К ЛР №32
«Высокоуровневые средства WPF для работы с документами»

На самостоятельную работу выносится освоение высокоуровневых средств WPF для работы с документами.

3.4.1 Для отображения больших объёмов текста (газетная статья, подробные оперативные справки) WPF предлагает набор высокоуровневых средств, которые работают с **документами**. Эти средства позволяют отображать большие объёмы содержимого в легко читаемом виде независимо от размеров окна, в котором они находятся.

Документы в WPF делятся на две крупных категории:

- документы нефиксированного формата (*flow document*);
- документы фиксированного формата (*fixed document*).

Документы **нефиксированного** формата предназначены для чтения на экране. Содержимое документа организовано в виде потока документа и изменяется в процессе изменения размера окна, поэтому часто эти документы называют **потоковыми**.

Потоковые документы - это документы, предназначенные для просмотра на экране монитора. Как и фиксированные документы, потоковые документы поддерживают расширенную компоновку. WPF может оптимизировать потоковый документ на основе указанного способа просмотра документа: динамически компоновать содержимое на основе такой информации, как размеры окна просмотра, разрешение экрана и т.д. В принципе потоковые документы в основном используются для тех же целей, что и документы HTML, но обладают более совершенными возможностями компоновки текста.

Потоковые документы более важны для создания приложений, а фиксированные - для создания документов, которые нужно распечатывать без изменений (например, публикаций).

Документы **фиксированного формата** в основном ориентированы на печать и страничное представление, при этом их содержимое всегда организовано одинаковым образом. Расположение всего содержимого фиксировано (например, нельзя изменить способ разрыва строк и переноса слов). Фиксированные документы можно читать с монитора, но в первую очередь они предназначены для печати на принтере. В WPF имеется один тип фиксированных документов, в котором используется стандарт Microsoft XPS (*XML Paper Specification* – бумажная спецификация XML) для печатных документов.

Для создания документов нефиксированного и фиксированного формата используется основное пространство имён `System.Windows.Documents`. Это пространство имён содержит базовые классы, с помощью которых можно создавать похожие на Word документы нефиксированного формата, а также документы фиксированного формата в режиме полного соответствия WYSIWIG. Используются также дополнительные просторанства имён - `System.Windows.Xps`, `System.Printing`, `System.Windows.Annotations`, `System.IO.Packaging` и другие, организованные в .NET сборках `ReachFramework` и `System.Print`, которые необходимо включать в состав проекта.

На рисунке 3.25 показана иерархия наследования элементов вывода содержимого.

WPF обеспечивает поддержку обоих видов документов посредством контейнеров.

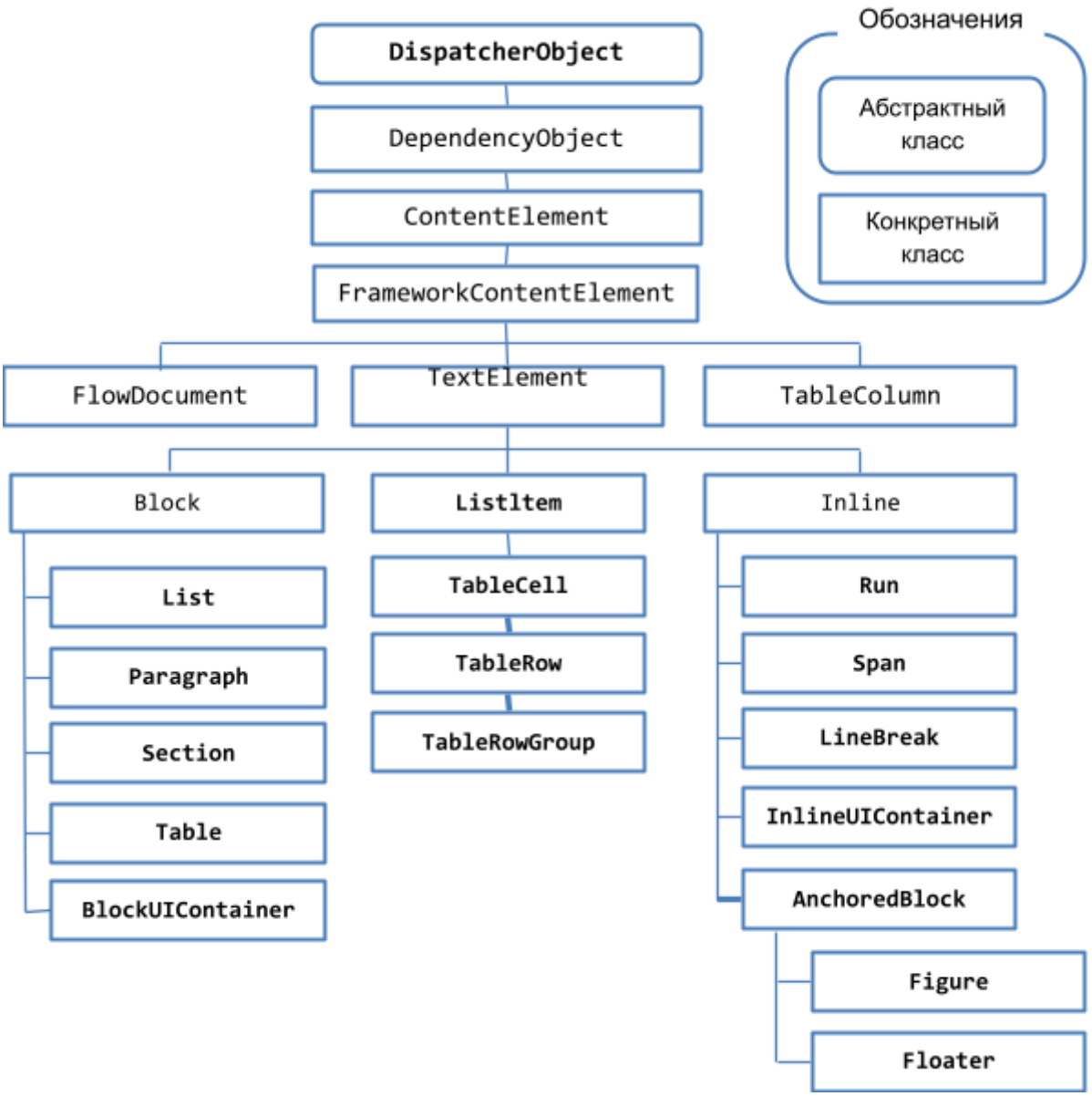

Рисунок 3.25

WPF предоставляет три контейнера, **доступных только для чтения**, которые можно использовать для отображения потоковых документов:

- <FlowDocumentScrollViewer>;
- <FlowDocumentPageViewer>;
- <FlowDocumentReader>.

<FlowDocumentScrollViewer> показывает весь документ с полосой прокрутки, которая позволяет перемещаться по документу, если он превышает размеры контейнера. Этот контейнер не поддерживает разбиение на страницы или несколько колонок, но, как и все контейнеры, поддерживает печать и изменение масштаба.

<FlowDocumentPageViewer> разбивает документ на страницы. Размер каждой страницы совпадает с доступным местом, а пользователь может переходить от одной страницы к другой. Контейнер <FlowDocumentPageViewer> требует больших затрат, чем <FlowDocumentScrollViewer> из-за дополнительных вычислений для разбивки содержимого на страницы.

<FlowDocumentReader> сочетает возможности контейнеров

<FlowDocumentScrollViewer> и <FlowDocumentPageViewer>. Он позволяет пользователю выбрать, как будет выполняться чтение содержимого - с прокруткой или постранично. Кроме того, в нем есть функция поиска. Контейнер <FlowDocumentReader> требует больших затрат, чем другие контейнеры потоковых документов. Для переключения с одного контейнера на другой нужно просто изменить объемлющий дескриптор. Каждый из этих контейнеров имеет дополнительные возможности: изменение масштаба, разбиение на страницы и печать.

Элементы-наследники DocumentViewer позволяют выводить фиксированные документы в окне WPF, а <FlowDocumentReader>, <FlowDocumentPageViewer> и <FlowDocumentScrollViewer> предоставляют разные способы отображения потоковых документов.

3.4.2 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 ещё хуже, широкоформатного монитора, получатся длинные неудобочитаемые строки.

Потоковый документ WPF создаётся с помощью сочетания потоковых элементов, которые формируют отдельную ветвь классов, порождённых от ContentElement и FrameworkContentElement.

