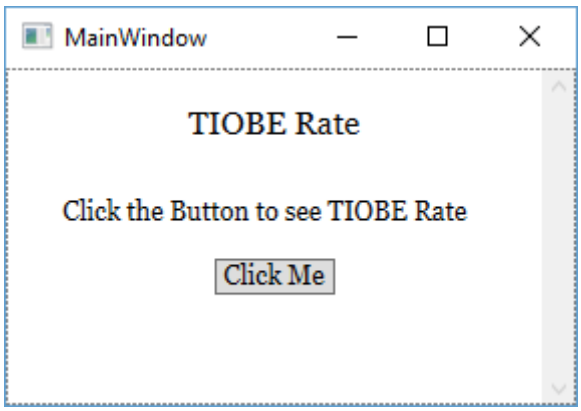
Элемент Section предназначен для группировки других блочных элементов и предназначен прежде всего для однобразной стилизации этих элементов:

```
<FlowDocument>
    <Section FontSize="16">
        <Paragraph>Флагманы 2016</Paragraph>
        <Paragraph>Xiaomi Mi5</Paragraph>
        <Paragraph>Samsung Galaxy S7</Paragraph>
        <Paragraph>HP Elite X3</Paragraph>
    </Section>
</FlowDocument>
```

Элемент BlockUIContainer

Элемент BlockUIContainer позволяет добавить в документ различные элементы управления, которые не являются блочными или строчными элементами. Так, мы можем добавить кнопку или картинку к документу. Такая функциональность особенно полезна, когда необходимо добавить интерактивную связь с пользователем:

```
<FlowDocument>
    <Paragraph TextAlignment="Center">TIOBE Rate</Paragraph>
    <BlockUIContainer FontSize="13">
        <StackPanel Orientation="Vertical">
            <TextBlock Height="40" Padding="10">Click the Button to see TIOBE Rate</TextBlock>
            <Button Width="60">Click Me</Button>
        </StackPanel>
    </BlockUIContainer>
</FlowDocument>
```



Строчные элементы

Выше мы уже использовали элемент **Run**, который хранит некоторый текст, выводимый в блочном элементе, например, в элементе Paragraph.

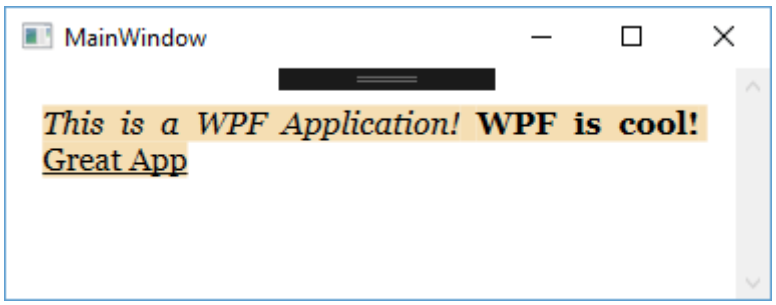
Span

Элемент **Span** объединяет другие строчные элементы и применяет ним определенное форматирование:

```
<FlowDocument>
    <Paragraph>
        <Span Background="Red">
            <Run>This is a WPF Application!</Run>
            <Run>WPF is cool</Run>
        </Span>
    </Paragraph>
</FlowDocument>
```

Чтобы задать для текста отдельные способы форматирования, применяются элементы **Bold**,**Italic** и **Underline**, которые позволяют создать текст полужирным шрифтом, курсивом и подчеркнутый текст соответственно.

```
<FlowDocument>
    <Paragraph>
        <Span Background="Wheat">
            <Italic>This is a WPF Application!</Italic>
            <Bold>WPF is cool!</Bold>
            <Underline>Great App</Underline>
        </Span>
    </Paragraph>
</FlowDocument>
```



Элемент **Hyperlink** позволяет вставить в документ ссылку для перехода и может использоваться в навигационных приложениях:

```
<Paragraph>
     <Hyperlink NavigateUri="http:\\microsoft.com">Microsoft</Hyperlink>
</Paragraph>
```

Элемент **LineBreak** задает перевод строки:

