МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ

НАПРАВЛЕНИЕ ПРИКЛАДНАЯ МАТЕМАТИКА

ГРУППА БПМ-16-2

Домашнее задание

ПО КУРСУ: «Математические методы в компьютерной графике»

ТЕМА: «Документы в WPF»

СТУДЕНТ: Новицкий Дмитрий

ПРЕПОДАВАТЕЛЬ: Куренкова Т. В.

Москва, 2020 г.

Оглавление

Введение

Работа с электронными документами крайне важна в мире информационных технологий. Набор инструментов Microsoft Office предоставляет обширный набор инструментов для работы с документами. Однако, не стоит забывать о более простых средствах для просмотра, сохранения и изменения документов, которые можно создавать с использованием передовых технологий и современных широко используемых программных средств. Одним из таких средств является WPF (Windows Presentation Foundation). С его помощью возможно создать свой набор инструментов для работы с документами.

Аналитический обзор

Типы документов в WPF

Приложение WPF разделяет документы на две основные категории на основе их предполагаемого использования; эти категории документов называются «документы фиксированного формата» и «документы нефиксированного формата».

- **Фиксированные документы** (fixed documents). Формат и расположение содержимого таких документов фиксировано и не может быть изменено. На различных устройствах с различным разрешением экрана их 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.

Фиксированные документы

Потоковые документы позволяют динамически компоновать сложное текстовое содержимое таким способом, который удобен для чтения на экране. Фиксированные документы — документы на основе XPS — гораздо менее гибки. Это документы, пригодные для печати, которые можно распространять и печатать на любом выводном устройстве в полном соответствии с первоначальным источником. В них используется точная, фиксированная компоновка, поддерживается внедрение шрифтов и исключается случайное изменение компоновки.

XPS не просто часть WPF. Это стандарт, тесно интегрированный в системы Windows Vista и Windows 7. Обе версии Windows содержат драйвер печати, который может создавать документы XPS (в любом приложении) и средство просмотра для их отображения.

Эти две части похожи на Adobe Acrobat, позволяя пользователям создавать и просматривать электронные документы, пригодные для печати, и добавлять аннотации. Кроме того, Microsoft Office 2007 позволяет сохранять документы в форматах XPS и PDF.

XPS-файлы на самом деле являются ZIP-файлами, содержащими библиотеку сжатых файлов: шрифтов, изображений и текстового содержимого для отдельных страниц (в том числе XML-разметку, похожую на XAML). Чтобы увидеть внутреннее содержимое XPS-файла, достаточно поменять его расширение на .zip и открыть его.

Вывести на экран документ XPS не труднее, чем потоковый документ. Единственным отличием является средство просмотра. Вместо использования одного из контейнеров FlowDocument (FlowDocumentReader, FlowDocumentScrollViewer или FlowDocumentPageViewer), применяется именованный DocumentViewer, в котором имеются элементы управления для поиска и изменения масштаба:

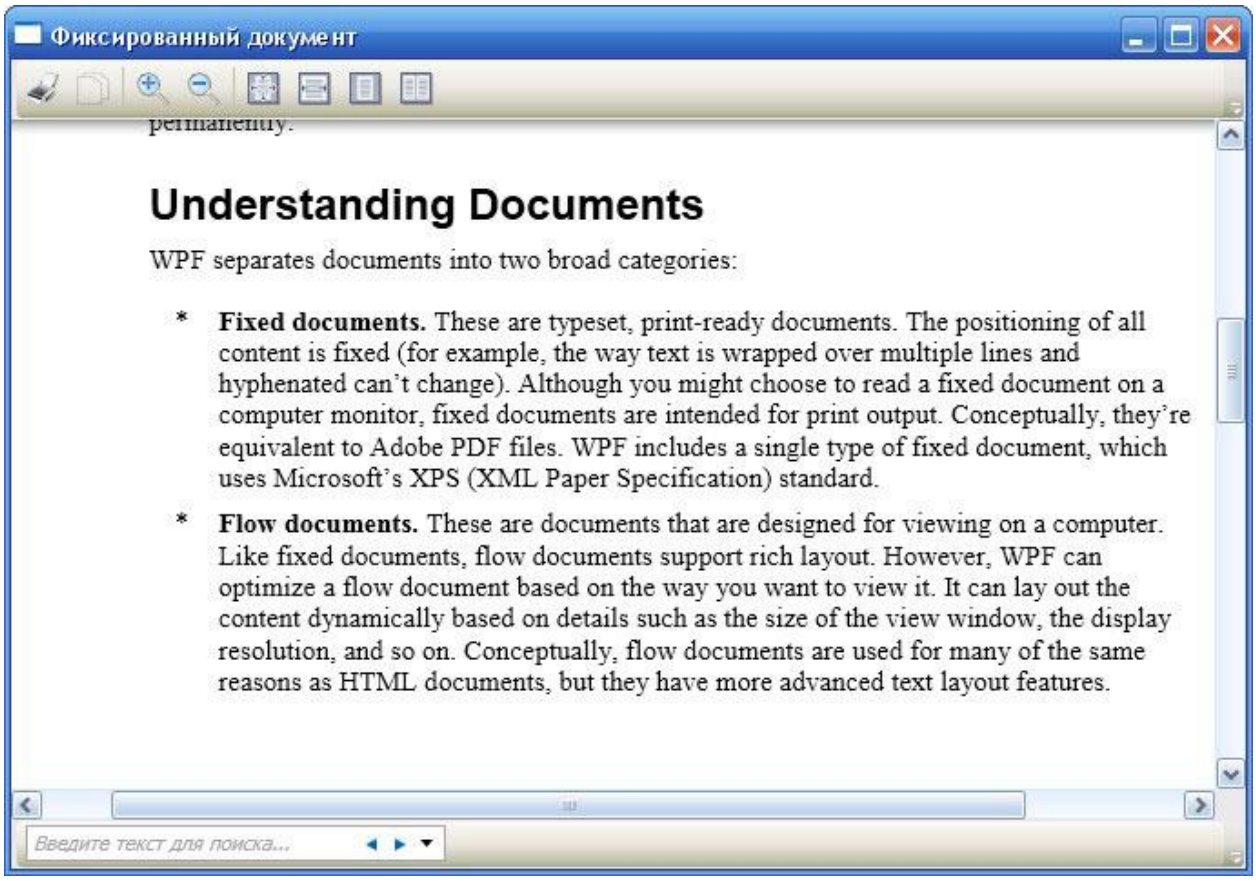


Рис. 1. Фиксированный документ.