Классы элементов вывода содержимого поддерживают набор базовых событий (включая события клавиатуры и мыши), операции перетаскивания, отображение всплывающих подсказок и инициализацию. Ключевым отличием элементов вывода содержимого от элементов, не связанных с содержимым, является то, что первые не выполняют свою прорисовку. Для этого им необходим контейнер, который сможет отобразить все элементы вывода содержимого. Такая отложенная прорисовка позволяет контейнеру выполнять разнообразную оптимизацию. Например, контейнер может выбрать наиболее подходящий способ разрыва строк текста в абзаце, даже если этот абзац является единственным элементом.

Элементы вывода содержимого делятся на две категории:

- блочные элементы;
- строковые элементы.

Блочные элементы могут применяться для группирования других элементов вывода содержимого. Например, <Paragraph> (абзац) является блочным элементом. Он может хранить текст разного формата. Каждый раздел текста с отличным форматом является отдельным элементом в абзаце.

Строковые элементы находятся внутри блочных элементов (или других строковых элементов). Например, элемент <Run> (фрагмент) содержит текст, который затем может быть вложен в элемент <Paragraph>.

Модель содержимого позволяет строить несколько уровней вложения. К примеру, внутрь элемента <Underline> можно поместить элемент <Bold>, чтобы создать жирный текст. Аналогично можно создать элемент <Section>, а внутри него несколько элементов <Paragraph>, каждый из которых содержит разнообразные строковые элементы с текстовым содержимым. Все эти элементы определены в пространстве имён System.Windows.Documents.

Однако в составе WPF имеются API для программного создания фиксированных документов, а класс RichTextBox позволяет пользователям редактировать потоковое содержимое.

3.4.2.1 Пример использования **блочных** и **строковых** элементов для создания простого потокового документа, который содержит абзацы текста, списки, рисунки и таблицу.

Среда Visual Studio позволяет либо создать новый потоковый документ как отдельный файл, либо определить его внутри существующего окна, используя один из поддерживаемых контейнеров.

В примере для создания потокового документа будет использован контейнер `<FlowDocumentScrollViewer>`.

Первоначальный вид разметки показан ниже:

```
<Window x:Class="WpfApplication1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
        <FlowDocumentScrollViewer>
            <FlowDocument>
            </FlowDocument>
        </FlowDocumentScrollViewer>
</Window>
```

Обычно элемент `<FlowDocumentScrollViewer>` позволяет выделять текст (как в web-браузере). Пользователь может копировать части документа в буфер обмена Windows и вставлять их в другие приложения. Чтобы содержимое было доступно только для чтения, необходимо задать свойство `IsSelectionEnabled="False"`.

Высокоуровневый потоковый документ создаётся с помощью элементов блочного уровня: `<Paragraph>`, `<Run>`, `<List>`, `<Table>` и `<Section>`.

Элемент `<Paragraph>` может выступать элементом верхнего уровня, на количество элементов верхнего уровня нет ограничений, А в случае необходимости автоматически добавляется линейка прокрутки. Шрифт по умолчанию (`FontFamily="Segoe UI"`) выбирается из системных настроек Windows, а не из вмещающего окна. `<Paragraph>` содержит свойство `TextIndent`, которое позволяет задавать величину отступа первой строки в не зависящих от устройства единицах. (По умолчанию оно имеет нулевое значение.)

Элемент `<Run>` содержит обычный текст, допускает применение форматирования, хотя обычно для этого используется элемент `<Span>`. Элементы `<Run>` часто создаются неявно (например, при добавлении текста в абзац).

Элемент `<List>` (список) представляет маркированный или нумерованный список. Формат списка определяется с помощью свойства `MarkerStyle`. Свойство `MarkerOffset` позволяет задать расстояние между каждым элементом списка и его маркером.

Элемент `<Table>` (таблица) предназначен для отображения табличной информации. Он был создан по подобию элемента `<table>` из языка HTML. Чтобы создать таблицу, необходимо выполнить следующие действия.

Действие 1. Поместить в `<Table>` элемент `<TableRowGroup>`. Элемент `<TableRowGroup>` хранит группу строк, и каждая таблица состоит из одного или более элементов `<TableRowGroup>`. Если используется несколько групп с разным форматированием, то можно изменить общий внешний вид таблицы, не занимаясь форматированием каждой строки.

Действие 2. В элемент `<TableRowGroup>` поместить элементы `<TableRow>` для каждой строки.

Действие 3. В каждый элемент `<TableRow>` поместить элементы `<TableCell>`, представляющие элементы строки.

Действие 4. В каждый элемент <TableCell> поместить какой-либо блочный элемент (как правило, <Paragraph>). В этот блочный элемент можно занести содержимое данной ячейки.

Если не указать явным образом ширину столбцов, WPF равномерно распределит пространство между всеми столбцами. Это поведение можно изменить, присвоив свойству <Table.Columns> набор объектов <TableColumn> и определив для каждого из них ширину Width.

Элемент <Section> (раздел) не имеет собственного встроенного форматирования. Он используется для упаковки других блочных элементов и позволяет применить единый формат ко всей порции документа. Например, если нужно, чтобы несколько смежных абзацев имели одинаковый цвет фона и шрифт, можно упаковать эти абзацы в один раздел и задать свойство Background. Элемент <Section> аналогичен элементу в языке HTML.

Элемент <BlockUIContainer> позволяет помещать элементы, не связанные с содержимым (классы-наследники UIElement), в документ, где должен помещаться блочный элемент. С помощью элемента <BlockUIContainer> можно добавить в документ кнопки, флажки и даже целые контейнеры компоновки наподобие <StackPanel> и <Grid>. Единственное ограничение - <BlockUIContainer> может иметь только один дочерний элемент.

WPF предлагает большой набор строковых элементов, которые можно помещать в блочные или другие строковые элементы. Классы строковых элементов вывода содержимого перечислены в таблице 3.7.

Таблица 3.7

Имя	Описание
Run	Содержит обычный текст. Допускает применение форматирования, часто создаётся неявно (например, при добавлении текста в абзац).
Span	Объединяет любое количество других строковых элементов при форматировании части текста. Можно просто поместить текст в элемент <Span>, а вложенный элемент <Run> будет создан автоматически. Элемент <Span> позволяет найти и обработать конкретный фрагмент текста. Элемент <Span> аналогичен элементу в языке HTML
Bold, Italic и Underline	Применяют соответственно полужирное, курсивное и подчёркнутое форматирование. Эти элементы порождены от <Span>. Но лучше упаковать формируемый текст в элемент <Span>, а затем с помощью свойства Style элемента <Span> указать стиль с нужным форматом. В этом случае можно будет легко изменять характеристики форматирования, не меняя разметку документа.
Hyperlink	Представляет ссылку, реагирующую на щелчок мыши, внутри потокового документа. В оконном приложении в ответ на событие Click можно выполнить действие (например, вывести другой документ). В страничном приложении можно использовать свойство NavigateUri, чтобы пользователь мог переходить к произвольной странице.
LineBreak	Добавляет разрыв строки в блочный элемент. Иногда лучше использовать большие значение Margin или Padding, чтобы увеличить промежутки между элементами.
InlineUIContainer	Позволяет помещать элементы, не связанные с содержимым (наследники UIElement), туда, где должны быть строковые элементы (например, в элемент <Paragraph>). Класс InlineUIContainer похож на класс BlockUIElement, но является строковым элементом, а не блочным.
Floater и Figure	Позволяют внедрять всплывающее окошко, в котором можно вывести важную информацию, рисунок или связанное содержимое (например, рекламные сообщения, ссылки, кодовые фрагменты и т.п.).

В реальных приложениях существует много типов документов, которые как-то должны взаимодействовать с пользователем (помимо элемента вывода содержимого <Hyperlink>). Например, если система потоковой компоновки использована для создания страниц оперативной справки, то можно добавить кнопку, запускающую некоторое действие.

В WPF добавлен механизм разбиения содержимого на страницы, его вывод в несколько колонок, алгоритмы переноса слов и обтекания текста, а также предпочтительные режимы отображения, выбираемые пользователем.

Ниже приведённая последовательность шагов обеспечивает создание потокового документа.

Шаг 1. Создать новый проект WPF Application.

Шаг 2. Изменить XAML-описание окна MainWindow.xaml на код, приведённый ниже.