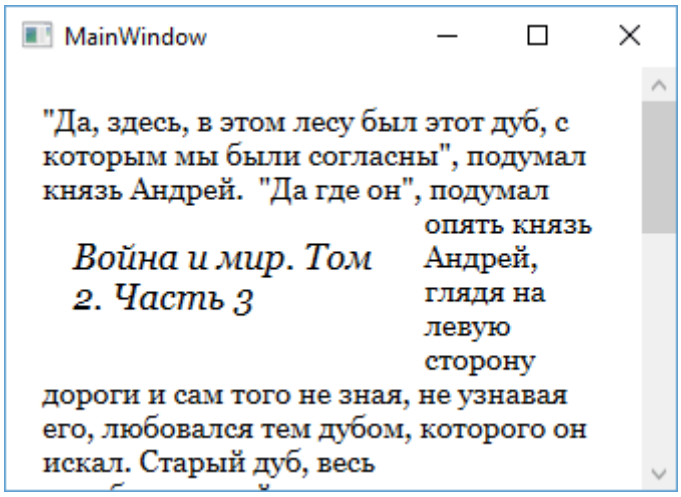
```
<FlowDocument>
    <Paragraph>
        <Run>Перевод</Run>
        <LineBreak />
        <Run>на другую строку</Run>
    </Paragraph>
</FlowDocument>
```

InlineUIContainer подобен элементу **BlockUIContainer** и также позволяет помещать другие элементы управления, например, кнопки, только является строковым.

Элементы **Floater** и **Figure** позволяют вывести плавающее окно с некоторой информацией, текстом, картинками и прочим:

```
<FlowDocument>
    <Paragraph TextAlignment="Left" FontSize="15">
        "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал князь Андрей.

        <Floater Width="170" Padding="5" HorizontalAlignment="Left" FontSize="18" FontStyle="Ita
            <Paragraph>Война и мир. Том 2. Часть 3</Paragraph>
        </Floater>

        "Да где он", подумал опять князь Андрей, глядя на левую сторону дороги и сам того не зна
    </Paragraph>
</FlowDocument>
```

Остальной текст будет обтекать элемент Floater, заданный таким образом, справа.



Figure

Figure во многом аналогичен элементу Floater за тем исключением, что дает больше возможностей по контролю за позиционированием. Так, следующая разметка будет создавать эффект, аналогичный примеру с Floater:

```
<FlowDocument>
    <Paragraph TextAlignment="Left" FontSize="15">
        "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал князь Андрей.
        <Figure Width="170" Padding="5" HorizontalAnchor="ContentLeft" FontSize="18" FontStyle="
            <Paragraph>Война и мир. Том 2. Часть 3</Paragraph>
        </Figure>
        "Да где он", подумал опять князь Андрей, глядя на левую сторону дороги и сам того не зна
    </Paragraph>
</FlowDocument>
```

С помощью свойства **HorizontalAnchor** мы можем управлять позиционированием элемента по горизонтали. Так, значение ContentLeft позволяет выравнить текст по левой стороне,

ContentRight - по правой стороне, а значение ContentCenter - по центру.

Другое свойство **VerticalAnchor** позволяет выравнить содержимое Figure по вертикали.

Контейнеры потоковых документов

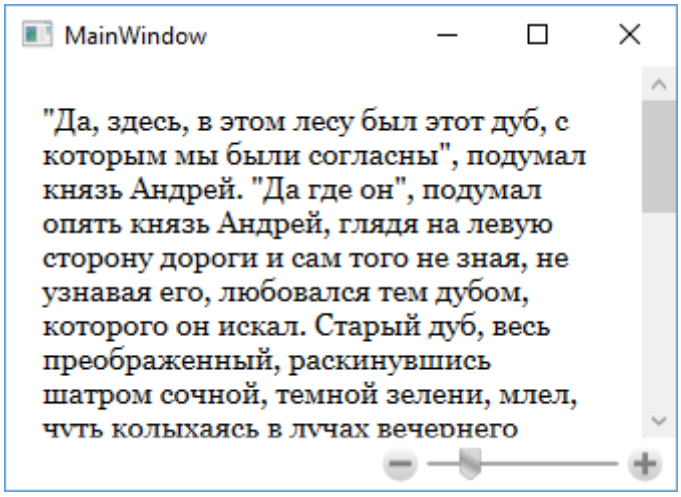
В WPF используется три контейнера для потоковых элементов: **FlowDocumentScrollViewer**, **FlowDocumentPageViewer** и **FlowDocumentReader**

FlowDocumentScrollViewer

FlowDocumentScrollViewer отображает документ с полосой прокрутки. Документ отображается как единое целое.