Кроме того, он предлагает такой же набор свойств, методов и команд, как и контейнеры FlowDocument.

Вот код, который позволяет загрузить XPS-файл в память и вывести его в контейнере DocumentViewer:

```
XpsDocument doc = new XpsDocument("filename.xps", FileAccess.Read);
docViewer.Document = doc.GetFixedDocumentSequence();
doc.Close();
```

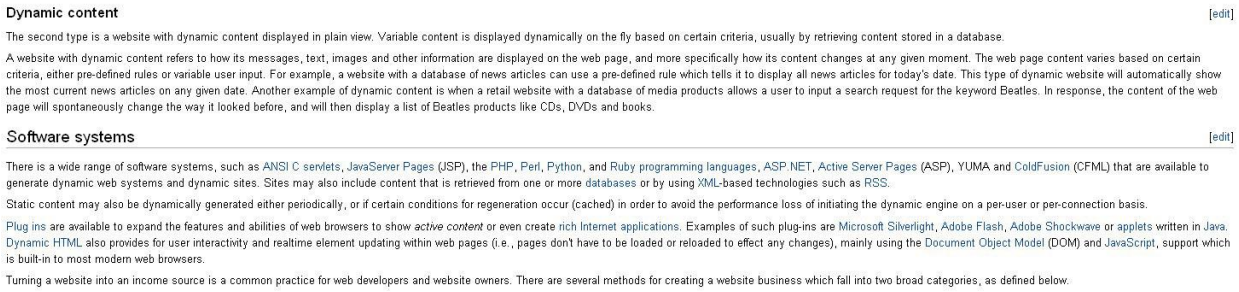
В классе XpsDocument нет ничего особенного. Он предоставляет нам метод GetFixedDocumentSequence(), который возвращает ссылку на корень документа со всем его содержимым. Он также содержит метод AddFixedDocument() для создания последовательности документов в новом документе и два метода для управления цифровыми подписями (SignDigitally() и RemoveSignature()).

XPS-документы тесно связаны с концепцией печати. Отдельный XPS-документ имеет фиксированный размер страницы и компонует свой текст так, чтобы он занял доступное место. Как и при работе с потоковыми документами, имеется непосредственная поддержка печати фиксированных документов с помощью команды ApplicationCommands.Print.

Потоковые документы

В потоковом документе содержимое адаптируется к содержащему его контейнеру. Потоковое содержимое ориентировано на экранный просмотр и лишено многих недостатков простых HTML-документов.

Обычно HTML-содержимое использует потоковую компоновку для заполнения окна браузера. (WPF упорядочивает элементы точно так же, если используется WrapPanel) Такой подход является очень гибким, но он годится лишь для ограниченного набора размеров окон. Если развернуть окно на весь экран монитора с высоким разрешением (или, что еще хуже, широкоформатного монитора), получатся длинные неудобочитаемые строки. Эта проблема демонстрируется на рисунке ниже на примере одной из веб-страниц Wikipedia.



Dynamic content [edit]

The second type is a website with dynamic content displayed in plain view. Variable content is displayed dynamically on the fly based on certain criteria, usually by retrieving content stored in a database.

A website with dynamic content refers to how its messages, text, images and other information are displayed on the web page, and more specifically how its content changes at any given moment. The web page content varies based on certain criteria, either pre-defined rules or variable user input. For example, a website with a database of news articles can use a pre-defined rule which tells it to display all news articles for today's date. This type of dynamic website will automatically show the most current news articles on any given date. Another example of dynamic content is when a retail website with a database of media products allows a user to input a search request for the keyword Beatles. In response, the content of the web page will spontaneously change the way it looked before, and will then display a list of Beatles products like CDs, DVDs and books.

Software systems [edit]

There is a wide range of software systems, such as ANSI C servlets, JavaServer Pages (JSP), the PHP, Perl, Python, and Ruby programming languages, ASP.NET, Active Server Pages (ASP), YUMA and ColdFusion (CFML) that are available to generate dynamic web systems and dynamic sites. Sites may also include content that is retrieved from one or more databases or by using XML-based technologies such as RSS.

Static content may also be dynamically generated either periodically, or if certain conditions for regeneration occur (cached) in order to avoid the performance loss of initiating the dynamic engine on a per-user or per-connection basis.

Plug ins are available to expand the features and abilities of web browsers to show *active content* or even create rich Internet applications. Examples of such plug-ins are Microsoft Silverlight, Adobe Flash, Adobe Shockwave or applets written in Java. Dynamic HTML also provides for user interactivity and realtime element updating within web pages (i.e., pages don't have to be loaded or reloaded to effect any changes), mainly using the Document Object Model (DOM) and JavaScript, support which is built-in to most modern web browsers.

Turning a website into an income source is a common practice for web developers and website owners. There are several methods for creating a website business which fall into two broad categories, as defined below.

Рис. 2. Веб-страница сайта «Википедия».

На многих сайтах с этой проблемой справляются с помощью какой-либо фиксированной компоновки, когда содержимое помещается в узкие колонки. (В WPF это тоже можно сделать, поместив содержимое в колонку в контейнере Grid и задав свойство ColumnDefinition.MaxWidth) Это действительно повышает удобство чтения содержимого, но в больших окнах появляется пустое место.

Для содержимого потокового документа в WPF добавлены усовершенствования этих современных подходов: лучший механизм разбиения на страницы, вывод в несколько колонок, более разумные алгоритмы переноса слов и обтекания текста, а также предпочтительные режимы отображения, выбираемые пользователем. В результате пользователю предоставляется более удобная среда для чтения больших объемов содержимого.