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <FlowDocumentScrollViewer >
        <FlowDocument>
            <Paragraph TextAlignment="Center">
                <Run FontWeight="ExtraBold">Документы</Run>
            </Paragraph>
            <Paragraph TextIndent="20">Потоковый документ создаётся с
помощью класса <Bold>FlowDocument</Bold>. Visual Studio позволяет либо
создать новый потоковый документ как отдельный файл, либо определить его
внутри существующего окна, используя один из поддерживаемых контейнеров.
                <Run xml:space="preserve"> </Run>
                <Floater FontFamily="Lucida Calligraphy" Width="200"
HorizontalAlignment="Center" FontStyle="Italic" Foreground="Yellow">
                    <Floater.Background>
                        <RadialGradientBrush>
                            <GradientStop Color="Black" Offset="0" />
                            <GradientStop Color="White" Offset="1" />
                        </RadialGradientBrush>
                    </Floater.Background>
                    <Paragraph TextAlignment="Center">"Высокоуровневые
средства WPF для работы с документами"</Paragraph>
                </Floater>
            </Paragraph>
            <Paragraph TextIndent="20">Элементы вывода содержимого делятся
на две категории:</Paragraph>
            <List>
            <ListItem>
                    <Paragraph>блочные элементы;</Paragraph>
            </ListItem>
            <ListItem>
                    <Paragraph>
 <Hyperlink Click="LinkClicked"> строковые </Hyperlink> элементы.
                    </Paragraph>
            </ListItem>
        </List>
            <Paragraph TextIndent="20">Задачи в виде нумерованного
```

```
списка:</Paragraph>
            <List MarkerStyle="Decimal" MarkerOffset="10">
                <ListItem>
                    <Paragraph>изучить создание WPF потоковых
документов;</Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>научиться программировать блочные и
строковые элементы управления содержимым;</Paragraph>
                </ListItem>
                <ListItem>
                    <Paragraph>освоить модель обработки
событий.</Paragraph>
                </ListItem>
            </List>
            <Paragraph TextIndent="20">
                <Floater Width="70" Padding="5,0,5,0"
HorizontalAlignment="Left">
                    <BlockUIContainer>
                        <Image Source="d:\superbar.png"
Stretch="Uniform"></Image> </BlockUIContainer>
                </Floater>
                Элемент
                <Bold>Floater</Bold> можно применять и для вывода
рисунков. При этом элемент <Bold>Image</Bold> используется совместно с
элементом
                <Bold>BlockUIContainer</Bold> или
<Bold>InlineUIContainer</Bold>.
                <Figure Width="80" Padding="5,0,5,0"
HorizontalAnchor="ContentCenter">
                    <BlockUIContainer>
                        <Image Source="d:\ii.jpg"
Stretch="Uniform"></Image>
                    </BlockUIContainer>
                </Figure>
                При вставке элемента <Bold>Floater</Bold>, заключающего в
себе изображение, потоковый документ предполагает, что рисунок должен быть
по ширине таким же, как и вся колонка текста. Размер находящегося внутри
элемента <Bold>Image</Bold> будет изменён, что может привести к
возникновению проблем при работе с растровыми рисунками. С помощью
свойства <Bold>Image.Stretch</Bold> можно запретить изменение размеров
изображения, хотя в этом случае плавающее окошко все равно займёт всю
колонку по ширине – просто вокруг рисунка останутся пустые места.
            </Paragraph>
            <Paragraph TextAlignment="Center">Таблица значений из
перечисления TextMarkerStyle</Paragraph>
            <Table>
                <Table.Columns>
                    <TableColumn Width="*"></TableColumn>
                    <TableColumn Width="*"></TableColumn>
                    <TableColumn Width="3*"></TableColumn>
                </Table.Columns>

                <TableRowGroup Paragraph.TextAlignment="Justify">
                    <TableRow FontWeight="Bold" >
```

```
                        <TableCell>
                            <Paragraph>№</Paragraph>
                        </TableCell>
                        <TableCell>
                            <Paragraph>Свойство</Paragraph>
                        </TableCell>
                        <TableCell>
                            <Paragraph>Описание</Paragraph>
                        </TableCell>
                    </TableRow>
                    <TableRow>
                        <TableCell>
                            <Paragraph>1</Paragraph>
                        </TableCell>
                        <TableCell>
                            <Paragraph>Disc</Paragraph>
                        </TableCell>
                        <TableCell>
 <Paragraph >Сплошной кружок. Установлен по умолчанию</Paragraph>
                        </TableCell>
                    </TableRow>
                    <TableRow>
                        <TableCell>
                            <Paragraph>2</Paragraph>
                        </TableCell>
                        <TableCell>
                            <Paragraph>Box</Paragraph>
                        </TableCell>
                        <TableCell>
 <Paragraph>Сплошной квадратик</Paragraph>
                        </TableCell>
                    </TableRow>
                    <TableRow>
                        <TableCell>
                            <Paragraph>3</Paragraph>
                        </TableCell>
                        <TableCell>
                            <Paragraph>Circle</Paragraph>
                        </TableCell>
                        <TableCell>
 <Paragraph>Контурный кружок</Paragraph>
                        </TableCell>
                    </TableRow>
                </TableRowGroup>
            </Table>
            <Section FontFamily="Arial"  FontSize="12" Background="Bisque">
                <Paragraph
LineStackingStrategy="BlockLineHeight">Элементом верхнего уровня может
выступать элемент <Bold>Paragraph</Bold>.</Paragraph>
                <Paragraph LineStackingStrategy="BlockLineHeight">Линейка
прокрутки добавляется автоматически.</Paragraph>
            </Section>
            <BlockUIContainer>
                <Button HorizontalAlignment="Left" Padding="5">Открыть
диалоговое окно свойств</Button>
```

```
            </BlockUIContainer>
        </FlowDocument>
        </FlowDocumentScrollViewer>
</Window>
```

Шаг 3. Для события `Click="LinkClicked"` гиперссылки `<Hyperlink>` с помощью контекстного меню добавить в файл MainWindow.xaml.cs метод обработки с пустым кодом:

```
private void LinkClicked(object sender, RoutedEventArgs e)
        {
        }
```

Шаг 4. Запустить проект на выполнение и проверить работоспособность дескрипторной разметки документа.

Рисунок 3.26 иллюстрирует внешний вид потокового документа.

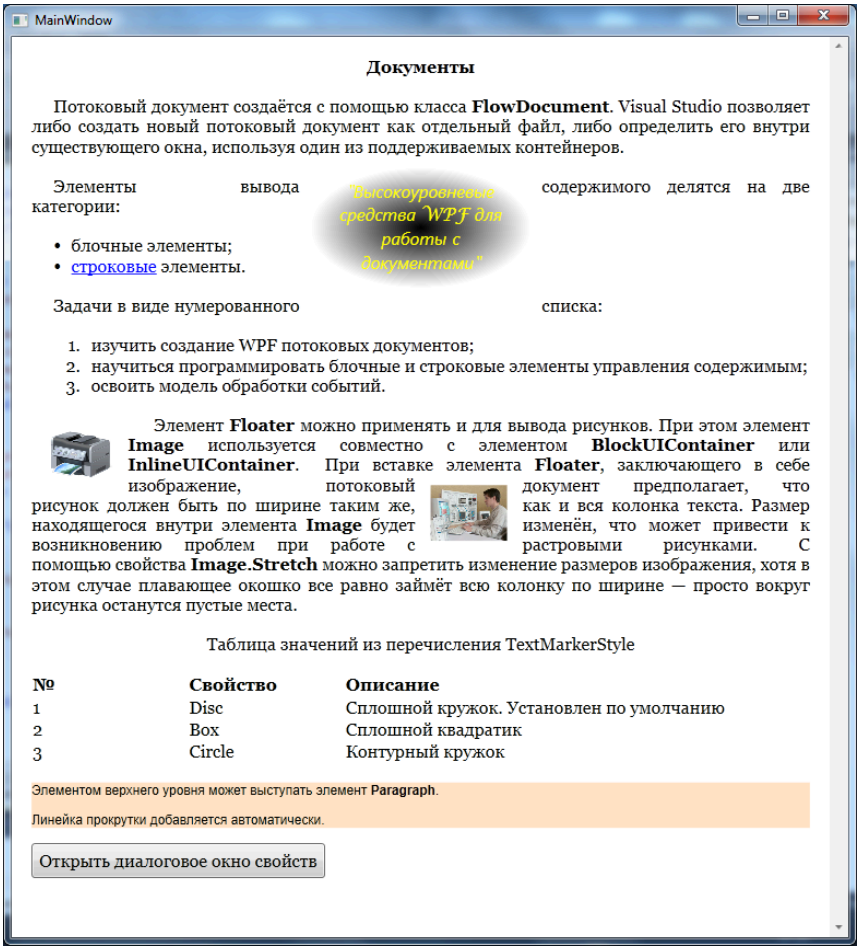

Рисунок 3.26

Далее рассмотрены основные элементы, используемые в примере.

Элемент `<Floater>` (плавающее окошко) позволяет выделить часть содержимого из главного документа. Для его создания необходимо поместить элемент `<Floater>` в другом блочном элементе (например, в абзаце как в примере). Сам элемент `<Floater>` также может содержать один или более блочных элементов.

По умолчанию плавающее окошко, используемое для элемента `<Floater>`, является невидимым, но для него можно задать текстурированный фон с помощью

свойства Background или рамку с помощью свойств BorderBrush и BorderThickness, чтобы четко отделить его содержимое от остальной части документа. Свойство Margin определяет промежуток между плавающим окошком и документом, и свойство Padding - промежуток между краями окошка и его содержимым.

Элемент <Floater> может использоваться для вывода рисунка с помощью элемента <Image>, который применяется вместе с элементом <BlockUIContainer> или <InlineUIContainer>.

При вставке плавающего окошка с <Image> потоковый документ предполагает, что рисунок должен быть по ширине таким же, как и вся колонка текста, поэтому размер элемента <Image> будет или сильно уменьшен или увеличен.Чтобы избежать этой проблемы, необходимо использовать свойство Stretch элемента <Image> и указание фиксированных размеров плавающего окошка.

Элемент <Figure> (рисунок) подобен <Floater>, но позволяет точнее управлять местоположением. При работе со сложным документом и форматированием лучше использовать <Figure>. Для жесткого позиционирования рисунка необходимо использовать такие свойства <Figure> как, HorizontalAnchor, VerticalOffset и HorizontalOffset, но они не поддерживаются контейнером <FlowDocumentScrollViewer>, используемым для отображения потокового документа. Для них необходим один или несколько специальных контейнеров потоковых документов, доступных только для чтения.

Поскольку XAML основан на языке XML, в нем действуют те же правила, поэтому если вставить в содержимое несколько пробелов подряд, они будут преобразованы в один пробел.

В разработанном документе помимо блочных использованы строковые элементы, такие как <Run>, <Hyperlink>, <Floater> и <Figure>, а также ссылка <Hyperlink Click="LinkClicked"> строковые </Hyperlink>.

3.4.2.2 Пример создания простого потокового документа с помощью элемента <FlowDocument>, размеченного на странице многостраничного окна. Элемент <FlowDocument> предоставляет пользователю дополнительные сервисные функции, такие как поиск по тексту, представление содержимого в две колонки и др.

Шаг 1. Добавить в состав проекта окно WPF. Заменить XAML-описание окна на следующий код:

```
<NavigationWindow x:Class="WpfApplication1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window1" Height="300" Width="300">
</NavigationWindow>
```

Шаг 2. В соответствии с функциональностью нового окна модифицировать программный код файла Window1.xaml.cs.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
// Добавить необходимое пространство имён
using System.Windows.Navigation;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
// Изменить тип класса Window1
    public partial class Window1 : NavigationWindow
    {
        public Window1()
        {
            InitializeComponent();
        }
    }
}
```

Шаг 3. Добавить в состав проекта страницу WPF с именем Page1.xaml и изменить XAML-описание на следующее:

```
<Page x:Class="WpfApplication1.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="300" d:DesignWidth="300" Title="Page1">
    <FlowDocument Name="document" IsOptimalParagraphEnabled="True">
        <Paragraph FontSize="20" FontWeight="Bold" TextAlignment="Center">
            Высокоуровневые средства WPF для работы с документами
        </Paragraph>
        <Paragraph TextIndent="20">  Для отображения больших объёмов
текста (например, газетной статьи или подробных оперативных справок) WPF
предлагает набор высокоуровневых средств, которые работают с документами.
Эти средства позволяют отображать большие объёмы содержимого так, чтобы их
было легко читать независимо от размеров окна, в котором они находятся. В
WPF добавлен механизм разбиения содержимого на страницы, его вывод в
несколько колонок, алгоритмы переноса слов и обтекания текста, а также
предпочтительные режимы отображения, выбираемые пользователем. Документы
в WPF делятся на две крупных категории:- фиксированные документы; -
потоковые документы.
        </Paragraph>
        <Paragraph TextIndent="20"> К категории
  <Run xml:space="preserve">фиксированных </Run> документов относятся
подготовленные для печати документы. Расположение всего содержимого
фиксировано (например, нельзя изменить способ разрыва строк и переноса
слов). Фиксированные документы можно читать с монитора, но в первую
очередь они предназначены для печати на принтере.
  <Floater Width="400" Padding="5,0,5,0"  HorizontalAlignment="Center">
                <BlockUIContainer>
```

```
  <Image Source="d:\Классы.png" Stretch="Uniform"></Image>
                </BlockUIContainer>
  </Floater>
```

В принципе они эквивалентны PDF-файлам Adobe. В WPF имеется один тип фиксированных документов, в котором используется стандарт Microsoft XPS (XML Paper Specification - XML-спецификация печатных документов). Потоковые документы - это документы, предназначенные для просмотра на экране монитора. Как и фиксированные документы, потоковые документы поддерживают расширенную компоновку. WPF может оптимизировать потоковый документ на основе указанного способа просмотра документа: динамически компоновать содержимое на основе такой информации, как размеры окна просмотра, разрешение экрана и т.д. В принципе потоковые документы в основном используются для тех же целей, что и документы HTML, но обладают более совершенными возможностями компоновки текста. Потоковые документы более важны для создания приложений, а фиксированные - для создания документов, которые нужно распечатывать без изменений (например, публикаций). WPF обеспечивает поддержку обоих видов документов посредством различных контейнеров.

```
        </Paragraph>
    </FlowDocument>
</Page>
```

Шаг 4. В файле MainWindow.xaml.cs дополнить программным кодом ранее созданный метод обработки события:

```
private void LinkClicked(object sender, RoutedEventArgs e)
        {
            NavigationWindow win = new NavigationWindow();
            win.NavigationService.Navigate(new Page1());
            win.Show();
        }
```

Шаг 5. Запустить проект на выполнение. Рисунок 3.27 иллюстрирует образ страницы окна с потоковым документом.

Содержимое документа отображено в две колонки. В левом нижнем углу продемонстрирована работа функции поиска.

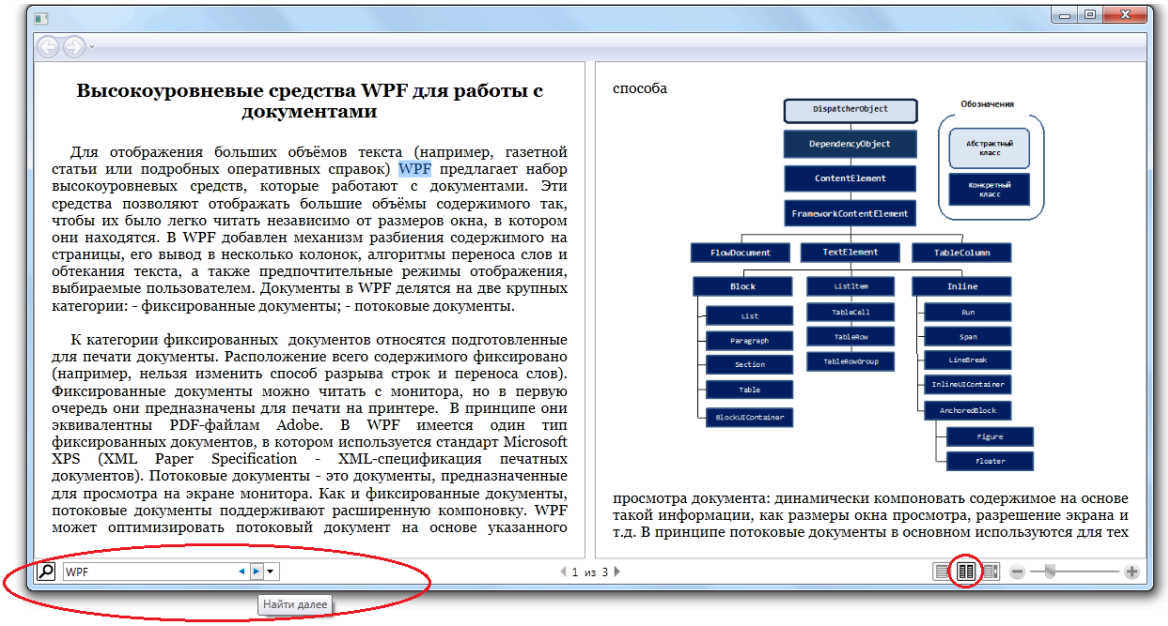

Рисунок 3.27

Текстовое содержимое в потоковом документе по умолчанию выравнивается по ширирне строки. Это поведение можно изменить с помощью свойства `TextAlignment`. Чтобы улучшить читаемость выровненного по ширине текста, можно воспользоваться оптимальной компоновкой абзаца, которая распределяет пробелы максимально равномерно и позволяет избежать появления «змеек» пробелов и неравномерно расставленных слов, что бывает в случае применения упрощенных алгоритмов выравнивания строк (например, используемых в web-браузерах).