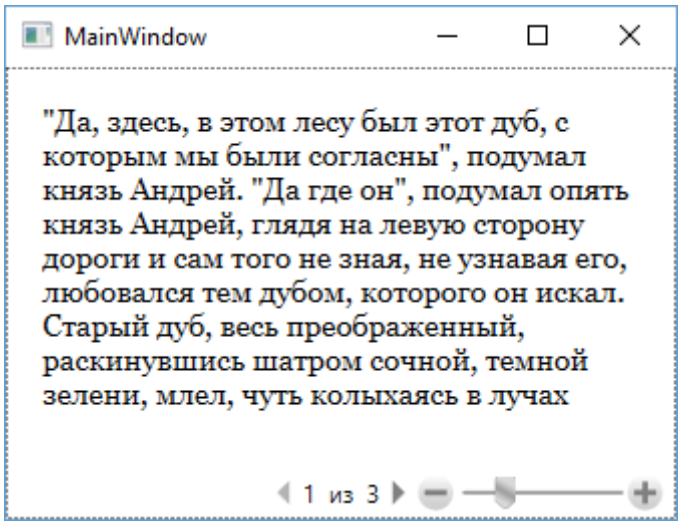
Из свойств следует отметить **IsToolBarVisible** - если оно имеет значение true, то у элемента появляется панель с функциями масштабирования:

```
<FlowDocumentScrollViewer IsToolBarVisible="True">
    <FlowDocument>
        <Paragraph TextAlignment="Left" FontSize="15">
            "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал князь Анд
        </Paragraph>
    </FlowDocument>
</FlowDocumentScrollViewer>
```



FlowDocumentPageViewer

FlowDocumentPageViewer разбивает документ на страницы:



Этот элемент также позволяет создать не только многостраничный способ отображения, но и разбить текст на несколько столбцов, если окно имеет необходимую длину. Для управления столбцами у элемента FlowDocument можно настроить следующие свойства:

ColumnWidth: устанавливает ширину столбца

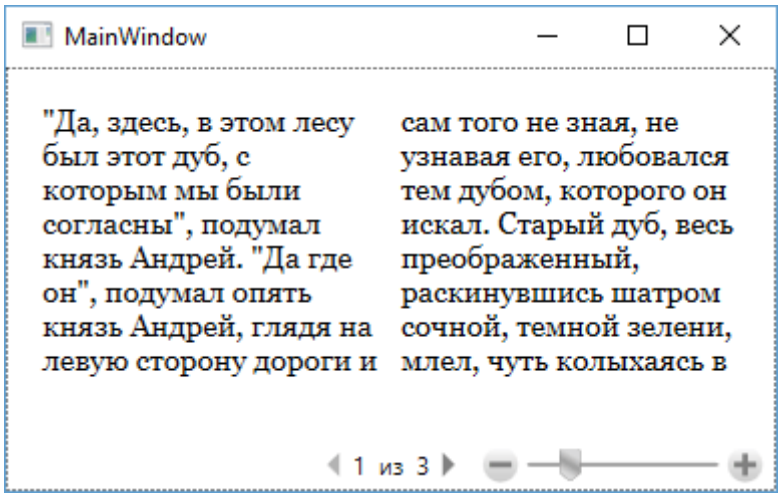
ColumnGap: устанавливает расстояние между столбцами

IsColumnWidthFlexible: при значении True контейнер сам корректирует ширину столбца

ColumnRuleWidth и **ColumnRuleBrush**: устанавливает соответственно ширину границы мужду столбцаи и ее цвет

Например:

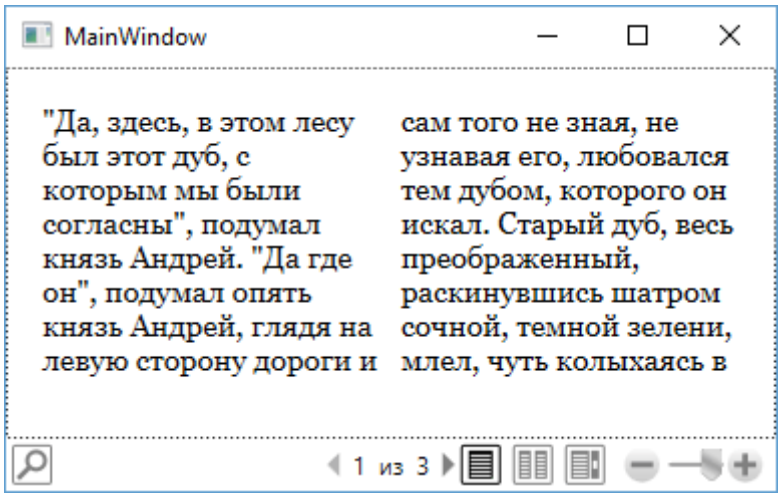
```
<FlowDocumentPageViewer>
    <FlowDocument ColumnWidth="150" ColumnGap="10">
        <Paragraph TextAlignment="Left" FontSize="15">
            "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал князь Анд
        </Paragraph>
    </FlowDocument>
</FlowDocumentPageViewer>
```



FlowDocumentReader

объединяет возможности элементов FlowDocumentScrollViewer и FlowDocumentPageViewer и позволяет переключаться между разными режимами отображения документа.

```
<FlowDocumentReader>
    <FlowDocument ColumnWidth="150" ColumnGap="10">
        <Paragraph TextAlignment="Left" FontSize="15">
            "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал князь Анд
        </Paragraph>
    </FlowDocument>
</FlowDocumentReader>
```



Загрузка и сохранение документов

WPF предоставляет возможности по загрузке и сохранению потоковых документов. Для примера определим соедующий код интерфейса:

```
<Window x:Class="DocumentsApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:DocumentsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="400">
    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="Height" Value="25" />
            <Setter Property="Width" Value="80" />
            <Setter Property="Margin" Value="5" />
        </Style>
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <FlowDocumentScrollViewer x:Name="docViewer">
            <FlowDocument>
                <Paragraph TextAlignment="Left" FontSize="15">
                    "Да, здесь, в этом лесу был этот дуб, с которым мы были согласны", подумал к
                </Paragraph>
            </FlowDocument>
        </FlowDocumentScrollViewer>
        <StackPanel Orientation="Horizontal" Grid.Row="1" HorizontalAlignment="Center">
            <Button Content="Сохранить" Click="Save_Click" />
            <Button Content="Удалить"  Click="Clear_Click" />
            <Button Content="Загрузить"  Click="Load_Click" />
        </StackPanel>
    </Grid>
</Window>
```

Здесь определен контейнер FlowDocumentScrollViewer с именем "docViewer". Внутри него потоковый документ. И для управления документа определены три кнопки.

Далее определим обработчики кнопок в файле кода:

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

namespace DocumentsApp
{
    public partial class MainWindow : Window
    {
        string path = "mydoc.xaml";
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            using (FileStream fs = File.Open(path, FileMode.Create))
            {
                if (docViewer.Document != null)
                {
                    XamlWriter.Save(docViewer.Document, fs);
                    MessageBox.Show("Файл сохранен");
                }
            }
        }
        private void Clear_Click(object sender, RoutedEventArgs e)
        {
            docViewer.ClearValue(FlowDocumentScrollViewer.DocumentProperty);
        }
        private void Load_Click(object sender, RoutedEventArgs e)
        {
            using (FileStream fs = File.Open(path, FileMode.Open))
            {
                FlowDocument document = XamlReader.Load(fs) as FlowDocument;
                if (document != null)
                    docViewer.Document = document;
            }
        }
    }
}
```

Для сохранения и загрузки документа определим путь mydoc.xaml. Далее с помощью классов XamlWriter/XamlReader происходит чтение и запись в файл mydoc.xaml.

RichTextBox и редактирование документов

Если для отображения потокового документа в WPF предназначены элементы FlowDocumentPageViewer, FlowDocumentScrollViewer и FlowDocumentReader, то для его редактирования применяется элемент **RichTextBox**.

Чтобы загрузить в RichTextBox файл, мы можем воспользоваться методом XamlReader.Load, который загружает xaml-содержимое. Однако чтобы загрузить иное содержимое, например, файлы rtf или простой текст, нам надо воспользоваться классом **TextRange**, который сам преобразует документ из формата rtf в объект FlowDocument.

Для работы с RichTextBox определим следующую разметку:

```
<Window x:Class="DocumentsApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:DocumentsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="400">
    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="Height" Value="25" />
            <Setter Property="Width" Value="80" />
            <Setter Property="Margin" Value="25 5 25 5" />
        </Style>
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <RichTextBox x:Name="docBox" />
        <StackPanel Orientation="Horizontal" Grid.Row="1" HorizontalAlignment="Center">
            <Button Content="Сохранить" Click="Save_Click" />
            <Button Content="Загрузить"  Click="Load_Click" />
        </StackPanel>
    </Grid>
</Window>
```

А в файле кода установим обработчики нажатия кнопок:

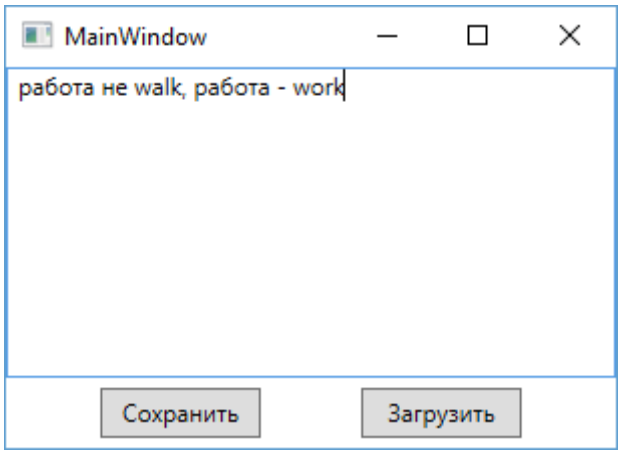
```
using Microsoft.Win32;
using System.IO;
using System.Windows;
using System.Windows.Documents;

namespace DocumentsApp
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "Text Files (*.txt)|*.txt|RichText Files (*.rtf)|*.rtf|XAML Files (*.xa
            if (sfd.ShowDialog() == true)
            {
                TextRange doc = new TextRange(docBox.Document.ContentStart, docBox.Document.Cont
                using (FileStream fs = File.Create(sfd.FileName))
                {
                    if (Path.GetExtension(sfd.FileName).ToLower() == ".rtf")
                        doc.Save(fs, DataFormats.Rtf);
                    else if (Path.GetExtension(sfd.FileName).ToLower() == ".txt")
                        doc.Save(fs, DataFormats.Text);
                    else
                        doc.Save(fs, DataFormats.Xaml);
                }
            }
        }

        private void Load_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "RichText Files (*.rtf)|*.rtf|All files (*.*)|*.*";

            if (ofd.ShowDialog() == true)
            {
                TextRange doc = new TextRange(docBox.Document.ContentStart, docBox.Document.Cont
                using (FileStream fs = new FileStream(ofd.FileName, FileMode.Open))
                {
                    if (Path.GetExtension(ofd.FileName).ToLower() == ".rtf")
                        doc.Load(fs, DataFormats.Rtf);
                    else if (Path.GetExtension(ofd.FileName).ToLower() == ".txt")
                        doc.Load(fs, DataFormats.Text);
                    else
                        doc.Load(fs, DataFormats.Xaml);
                }
```