Потоковые документы в WPF представлены классом FlowDocument, который может включать в себя различные 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 (рис. 3):

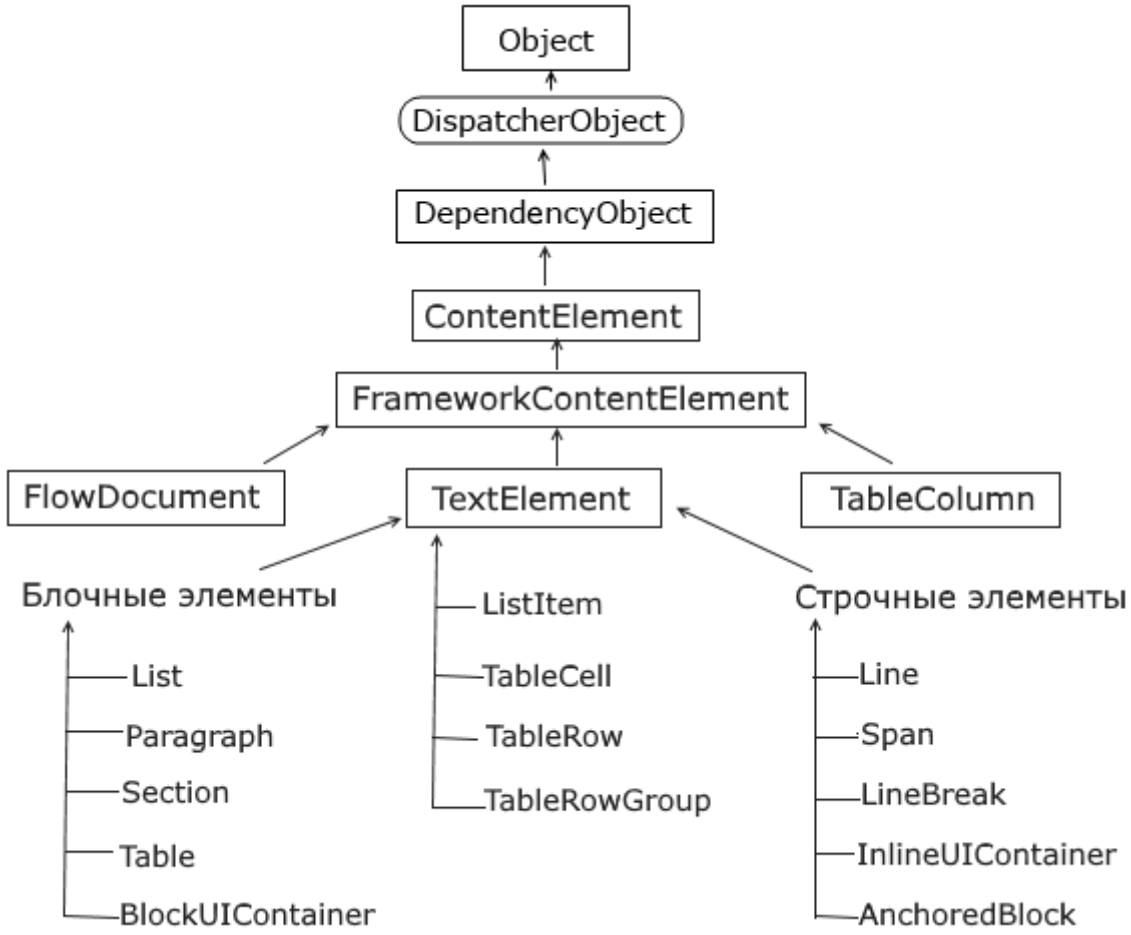


Рис. 3. Иерархия потоковых документов.

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>
```

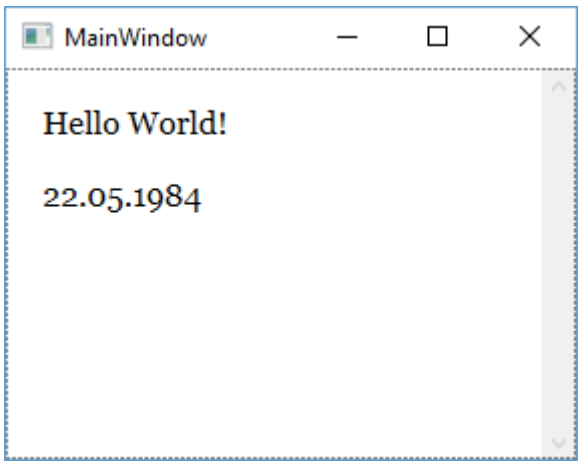


Рис. 4. Результат работы программы.

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, BackUIContainer 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>
```

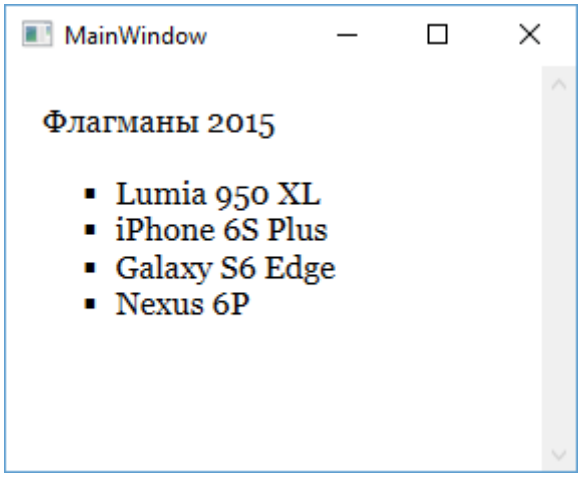


Рис. 5. Результат работы программы.

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>
```

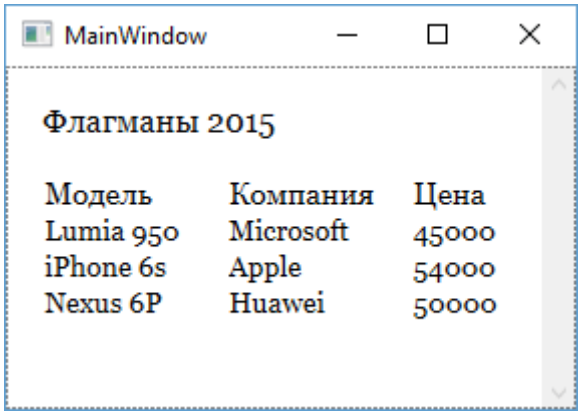


Рис. 6. Результат работы программы.

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 однообразной 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>
```

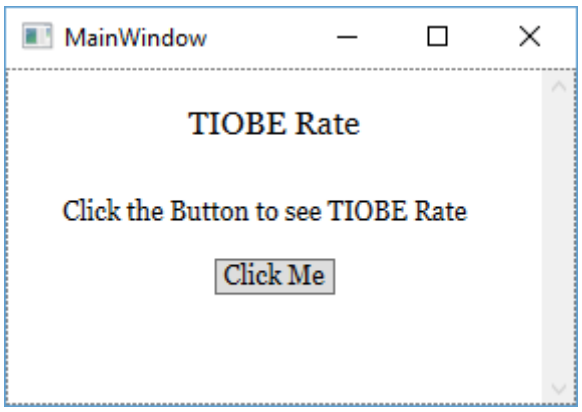


Рис. 7. Результат работы программы.