Обычно функция оптимальной компоновки абзаца не используется из-за повышенной сложности алгоритма тотального заполнения. Чтобы активизировать оптимальную компоновку абзацев, нужно присвоить свойству `IsOptimalParagraphEnabled="True"`.

3.4.2.3 Примеры программного взаимодействия с потоковыми документами. Потоковые документы можно сохранять в файле, загружать из файла, вывести на печать, создавать и изменять программно.

Для знакомства с основными приёмами реализации перечисленных действий необходимо выполнить последовательность шагов.

Шаг 1. Создать новый проект WPF Application.

Шаг 2. Изменить XAML-описание окна MainWindow.xaml на код, приведённый ниже.

Листинг MainWindow.xaml

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Работа с документами" Height="350" Width="525">
    <!-- Эта панель устанавливает содержимое окна -->
    <DockPanel Height="Auto">
        <!-- Закрепить систему меню в панели -->
        <Menu  DockPanel.Dock="Top" VerticalAlignment="Top" Height="Auto"
FontWeight="Bold" FontSize="16">
            <MenuItem Header ="Потоковые документы">
                <MenuItem Header="Сохранить" Click="Save_Click"/>
                <MenuItem Header="Загрузить" Click="Load_Click"/>
                <MenuItem Header="Печать" Click="Print_Click"/>
                <MenuItem Header="Создать" Click="Create_Click"/>
                <MenuItem Header="Динамическое изменение "
Click="Edit_Click"/>
            </MenuItem><Separator></Separator>
            <MenuItem Header="Выход" Click="Exit_Click"/>
        </Menu>
        <!-- Flow-документ для обработки -->
        <RichTextBox Height="Auto" DockPanel.Dock="Top" Name="richTextBox">
            <FlowDocument Name="p">
                <Paragraph FontSize="20" FontWeight="Bold"
TextAlignment="Center">Фиксированные документы</Paragraph>
                <Paragraph TextIndent="20" TextAlignment="Justify">
 Если потоковые документы позволяют динамически компоновать сложное
текстовое содержимое,
то фиксированные документы, гораздо менее гибки. Фиксированные документы
пригодны для печати, их можно распространять и печатать на любом выводном
```

```
устройстве в полном соответствии с первоначальным источником. Фиксированные документы являются документами на основе стандарта XPS, который требует специфической структуры и упакованного документа (http://www.microsoft.com/whdc/xps/xpsspec.mspx). Эта структура основана на OPC (Open Packaging Convention) - соглашении об открытой упаковке, на котором также базируется документ Word (OOXML, или Office Open XML). Внутри папки документа в XPS можно найти различные папки для метаданных, ресурсов (таких как шрифты и изображения), а также сам документ.
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
    </DockPanel>
</Window>
```

Дескриптивный код содержимого окна содержит несколько элементов, размещённых в контейнер `<DockPanel>`, это:

- `<Menu>` для создания главного меню;
- `<RichTextBox>` для размещения в нём элемента `<FlowDocument>`, который будет хранить содержимое потокового документа.

Шаг 3. С помощью контекстного меню создать методы обработки событий, указанных в XAML-описании файла: `Click="Save_Click"`, `Click="Load_Click"`, `Click="Print_Click"`, `Click="Create_Click"`, `Click="Edit_Click"` и `Click="Exit_Click"`.

Шаг 4. Используя ниже приведённый листинг файла MainWindow.xaml.cs перенести программный код обработки событий.

Листинг MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
// Используем пространство имён стандартных диалогов
using Microsoft.Win32;
using System.IO;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
```

```
        {
            InitializeComponent();
        }

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog saveFile = new SaveFileDialog();
            saveFile.Filter = "XAML Files (*.xaml)|*.xaml|RichText Files
(*.rtf)|*.rtf|All Files (*.*)|*.*";
            if (saveFile.ShowDialog() == true)
            {
                // Создание контейнера TextRange для всего документа.
                TextRange documentTextRange = new TextRange(
                richTextBox.Document.ContentStart,
richTextBox.Document.ContentEnd);
                // Если такой файл существует, он перезаписывается,
                using (FileStream fs = File.Create(saveFile.FileName))
                {
                    if
(System.IO.Path.GetExtension(saveFile.FileName).ToLower() == ".rtf")
                    {
                        documentTextRange.Save(fs, DataFormats.Rtf);
                    }
                    else
                    {
                        documentTextRange.Save(fs, DataFormats.Xaml);
                    }
                }
            }
        }

        private void Load_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFile = new OpenFileDialog();
            openFile.Filter = "XAML Files (*.xaml)|*.xaml|RichText Files
(*.rtf)|*.rtf|All Files (*.*)|*.*";
            if (openFile.ShowDialog() == true)
            {
                TextRange documentTextRange = new TextRange(
                richTextBox.Document.ContentStart,
richTextBox.Document.ContentEnd);
                using (FileStream fs = File.Open(openFile.FileName,
FileMode.Open))
                {
                    documentTextRange.Load(fs, DataFormats.Rtf);
                }
            }
        }

        private void Print_Click(object sender, RoutedEventArgs e)
        {
            PrintDialog printDialog = new PrintDialog();
            if (printDialog.ShowDialog() == true)
            {
                printDialog.PrintDocument(
```

```
 ((IDocumentPaginatorSource)richTextBox.Document).DocumentPaginator,
                "Потоковый документ");
            }
        }

        private void Create_Click(object sender, RoutedEventArgs e)
        {
            // Создание первой части фразы.
            Run runFirst = new Run();
            runFirst.Text = "Приведён фрагмент кода, который создаёт
документ с одним абзацем, часть текста в котором выделена  ";
            // Создание жирного текста.
            Bold bold = new Bold();
            Run runBold = new Run();
            runBold.Text = " жирным ";
            bold.Inlines.Add(runBold);
            // Создание последней части фразы.
            Run runLast = new Run();
            runLast.Text = "шрифтом. Затем он выводит документ в
существующем контейнере с именем richTextBox. ";
            // Добавление трех частей фразы в абзац по порядку.
            Paragraph paragraph = new Paragraph();
            paragraph.Inlines.Add(runFirst);
            paragraph.Inlines.Add(bold);
            paragraph.Inlines.Add(runLast);
            // Создание документа и добавление в него этого абзаца.
            FlowDocument document = new FlowDocument();
            document.Blocks.Add(paragraph);
            // Вывод документа.
            richTextBox.Document = document;
        }

        private void Edit_Click(object sender, RoutedEventArgs e)
        {
            Window1 w1 = new Window1();
            w1.Show();
        }

        private void Exit_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```

Шаг 4. Добавить в состав проекта новое WPF окно и заменить содержимое файла Window1.xaml на ниже следующий код разметки.

Листинг Window1.xaml