```
            }
        }
    }
}
```

Чтобы открыть файл, мы используем класс **TextRange**. Конструктор этого класса принимает в качестве параметров начальную и конечную точки документа, которые определяют ту часть документа, которую надо изменить. В данном случае мы изменяем весь документ, поэтому указываем в качестве начальной точки начало документа, а в качестве конечной - конец документа.

Далее мы получаем поток, связанный с выбранным файлом и загружаем его в RichTextBox с помощью метода Load - Load(fs,DataFormats.Rtf). Здесь мы указываем значение перечисления DataFormats, которое позволяет нам правильно преобразовать тип документа в объект FlowDocument.

При сохранении документа мы также создаем объект TextRange, который в конструкторе принимает начало и конец документа. И затем используем метод Save для сохранения документа в нужный формат.



Но при работе с документами xaml в данном случае надо учитывать, что TextRange загружает и сохраняет документ с корневым узлом Section. И если у нас содержимое файла, который надо загрузить, отличается по структуре, то опять же для загрузки xaml-файлов мы можем использовать другой способ:

```
FlowDocument document = System.Windows.Markup.XamlReader.Load(fs) as FlowDocument;
if (document != null)
    docBox.Document = document;
```

Фиксированные документы

Фиксированные документы характеризуются точной неизменной компоновкой и предназначены преимущественно для печати, но могут использоваться также и для чтения текста с экрана. К таким документам в WPF относятся документы на основе XPS. Для просмотра XPS-документа используется элемента DocumentViewer. В качестве содержимого DocumentViewer принимает элемент FixedDocument, который как раз и представляет фиксированный документ.

Например:

```
<Window x:Class="DocumentsApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:DocumentsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="400">
    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="Height" Value="25" />
            <Setter Property="Width" Value="80" />
            <Setter Property="Margin" Value="25 5 25 5" />
        </Style>
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
```

Фиксированный документ FixedDocument может принимать различное количество страниц. Каждая страница представляет элемент **PageContent**. В этот элемент помещается объект **FixedPage**, в который в свою очередь помещаются другие элементы - это могут эллипсы, текстовые поля и т.д. Так, в данном случае у нас две страницы. Используя свойства Width и Height, мы можем управлять размером страниц.

Также, здесь мы разместили две кнопки для загрузки и сохранения документа. Теперь в файле кода c# определим для них обработчики событий:

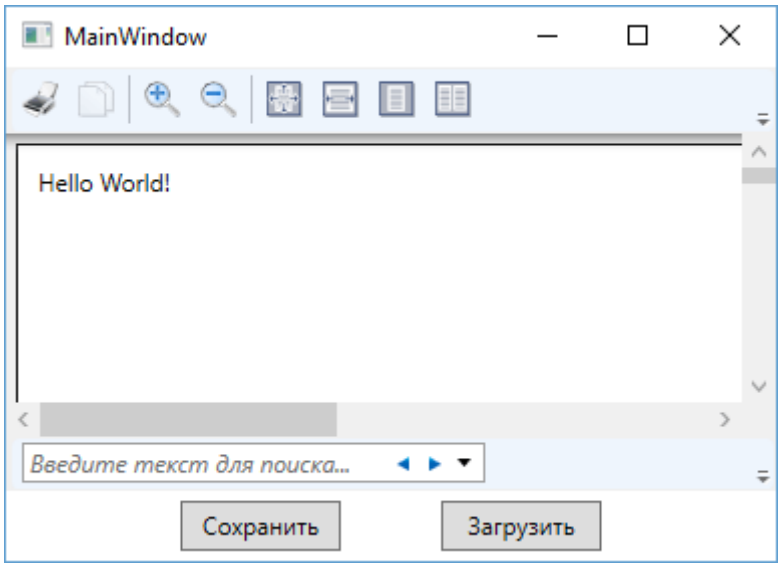
```
using Microsoft.Win32;
using System.IO;
using System.Windows;
using System.Windows.Xps.Packaging;
using System.Windows.Xps;
using System.Windows.Documents;

namespace DocumentsApp
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "XPS Files (*.xps)|*.xps";
            if (sfd.ShowDialog() == true)
            {
                XpsDocument doc = new XpsDocument(sfd.FileName, FileAccess.Write);
                XpsDocumentWriter writer = XpsDocument.CreateXpsDocumentWriter(doc);
                writer.Write(documentViewer.Document as FixedDocument);
                doc.Close();
            }
        }