Строчные элементы

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

- Элемент 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>
```

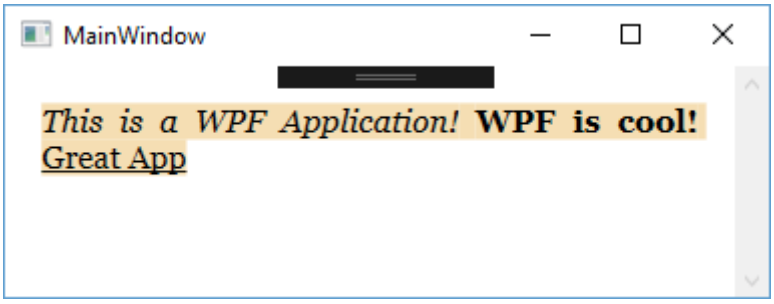


Рис. 8. Результат работы программы.

- Элемент Hyperlink

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

Элемент LineBreak задаёт перевод строки:

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

- Элемент InlineUIContainer

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="Italic">
                <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 зная, не узнавая его...
        </Paragraph>
    </FlowDocument>
```

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

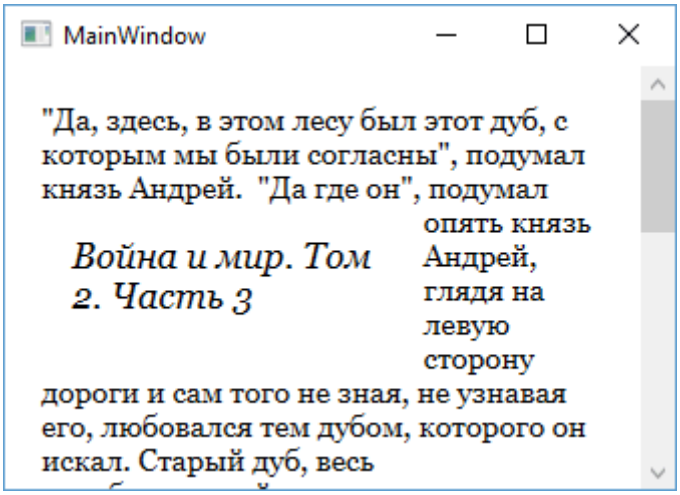


Рис. 9. Результат работы программы.

- Элемент 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="Italic">
                <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 зная...
        </Paragraph>
    </FlowDocument>
```

С помощью свойства HorizontalAnchor мы можем управлять позиционированием элемента по горизонтали. Так, значение ContentLeft позволяет выровнять текст по левой стороне, ContentRight - по правой стороне, а значение ContentCenter - по центру.

Другое свойство VerticalAnchor позволяет выровнять содержимое Figure по вертикали.

Контейнеры потоковых документов

- 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 Андрей...
            </Paragraph>
        </FlowDocument>
    </FlowDocumentScrollViewer>
```

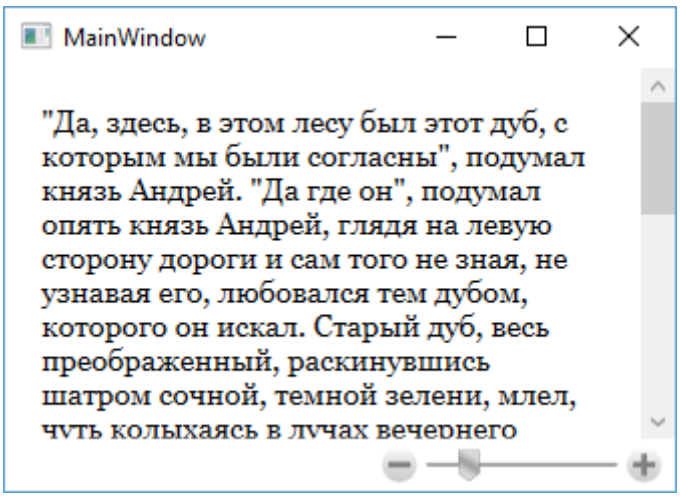


Рис. 10. Контейнер FlowDocumentScrollViewer.

- FlowDocumentPageViewer

  FlowDocumentPageViewer разбивает документ на страницы:

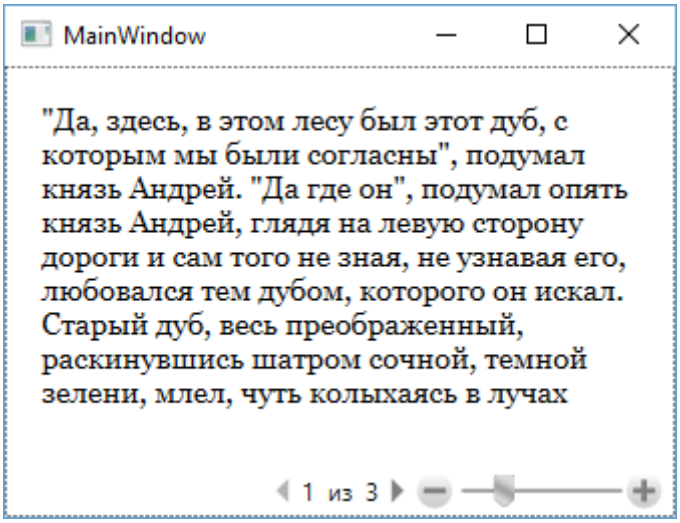


Рис. 11. Контейнер FlowDocumentPageViewer.