```
<Window x:Class="WpfApplication1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Изменение потокового документа" Height="300" Width="300"
Loaded="Window_Loaded">
    <DockPanel>
     <Grid Name="gridWords" DockPanel.Dock="Top">
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition></ColumnDefinition>
                <ColumnDefinition></ColumnDefinition>
            </Grid.ColumnDefinitions>
        </Grid>
        <Button DockPanel.Dock="Top" Click="Generate_Click">
         Генерировать с изменениями
        </Button>
        <FlowDocumentScrollViewer Name="docViewer">
            <FlowDocument Name="document">
                <Paragraph FontSize="20"  FontWeight="Bold"
TextAlignment="Center">
                    АКТ
                </Paragraph>
                <Paragraph TextIndent="20">
                    Обследование учебной лаборатории
                    <Span Tag="Аудитория">t1</Span>
                    с целью:
                    <Span Tag="Цель проверки">t2</Span>.
                </Paragraph>
                <Paragraph TextIndent="20">
                    Состояние аудитории:
                    <Span Tag="Состояние помещения">t3</Span>,
оборудование для проведения занятий
                    <Span Tag="Оценка оборудования">t4</Span>.
                </Paragraph>
                <Paragraph TextIndent="20">
                    Акт составил
                    <Span Tag="Ответственный">t5</Span>
                </Paragraph>
                <Paragraph TextAlignment="Right" >
                    Подпись, дата
                </Paragraph>
            </FlowDocument>
        </FlowDocumentScrollViewer>
    </DockPanel>
</Window>
```

Шаг 5. С помощью контекстного меню создать методы обработки событий, указанных в XAML-описании файла: `Loaded="Window_Loaded"` и `Click="Generate_Click"`.

Шаг 6. Используя ниже приведённый листинг файла Window.xaml.cs перенести программный код обработки событий.

Листинг Window.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Markup;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
// Очистка таблицы элементов управления текстовыми записями.
            gridWords.Children.Clear();
// Перебор абзацев
            foreach (Block block in document.Blocks)
            {
                Paragraph paragraph = block as Paragraph;
// Поиск объектов Span
                foreach (Inline inline in paragraph.Inlines)
                {
                    Span span = inline as Span;
                    if (span != null)
                    {
// Создание ячейки в строке для данного термина.
                        RowDefinition row = new RowDefinition();
                        gridWords.RowDefinitions.Add(row);
// Добавление описательной метки для данного термина.
                        Label lbl = new Label();
                        lbl.Content = inline.Tag.ToString() + ":";
                     Grid.SetColumn(lbl, 0);
                     Grid.SetRow(lbl, gridWords.RowDefinitions.Count - 1);
                        gridWords.Children.Add(lbl);
// Добавление текстового поля, в котором пользователь
                        // может ввести значение для данного термина.
                        TextBox txt = new TextBox();
                     Grid.SetColumn(txt, 1);
                     Grid.SetRow(txt, gridWords.RowDefinitions.Count - 1);
                        gridWords.Children.Add(txt);
// Привязка текстового поля к элементу Run, где должен появиться текст,
                        txt.Tag = span.Inlines.FirstInline;
                    }
                    docViewer.Visibility = Visibility.Hidden;
                }
            }
        }

        private void Generate_Click(object sender, RoutedEventArgs e)
```

```
        {
            foreach (UIElement child in gridWords.Children)
            {
                if (Grid.GetColumn(child) == 1)
                {
                    TextBox txt = (TextBox)child;
                    if (txt.Text != "") ((Run)txt.Tag).Text = txt.Text;
                }
            }
            docViewer.Visibility = Visibility.Visible;
        }
    }
}
```

Шаг 7. Запустить проект на выполнение для проверки работоспособности программного кода.

Рисунок 3.28 иллюстрирует стандартный диалог, сопровождающий **сохранение** потокового документа. При этом используется объект класса `TextRange`, при создании которого указывается пара объектов `TextPointer`, определяющих начало и окончание содержимого. Далее вызывается метод `documentTextRange.Save()` с указанием требуемого формата экспорта (текст, XAML, пакет XAML или RTF) с помощью поля из класса `DataFormats`.

Форматы пакета XAML и RTF требуют полномочий на выполнение неуправляемого кода.

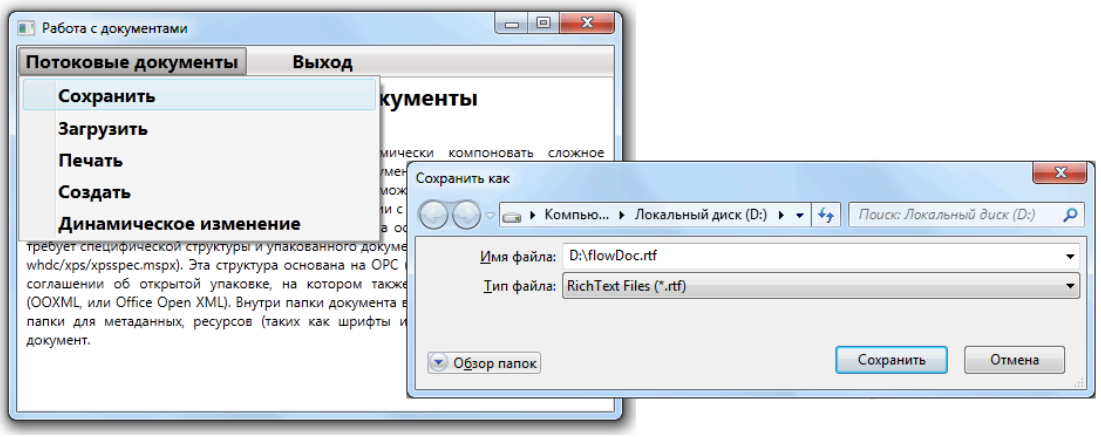

Рисунок 3.28

При **загрузке** документа из файла также используется класс `TextRange` и метод `documentTextRange.Load()`. Класс `TextRange` позволяет создать полезный контейнер, позволяющий преобразовывать файлы из одного формата в другой, а также применять форматирование.

Напечатать потоковый позволяет метод `PrintDocument()` системного класса `PrintDialog`. Все контейнеры потоковых документов поддерживают печать. В диалоговом окне `PrintDialog` пользователь может выбрать принтер и задать параметры печати.

Потоковые документы могут **создаваться** не только с помощью разметки, но и программно.

Программная генерация сложного потокового документа – достаточно трудоёмкая процедура, так как приходится вручную создавать каждый элемент XAML, а затем устанавливать его свойства. Понадобится также создавать элементы `<Run>` для

упаковки каждой порции текста, так как они не будут создаваться автоматически.

В примере создаётся простой документ с одним абзацем, часть текста в котором выделена жирным шрифтом. Созданный программно `FlowDocument` размещается в контейнер с именем `richTextBox`.

Программная генерация потокового документа полезна, если необходимо просматривать части потокового документа и динамически **изменять** их.

Основная проблема поиска элемента, который необходимо изменить, связана с тем, что в потоковых документах используется глубокое вложение содержимого неизвестной заранее структуры. Содержимое всегда хранится в элементе `<Run>`, даже если он не объявлен явным образом.

В перемещении по структуре потокового документа используются несколько приёмов:

- использование коллекции `FlowDocument.Blocks` и её инструкций перехода к первому или последнему блочному элементу `FlowDocument.Blocks.FirstBlock()` или `FlowDocument.Blocks.LastBlock()`;

- использование свойства `Block.NextBlock` или `Block.PreviousBlock` при переходе от одного блочного элемента к следующему (или предыдущему) блоку, а также использование коллекции `Block.SiblingBlocks` для просмотра всех блочных элементов, находящихся на одном уровне;

- использование в блочных элементах элементов-коллекций, например таких, как `List` с коллекцией `ListItem`, элемент `Section` с коллекцией `Blocks`, элемент `Paragraph` с коллекцией `Inlines`.

Если требуется изменить текст внутри потокового документа, то необходимо выделить с помощью элемента `<Span>` именно ту часть, которую нужно изменить.

Рисунок 3.29 иллюстрирует сценарий формирования потокового документа, в соответствии с которым пользователь вводит текст в поля таблицы, далее этот текст используется для изменения документа.

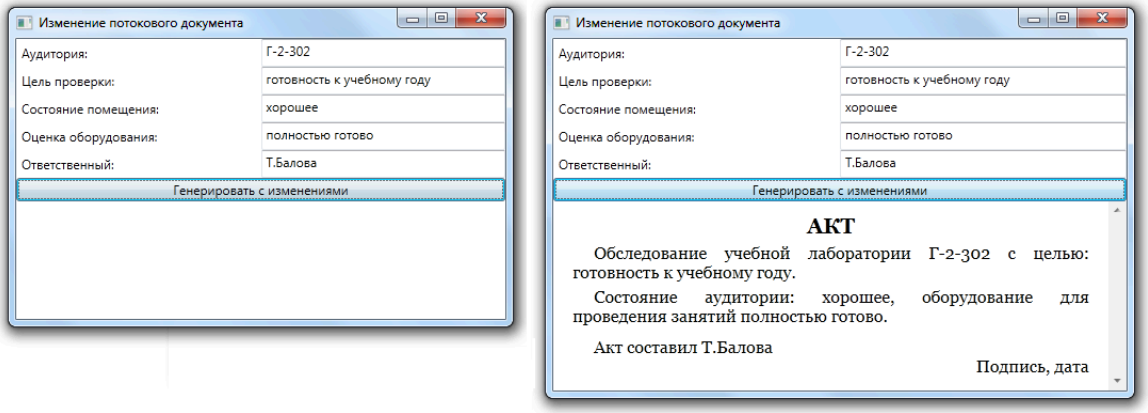

Рисунок 3.29

В примере метод `Window_Loaded()` содержит два цикла `foreach`, в которых осуществляется выбор именованных элементов `<Span>` из всех абзацев верхнего уровня в любом документе. Сначала выбираются абзацы в коллекции `Blocks`, а затем элементы `<Span>` в коллекции `Paragraph.Inlines` каждого абзаца.