        private void Load_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "XPS Files (*.xps)|*.xps";

            if (ofd.ShowDialog() == true)
            {
                XpsDocument doc = new XpsDocument(ofd.FileName, FileAccess.Read);
                documentViewer.Document = doc.GetFixedDocumentSequence();
            }
        }
    }
}
```

Для сохранения и открытия документа применяется класс **XpsDocument**. Для его использования нам надо добавить в проект библиотеки **ReachFramework.dll** и **System.Printing.dll**.

Чтобы сохранить документ, получаем объект XpsDocumentWriter и вызываем его метод Write().

Для открытия документа просто используем метод GetFixedDocumentSequence() объекта XpsDocument.

И после запуска приложения мы сможем увидеть наш определенный в разметке xaml документ:



Как видно из скриншота, контейнер фиксированных документов DocumentViewer уже имеет некоторую базовую функциональность (например, масштабирование, поиск), которые мы можем использовать.

Для эксперимента можно также попробовать создать простейшие xps-файлы в MS Word (для этого стандартный документ docx можно экспортировать в формат xps) и загрузить этот документ в нашу программу.

Аннотации

Потоковые и фиксированные документы поддерживают такую функциональность как аннотации. Аннотации позволяют добавлять к документам комментарии, выделять какие-то куски текста и т.д.

Для работы с аннотациями определим следующую разметку интерфейса:

```
<Window x:Class="AnnotationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnnotationApp"
        xmlns:a="clr-namespace:System.Windows.Annotations;assembly=PresentationFramework"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="400">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ToolBar>
            <Button Command="a:AnnotationService.CreateTextStickyNoteCommand" FontSize="10">
                Комментировать
            </Button>
            <Button Command="a:AnnotationService.CreateHighlightCommand" FontSize="10"
                    CommandParameter="{x:Static Brushes.Yellow}">
                Выделить
            </Button>
            <Button Command="a:AnnotationService.ClearHighlightsCommand" FontSize="10">
                Убрать выделение
            </Button>
            <Button Command="a:AnnotationService.DeleteStickyNotesCommand" FontSize="10">
                Убрать комментарий
            </Button>
        </ToolBar>
        <FlowDocumentScrollViewer x:Name="docViewer" Grid.Row="1">
            <FlowDocument FontSize="12">
                <Paragraph TextIndent="20">
                    В 1808 году император Александр ездил в Эрфурт для новой встречи с Наполеонс
                    и в высшем свете много говорили о важности этого события.
                    В 1809 году близость двух «властелинов мира», как называли Александра и Напс
                    дошла до того, что когда Наполеон объявил войну Австрии, русский корпус выст
                    чтобы сражаться на стороне бывшего противника против бывшего союзника, австр
                </Paragraph>
            </FlowDocument>
        </FlowDocumentScrollViewer>
    </Grid>
</Window>
```

Итак, начнем с начала. В строчке xmlns:a="clr-namespace:System.Windows.Annotations;assembly=PresentationFramework" мы подключаем пространство имен для аннотаций и отображаем его на префикс a. Далее в панели инструментов мы создаем ряд кнопок, с помощью которых мы будем управлять аннотациями. Первая кнопка предназначена для создания текстовой аннотации. Для этого мы используем

команду **AnnotationService.CreateTextStickyNoteCommand**. Мы также можем использовать одноименный метод у объекта AnnotationService и создать аннотацию в коде. Затем для следующих кнопок мы добавляем команды создания графической аннотации, выделения цветом и удаления аннотаций. Обратите внимание, что для выделения цветом мы передаем в команду параметр - в данном случае цвет CommandParameter="{x:Static Brushes.Yellow}".

Всего для аннотаций нам доступно шесть команд:

- AnnotationService.ClearHighlightsCommand: удаляет выделение цветом
- AnnotationService.CreateHighlightCommand: добавляет выделение цветом
- AnnotationService.CreateInkStickyNoteCommand: добавляет графический комментарий
- AnnotationService.CreateTextStickyNoteCommand: добавляет обычный текстовый комментарий
- AnnotationService.DeleteAnnotationsCommand: удаляет графические и текстовые комментарии, а также убирает выдеоение цветом
- AnnotationService.DeleteStickyNotesCommand: удаляет графические и текстовые комментарии

При этом нам не надо определять для кнопок обработчики нажатия, так как команды аннотаций все сделают за нас.

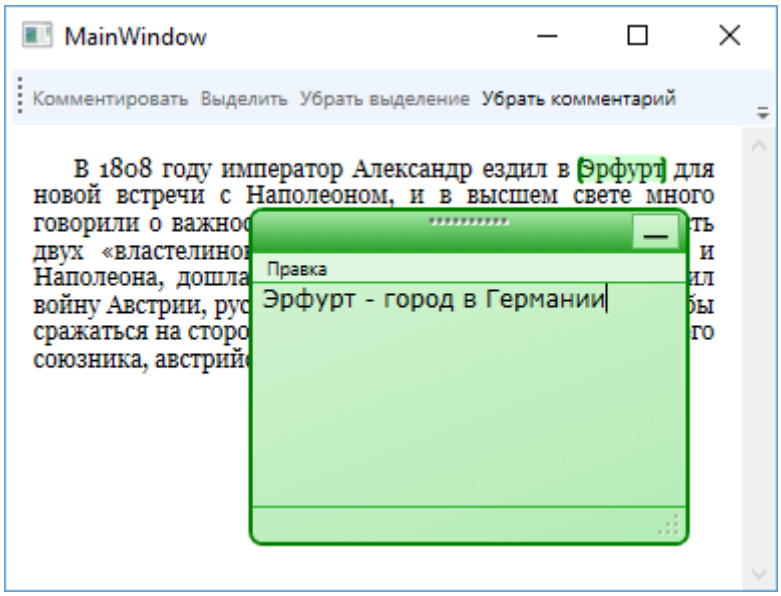
Все основные классы по работе с аннотациями хранятся в пространстве имен **System.Windows.Annotations**. Чтобы сделать аннотации доступными для нашего приложения, их надо подключить. Подключение сделаем в обработчике события загрузки окна. Также удалим поддержку аннотаций из приложения в обработчике закрытия окна:

```
using System.IO;
using System.Windows;
using System.Windows.Annotations;
using System.Windows.Annotations.Storage;

namespace AnnotationApp
{
    public partial class MainWindow : Window
    {
        FileStream fs;
        AnnotationService anService;

        public MainWindow()
        {
            InitializeComponent();
            this.Loaded += Window_Loaded;
            this.Unloaded += Window_Unloaded;
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            //инициализация службы аннотаций
            anService = new AnnotationService(docViewer);
            // создание связанного потока
            fs = new FileStream("storage.xml", FileMode.OpenOrCreate);
            // привязка потока к хранилищу аннотаций
            AnnotationStore store = new XmlStreamStore(fs);
            store.AutoFlush = true;
            // включение службы
            anService.Enable(store);
        }

        private void Window_Unloaded(object sender, RoutedEventArgs e)
        {
            if (anService != null && anService.IsEnabled)
            {
                anService.Store.Flush();
                anService.Disable();
                fs.Close();
            }
        }
    }
}
```

Итак, в данном коде мы создаем службу аннотаций (класс **AnnotationService**) и ему в конструктор передаем объект контейнера документа. В данном случае мы передаем переменную docViewer, которая представляет объект FlowDocumentScrollViewer, определенный в xaml. Затем создаем поток, по которому создаем хранилище аннотаций - **AnnotationStore**. С

помощью метода **Enable** делаем доступным хранилище аннотаций для объекта docViewer. Обратите внимание, что для одного контейнера документа мы можем создать один объект AnnotationService и один объект AnnotationStore. Для нового документа придется создавать эти объекты заново. Файл storage.xml будет хранить у нас аннотации, которые затем будут загружаться в документе. Пока он не создан, поэтому установим режим OpenOrCreate.

Теперь загрузим приложение и выделим часть текста - нам станут доступны кнопки с вышеописанными командами. Попробуем создать аннотацию:



После закрытия приложения наша аннотация будет сохранена в файл и затем при новом открытии программы будет подгружаться в документ.