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 между столбцами 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 Андрей...
        </Paragraph>
    </FlowDocument>
</FlowDocumentPageViewer>
```

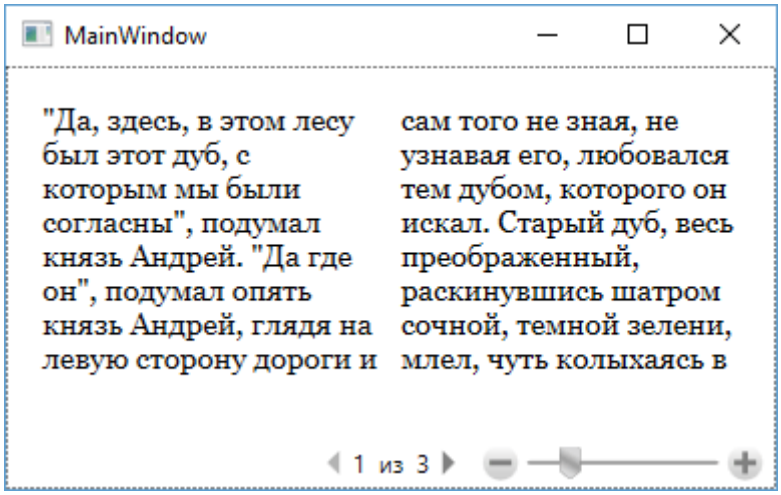


Рис. 12. Контейнер FlowDocumentPageViewer.

- FlowDocumentReader

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 Андрей...
        </Paragraph>
    </FlowDocument>
</FlowDocumentReader>
```

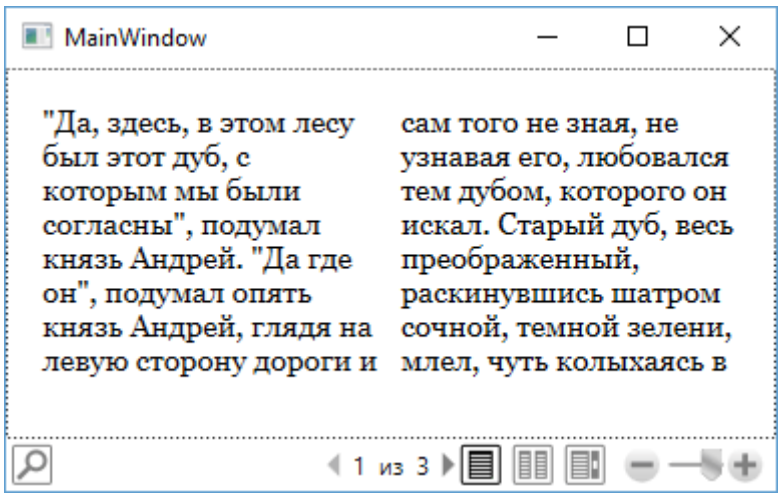


Рис. 13. Контейнер FlowDocumentReader.

Создание фиксированных документов

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:

```
<DocumentViewer x:Name="documentViewer">
    <FixedDocument>
        <PageContent>
            <FixedPage>
                <Grid Margin="10" Width="450" Height="600">
                    <TextBlock Text="Hello World!" />
                    <Ellipse Stroke="Green" Width="50" Height="50" Fill="Red" />
                </Grid>
            </FixedPage>
        </PageContent>
        <PageContent Width="650" Height="900">
            <FixedPage>
                <StackPanel Margin="10" Width="650" Height="900"
Background="LightBlue">
                    <TextBlock Text="Page 2" />
                </StackPanel>
            </FixedPage>
        </PageContent>
    </FixedDocument>
</DocumentViewer>
```

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:

```
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:

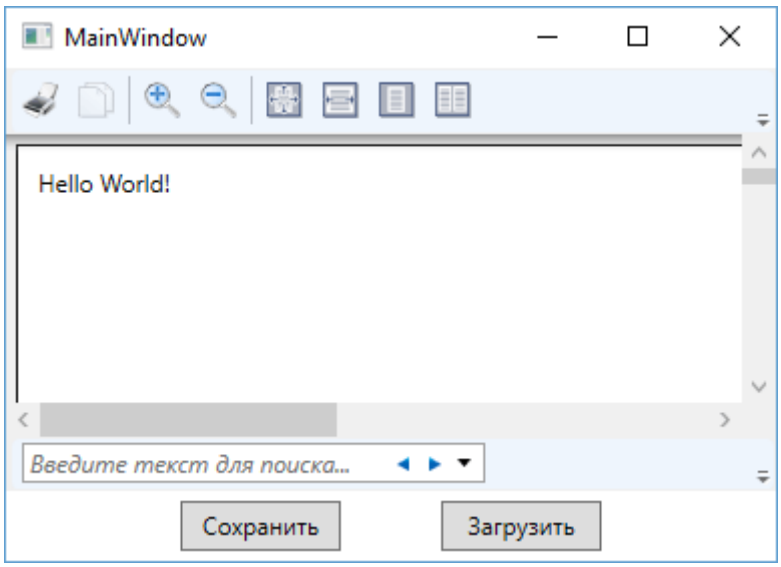


Рис. 14. Фиксированный документ FixedDocument.

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:

```
<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
Наполеоном,
                    и в высшем свете много говорили о важности этого события.
                    В 1809 году близость двух «властелинов мира», как называли Александра
и Наполеона,
                    дошла до того, что когда Наполеон объявил войну Австрии, русский
корпус выступил за границу,
                    чтобы сражаться на стороне бывшего противника против бывшего
союзника, австрийского императора.
            </Paragraph>
        </FlowDocument>
    </FlowDocumentScrollViewer>
```

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:

```
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:

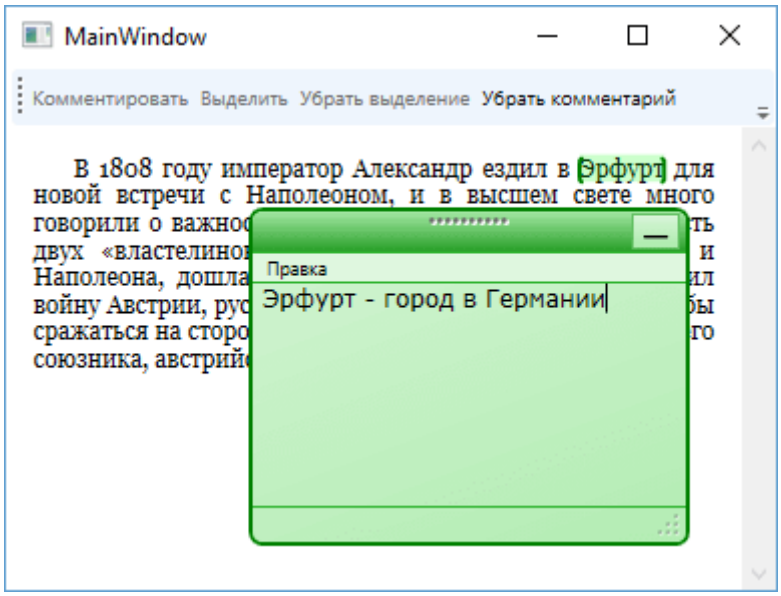


Рис. 15. Создание аннотации.