Каждый раз, находя элемент `<Span>`, метод создаёт текстовое поле, в котором пользователь сможет ввести новое значение, и добавляет его в таблицу (в разметке элемент `<Grid Name="gridWords">`) над документом (вместе с описательными метками `t1`,…, `t5`). Для реализации замены каждое текстовое окно `TextBox` хранит

ссылку (в свойстве `TextBox.Tag`) на элемент `<Run>` с текстом внутри соответствующего элемента `<Span>`, с помощью которого выделяется в блоке заданный текст. Тип выделения указывается с помощью дополнительной информации - строки, хранящейся в свойстве `Span.Tag`.

Свойство `Tag` в любом элементе зарезервировано для использования программистом. Оно может содержать любое значение или объект.

3.4.2.4 Пример работы с фиксированными документами.

Если потоковые документы позволяют динамически компоновать сложное текстовое содержимое, то фиксированные документы, гораздо менее гибки. Фиксированные документы пригодны для печати, их можно распространять и печатать на любом выводном устройстве в полном соответствии с первоначальным источником.

Фиксированные документы являются документами на основе стандарта XPS, который требует специфической структуры и упакованного документа (http://www.microsoft.com/whdc/xps/xpsspec.mspx). Эта структура основана на OPC (*Open Packaging Convention*) - соглашении об открытой упаковке, на котором также базируется документ Word (OOXML, или Office Open XML).

Документы фиксированного формата предназначены для приложений, требующих точного представления в режиме «what you see is what you get» (WYSIWYG), независимо от используемого дисплея или принтера.

Документы фиксированного формата обычно используются при подготовке публикаций с помощью настольных издательских средств, обработке текста и разметке формы, где строгое соблюдение исходного дизайна страницы является обязательным. Документы фиксированного формата, как часть макета, поддерживают точное позиционирование содержимого элементов, независимо от используемых устройств отображения или печати. Например, страница формата фиксированного документа, просматриваемая на экране с разрешением 96 точек на дюйм, будет отображаться точно так же на выводе лазерного принтера с разрешением 600 точек на дюйм или устройстве фотовывода с разрешением 4800 точек на дюйм. Макет страницы остаётся неизменным во всех случаях, в то время как качество документа повышается в соответствии с возможностями каждого устройства.

Внутри папки документа в XPS можно найти различные папки для метаданных, ресурсов (таких как шрифты и изображения), а также сам документ. Описание метаданных документа осуществляется на XPS-подмножестве XAML. В них используется точная, фиксированная компоновка, поддерживается внедрение шрифтов и исключается случайное изменение компоновки.

XPS - это стандарт, тесно интегрированный в операционную систему Windows 7 и поддерживаемый драйвером печати, который может создавать документы XPS (в любом приложении), и средством просмотра для их отображения. Эти две части похожи на Adobe Acrobat, позволяя пользователям создавать и просматривать электронные документы, пригодные для печати, и добавлять аннотации.

Приложения Microsoft Office 2007 (и версии выше) позволяют сохранять документы в форматах XPS и PDF, так рисунок 3.30 иллюстрирует порядок сохранения документа в формате XPS. XPS-файлы поддерживают электронную подпись и широко используются в электронном документообороте. На рисунке 3.31 демонстрируются процесс оформления электронной подписи с помощью обозревателя XPS-файла Windows 2010. 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 (рисунок 3.32).

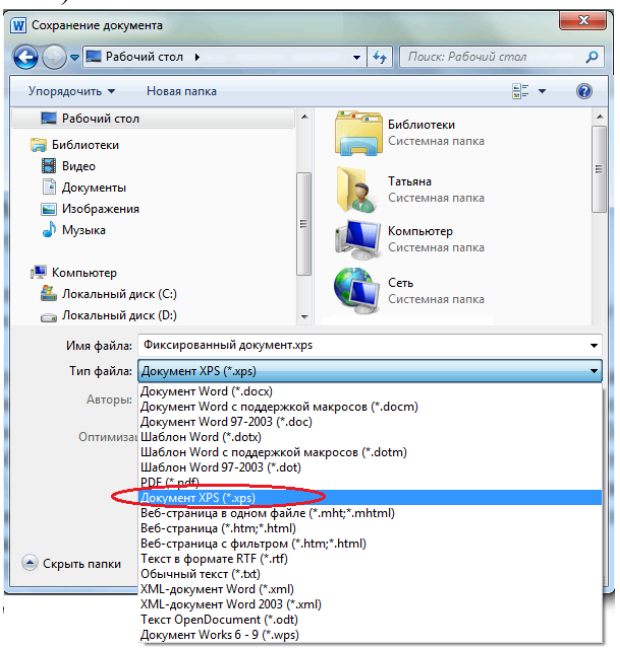

Рисунок 3.30

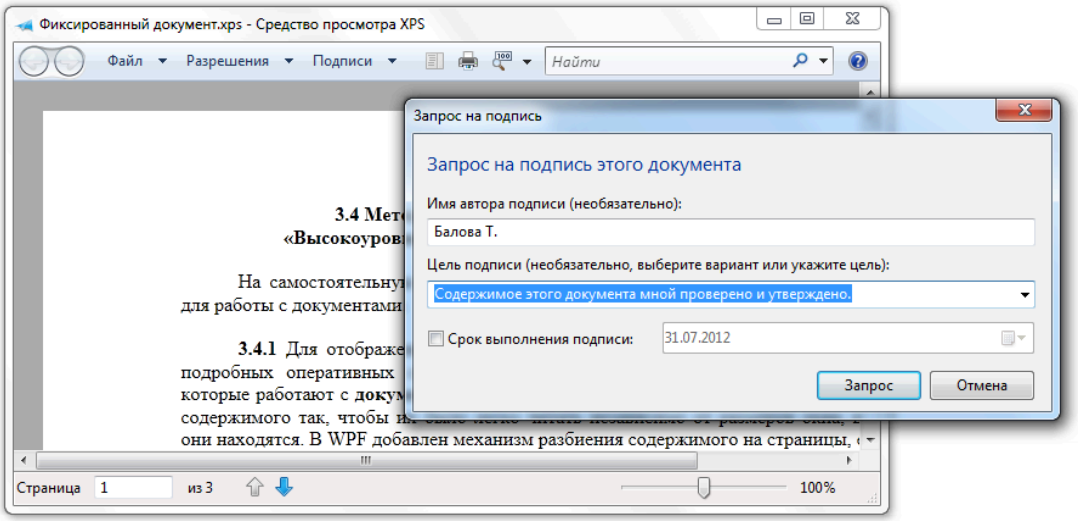

Рисунок 3.31

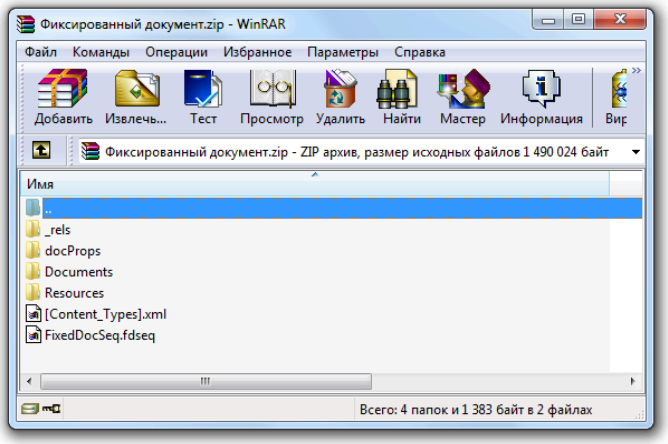

Рисунок 3.32

Для вывода на экран документа XPS применяется именованный контейнер <DocumentViewer>, в котором имеются элементы управления для поиска и изменения масштаба. Контейнер <DocumentViewer> предлагает такой же набор свойств, методов и команд, как и контейнеры <FlowDocument>.

Для изучения методов класса XpsDocument и элемента <DocumentViewer> необходимо в проект подраздела **3.4.2.3** внести следующие дополнения.

Шаг 1. С помощью пункта меню проводника **Add Reference** добавить в состав проекта две .NET сборки `ReachFramework` и `System.Printing`.