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

Содержательная постановка задачи

В среде программирования Visual Studio 2015 (или выше) с помощью системы для построения клиентских приложений WPF на языке программирования C# написать программу, которая будет содержать средства для работы с потоковыми и фиксированными документами.

При работе с потоковыми документами продемонстрировать работу со следующими элементами:

- **FlowDocumentReader**
- **FlowDocumentPageViewer**
- **FlowDocumentScrollViewer**

Кроме этого, продемонстрировать работу со следующими блочными элементами:

- Элемент **Paragraph**
- Элемент **List**
- Элемент **Table**
- Элемент **Section**
- Элемент **BlockUIContainer**

Также продемонстрировать работу со следующими строчными элементами:

- Элемент **Span**
- Элемент **Hyperlink**
- Элемент **LineBreak**
- Элемент **InlineUIContainer**
- Элемент **Floater**
- Элемент **Figure**

С помощью элемента **RichTextBox** создать инструмент для редактирования текста.

Также создать набор инструментов для загрузки документов с файла с различными расширениями и набор инструментов для сохранения файла также с различными расширениями.

При работе с фиксированными документами продемонстрировать работу со следующими элементами:

- **DocumentViewer**

Демонстрация работы программы

При запуске приложения открывается начальное окно, в котором пользователю предлагается выбрать один из пяти предложенных контейнеров для работы с документами (рис. 16).

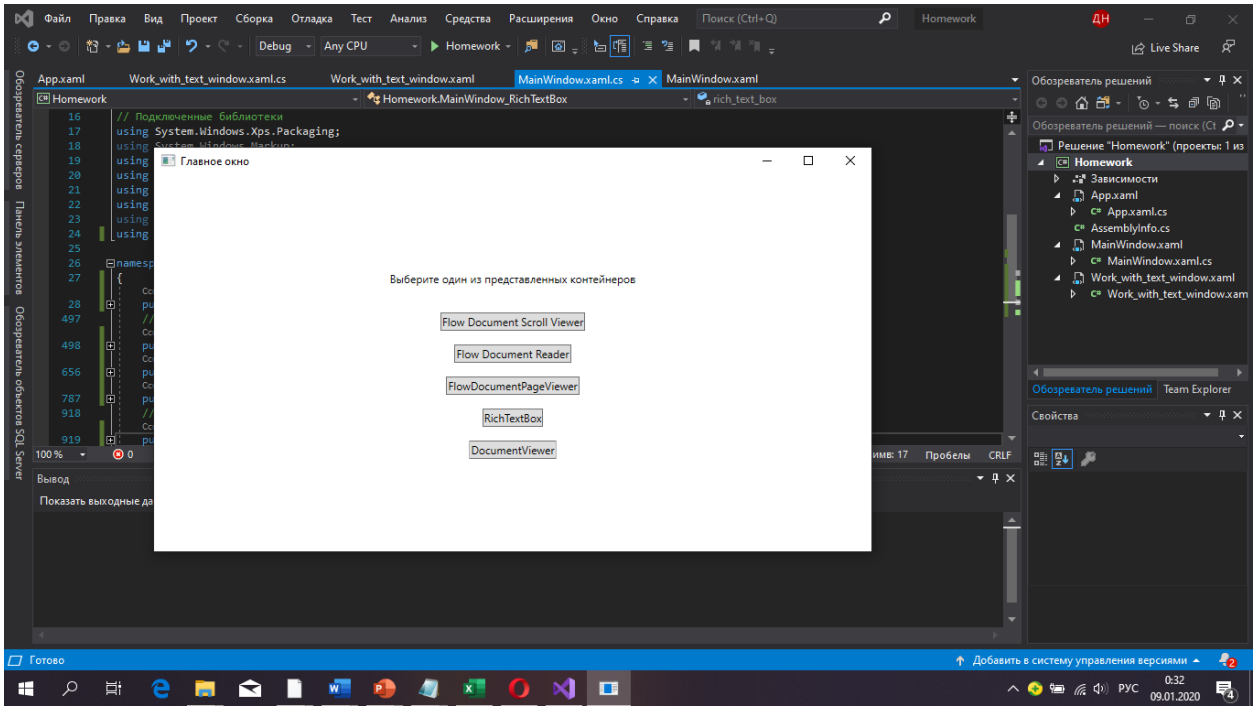


Рис. 16. Главное окно программы.

Инструменты для каждого из представленных выше контейнеров практически идентичны, поэтому рассмотрим только работу программы для контейнера Flow Document Scroll Viewer. При нажатии на данную кнопку открывается новое окно (рис. 17), в котором пользователь может либо открыть существующий файл, либо посмотреть примеры с блочными и строчными элементами.