Шаг 2. В состав проекта включить новое окно Window2.xaml и модифицировать его дескрипторный код следующим образом:

```
<Window x:Class="WpfApplication1.Window2"

xmlns:annot="clr-namespace:System.Windows.Annotations;assembly=PresentationFramework"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Работа с документом  XPS" Loaded="Window_Loaded">
    <DocumentViewer Name="docViewer">
    </DocumentViewer>
</Window>
```

Шаг 3. С помощью контекстного меню добавить метод обработки события `Loaded="Window_Loaded"` и добавить программный код обработки в файле Window2.xaml.cs:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFile = new OpenFileDialog();
            openFile.Filter = "XPS-файлы|*.xps|All Files|*.*";
            if (openFile.ShowDialog() == true)
            {
 XpsDocument doc = new XpsDocument(openFile.FileName, FileAccess.Read);
                docViewer.Document = doc.GetFixedDocumentSequence();
                doc.Close();
            }
        }
```

Шаг 4. В текст файла Window2.xaml.cs добавить необходимые пространства имён подключённых .NET сборок:

```
// Используем пространство имён окон стандартных диалогов
using Microsoft.Win32;
// Используем пространство имён Xps
using System.Windows.Xps;
using System.Windows.Xps.Packaging;
using System.IO;
using System.IO.Packaging;
```

Шаг 5. В разметку меню главного окна в файле MainWindow.xaml добавить элемент:

```
<MenuItem Header="Загрузить XPS" Click="Load_xps"/><Separator></Separator>
```

Для события `Click="Load_xps"` с помощью контекстного меню добавить обработчик события с процедурным кодом:

```
private void Load_xps(object sender, RoutedEventArgs e)
        {
            Window2 w2 = new Window2();
            W2.Show();
        }
```

Шаг 6. Запустить проект на выполнение для проверки работоспособности добавленного программного кода.

Рисунок 3.33 иллюстрирует результат выбора, рисунок 3.34 – результат отображения XPS-документа контейнере `<DocumentViewer>`.

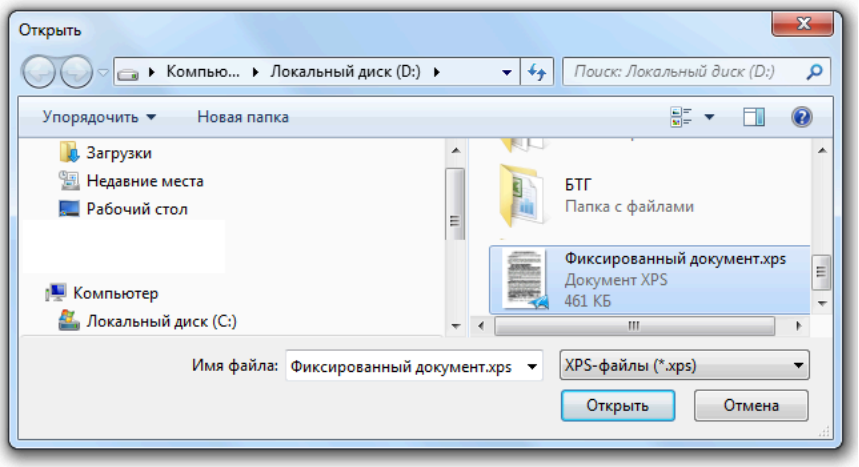

Рисунок 3.33

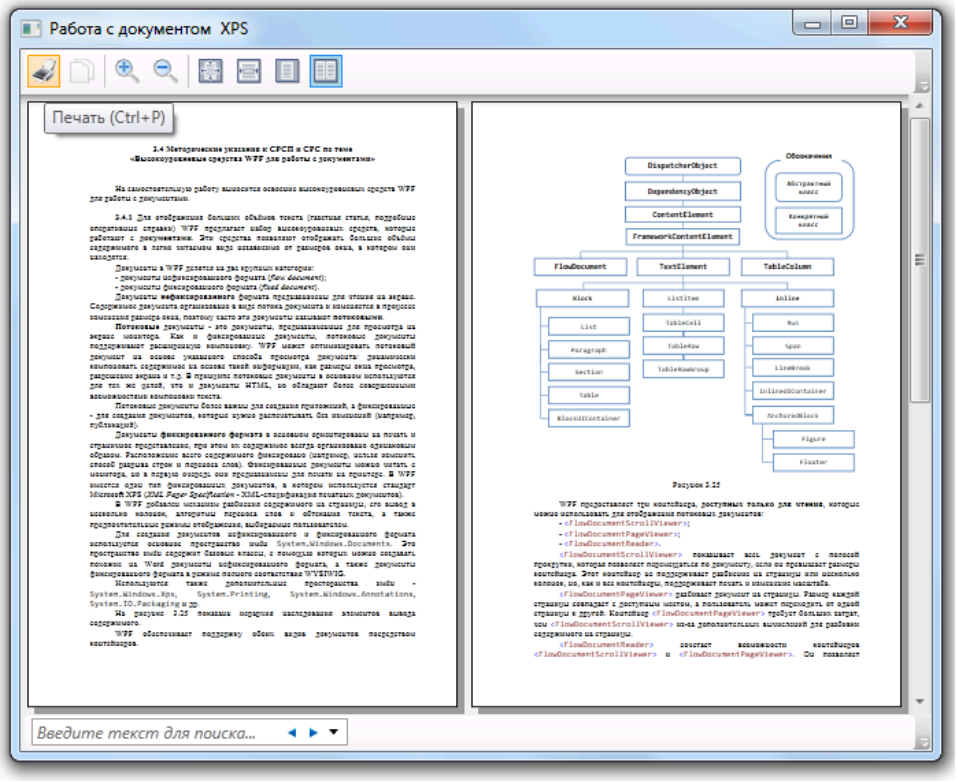

Рисунок 3.34

Класс `XpsDocument` предоставляет метод `GetFixedDocumentSequence()`, который возвращает ссылку на документ со всем его содержимым.

XPS-документы тесно связаны с концепцией печати. Отдельный XPS-документ имеет фиксированный размер страницы и компонует свой текст так, чтобы он занял доступное место.

3.5 Задания и контрольные вопросы для самопроверки по тематике самостоятельной работы

Задания по СРСП:

- используя примеры создания потоковых и фиксированных документов, спректировать приложения, обеспечивающие работу с документами;
- использую ссылки на информационные ресурсы и источники литературы изучить процедуры создания фиксированных документов.

Вопросы для защиты лабораторной работы и СРСП:

- дать характеристику двум категориям документов WPF;
- основные отличия потоковых документов от документов XPS;
- дать характеристику WPF контейнеров, используемых для отображения потоковых документов: `<FlowDocumentScrollViewer>`, `<FlowDocumentPageViewer>`, `<FlowDocumentReader>`;
- какие элементы вывода содержимого относятся к блочным и строковым элементам;
- свойства и методы элементов блочного уровня `<Paragraph>`, `<Run>`, `<List>`, `<Table>` и `<Section>`, которые используются при работе с высокоуровневыми потоковыми документами;
- возможности программного взаимодействия с потоковыми документами;
- соглашении об открытой упаковке OPC и стандарт XPS;
- контейнеры и классы для работы с фиксированными документами.

4 СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Мак-Дональд М. WPF 4: Windows Presentation Foundation в .NET 4.0 с примерами на C# 2010 для профессионалов. : Пер. с англ. - М. : ООО "И.Д. Вильямс", 2011. – 1024с.

2. Мак-Дональд М. WPR Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильямс". 2008.

3. Троелсен, Эндрю. Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд.: Пер. с англ. - М. : ООО "И.Д. Вильямс", 2011. - 1392 с.

4. Ватсон Б. C#4.0 на примерах. - СПб.: БХВ-Петербург, 2011- 608 с.

5. К. Нейгел, Б. Ивьен, Дж. Глинн, К. Уотсон C# 4.0 и платформа .NET 4 для профессионалов М. : ООО "И.Д. Вильямс", 2011,

6. Алекс Макки Введение в .NET 4.0 и Visual Studio 2010. Пер. с англ. - М. : ООО "И.Д. Вильямс", 2010. - 416 с.

7. Морони, Л. Введение в Silverlight 2 Microsoft Press, 2009

Интернет-ресурсы:

http://msdn.microsoft.com/ru-ru/library/system.windows.propertypath.aspx
http://www.intuit.ru/department/se/dawpfsl
http://www.microsoft.com/whdc/xps/xpsspec.mspx