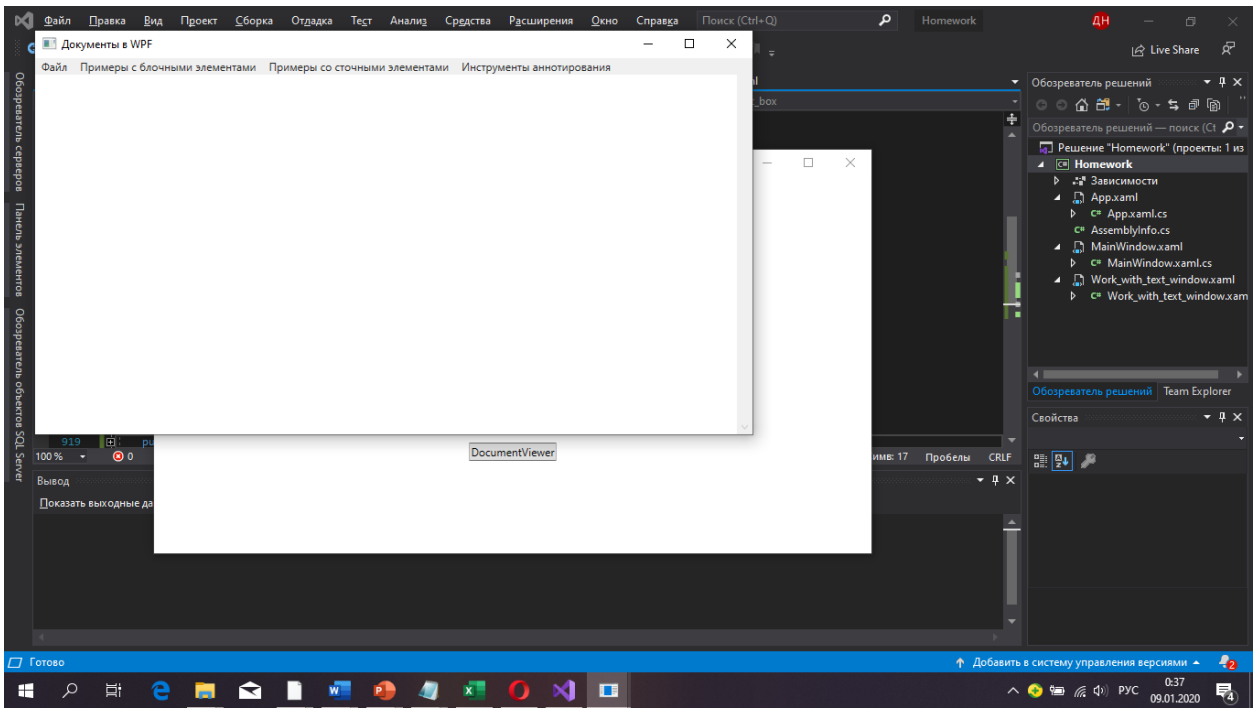


Рис. 17. Окно для работы с контейнером Flow Document Scroll Viewer.

Продемонстрируем работу одного из примеров. Пример со строчным элементом Span будет выглядеть следующим образом:

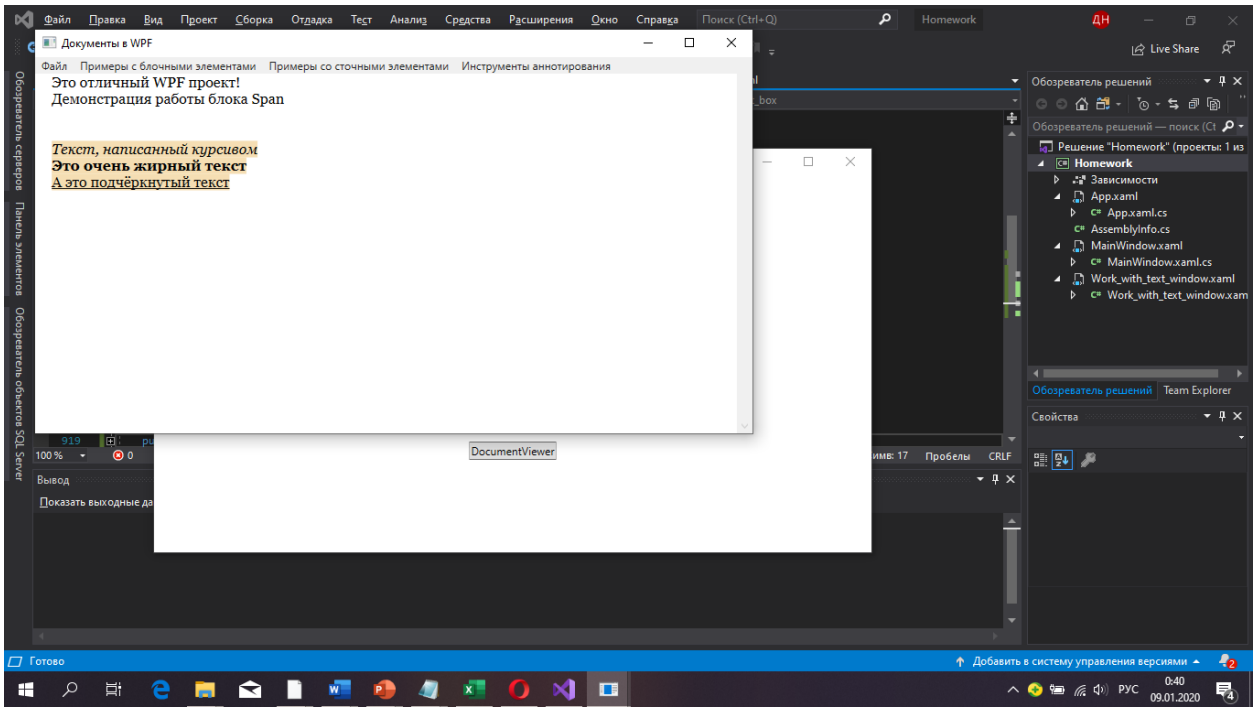


Рис. 18. Пример работы со строчным элементом Span.

На рис. 19 показан результат открытия текстового файла.

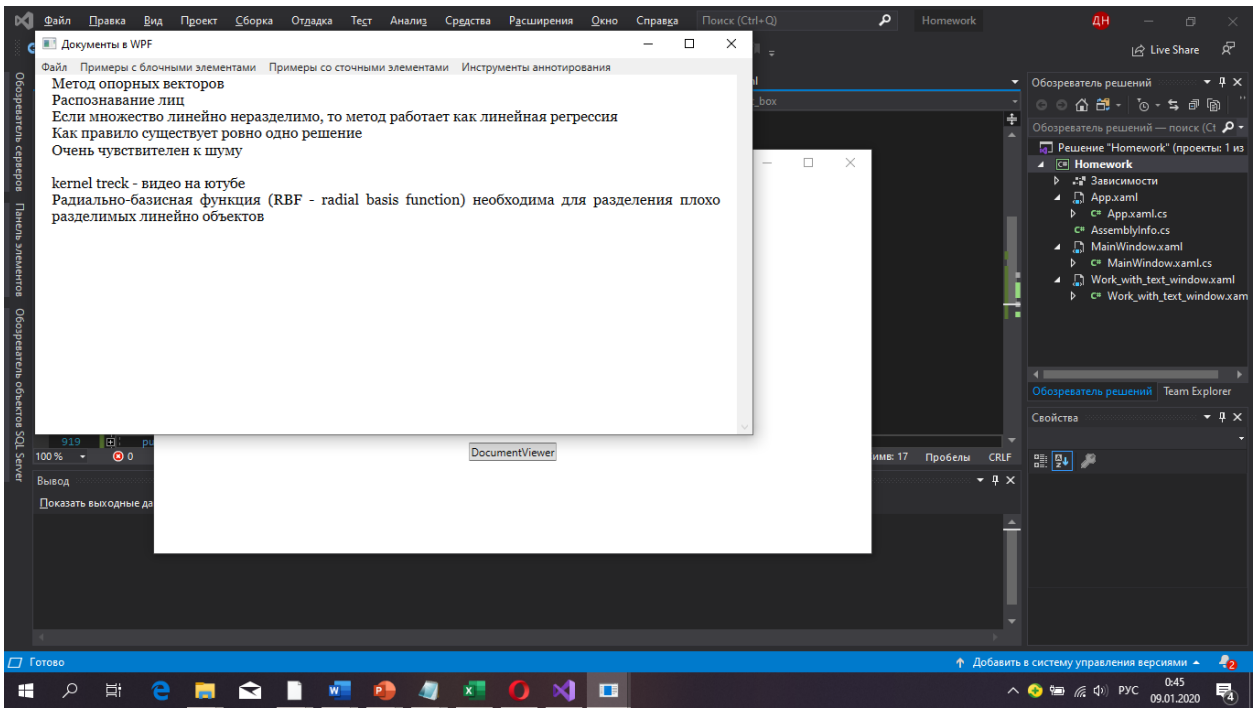


Рис. 19. Результат открытия текстового файла.

На рис. 20 продемонстрирована работа инструментов аннотирования.

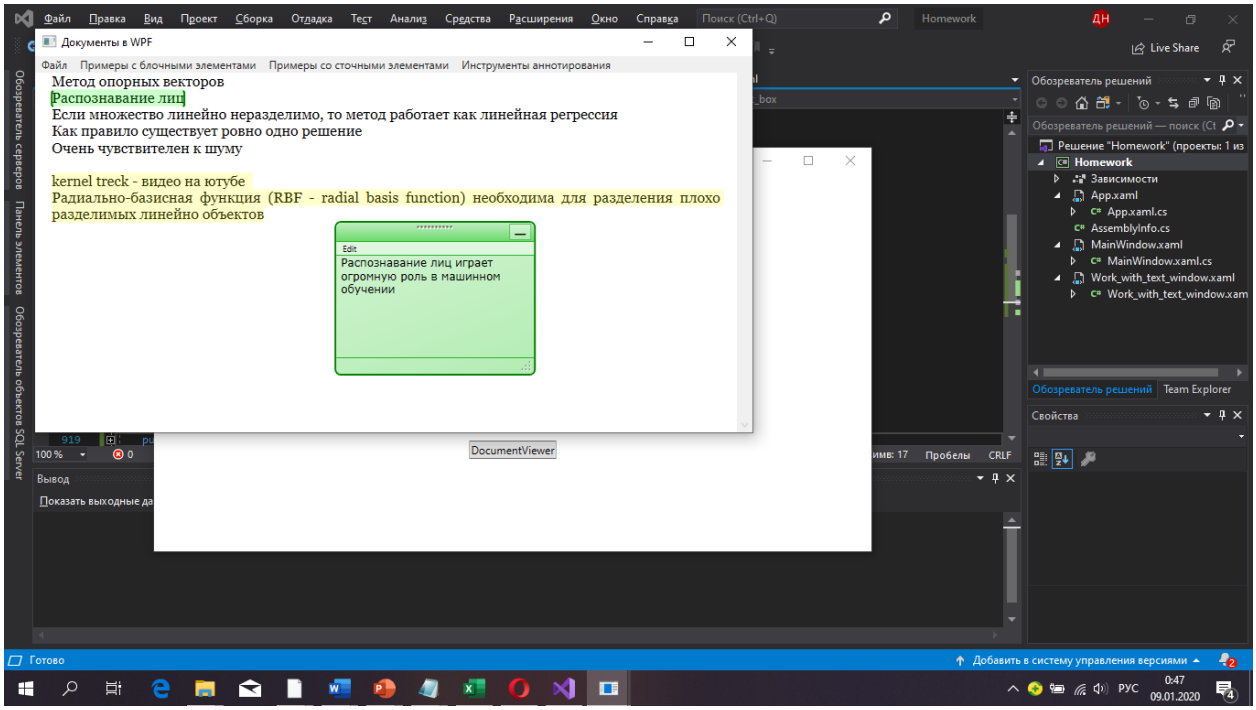


Рис. 20. Работа с инструментами аннотирования.

Выводы

В ходе данного домашнего задания был проведён анализ литературы на тему «Документы в WPF». В ходе данного анализа были найдены средства метода для работы с документами и, исходя из цели данного домашнего задания, которая заключалась в демонстрации основных элементов для работы с документов, были выбраны необходимые инструменты: 5 основных контейнеров, демонстрирующие работу с поточными и фиксированными документами, 5 основных блочных элементов и 6 основных строчных элементов. Также были выбраны для демонстрации элементы аннотирования и инструменты для работы с файлами. В ходе данного домашнего задания данные инструменты были реализованы в приложении, а также продемонстрирована их работа. Домашнее задание выполнено.

Список используемой литературы

- https://technet.microsoft.com/ru-ru/microsoft-edge/ms748388(v=vs.71) – Типы документов в WPF.
- https://metanit.com/sharp/wpf/15.php – Документы в WPF.
- https://professorweb.ru/my/WPF/documents_WPF/level28/28_2.php – Создание элементов в потоковых документах
- http://www.nookery.ru/hyperlink-wpf/ – Работа с элементом Hyperlink