Практическая работа 29
Разработка пользовательского интерфейса в WPF

Цель занятия: Получить практический опыт разработки пользовательского Windows-интерфейса с использованием WPF

Перечень оборудования и программного обеспечения

Персональныйкомпьютер
Microsoft Office (Word, Visio)
Microsoft Visual Studio

Краткие теоретические сведения

WindowsPresentationFoundation (WPF) изменила мир программирования desktop приложений. Положив в основу технологию DirectX, Microsoft дала возможность разработчикам быстро создавать сложные элементы управления и полностью управлять процессом визуализации. Теперь создать красивую кнопку с анимационными эффектами можно не написав ни одной строки кода на C#. Работать с мультимедийным содержимым стало намного проще, расширилась модель привязки данных, печати и работы с документами. WPF комбинирует лучшие аспекты традиционной разработки для Windows с множеством нововведений, позволяя строить богатые графикой пользовательские интерфейсы.

Основные классы:

System.Threading.DispatcherObject: Приложения WPF используют однопоточную модель, а это означает, что весь пользовательский интерфейс принадлежит единственному потоку.

Взаимодействовать с элементами пользовательского интерфейса из других потоков небезопасно. Чтобы содействовать работе этой модели, каждое приложение WPF управляется диспетчером, координирующим сообщения. Будучи унаследованным от DispatcherObject, каждый элемент пользовательского интерфейса может удостовериться, выполняется ли код в правильном потоке, и обратиться к диспетчеру, чтобы направить код в поток пользовательского интерфейса;

System.Windows.DependencyObject: В WPF центральный путь взаимодействия с экранными элементами проходит через свойства. На ранней стадии цикла проектирования архитекторы WPF решили создать более мощную модель свойств, которая положена в основу таких средств, как уведомления об изменениях, наследуемые значения по умолчанию и более экономичное хранилище свойств. Конечным результатом стало средство

свойств зависимости (dependencyproperty). Наследуясь от DependencyObject, классы WPF получают поддержку свойств зависимости;

System.Windows.Media.Visual: Каждый элемент, появляющийся в WPF, в основе своей является Visual. Можно воспринимать класс Visual как единственный объект рисования, инкапсулирующий в себе базовые инструкции рисования, дополнительные возможности рисования и базовую функциональность. Любой класс, унаследованный от Visual, обладает способностью отображаться в окне;

System.Windows.UIElement: UIElement добавляет поддержку таких сущностей WPF, как компоновка (layout), ввод (input), фокус (focus) и события (events) – все, что команда разработчиков WPF называет аббревиатурой LIFE. Как и со свойствами, WPF реализует расширенную систему передачи событии, именуемую маршрутизируемыми событиями;

System.Windows.FrameworkElement: FrameworkElement – конечный пункт в центральном дереве наследования WPF. Он реализует некоторые члены, которые просто определены в UIElement;

System.Windows.Shapes.Shape: От этого класса наследуются базовые фигуры, такие как Rectangle, Polygon, Ellipse, Line и Path. Эти фигуры могут быть использованы наряду с более традиционными визуальными элементами Windows вроде кнопок и текстовых полей;

System.Windows.Controls.Control: Элемент управления (control) – это элемент, который может взаимодействовать с пользователем. К нему, очевидно, относятся такие классы, как TextBox,Button и ListBox. Класс Control добавляет дополнительные свойства для установки шрифта, а также цветов переднего плана и фона. Но наиболее интересная деталь, которую он предоставляет – это поддержка шаблонов, которая позволяет заменять стандартный внешний вид элемента управления собственным рисованием;

System.Windows.Controls.ContentControl: Это базовый класс для всех элементов управления, которые имеют отдельный блок содержимого. Сюда относится все – от скромной метки Label до окна Window. Наиболее впечатляющая часть этой модели заключается в том, что единственный кусок содержимого может быть чем угодно – от обычной строки до панели компоновки, содержащей комбинацию других фигур и элементов управления;

System.Windows.Controls.ItemsControl: Это базовый класс для всех элементов управления, которые отображают коллекцию каких-то единиц информации, вроде ListBox и TreeView. Фактически, в WPF все меню, панели инструментов и линейки состояния на самом деле являются специализированными списками, и классы, реализующие их, наследуются от ItemsControl;

System.Windows.Controls.Panel: Это базовый класс для всех контейнеров компоновки – элементов, которые содержат в себе один или более дочерних элементов и упорядочивают их в соответствии с определенными правилами компоновки.

Эти контейнеры образуют фундамент системы компоновки WPF, и их использование – ключ к упорядочиванию содержимого наиболее привлекательным и гибким способом.

ОСНОВЫ XAML

XAML представляет собой язык разметки, используемый для создания экземпляров объектов .NET. Хотя язык XAML – это технология, которая может быть применима ко многим различным предметным областям, его основное назначение – конструирование пользовательских интерфейсов WPF. Другими словами, документы XAML определяют расположение панелей, кнопок и прочих элементов управления, составляющих окна в приложении WPF.

Код XAML редко приходится писать вручную. Вместо этого обычно используется инструмент, генерирующий необходимый код XAML. Для графического дизайнера таким инструментомявляется MicrosoftExpressionBlend, а для разработчика -VisualStudio. Поскольку оба инструмента поддерживают XAML, можно создать базовый пользовательский интерфейс в VisualStudio, а затем передать его команде дизайнеров, которые доведут его до совершенства, добавив специальную графику в ExpressionBlend. Фактически такая способность интегрировать рабочий поток разработчиков и дизайнеров – одна из ключевых причин создания языка XAML.

Несмотря на то, что код XAML создается средой разработки автоматически, понимание XAML чрезвычайно важно при разработке дизайна приложения WPF. В этом отношении приложения WPF существенно отличаются от приложений WindowsForms. Понимание XAML поможет разобраться с ключевыми концепциями WPF, такими как прикрепленные свойства, компоновка, модель содержимого, маршрутизируемые события и т. д. Что более важно – есть целый ряд задач, решение которых возможно только с помощью вручную написанного XAML, либо существенно облегчается. К ним относятся перечисленные ниже:

Привязка обработчиков событий. Прикрепление обработчиков событий в наиболее распространенных местах – например, к событию Click для Button – легко сделать в VisualStudio. Однако XAML позволяет создавать более изощренные соединения. Например, можно установить обработчик события, реагирующий на событие Click в каждой кнопке окна;

Определение ресурсов. Ресурсы – это объекты, которые определены однажды в коде XAML, в специальном разделе, а затем повторно используются в разных местах кода разметки.

Ресурсы позволяют централизовать и стандартизовать форматирование и создание невизуальных объектов, таких как шаблоны и анимации;

Определение шаблонов элементов управления. Элементы управления WPF проектируются как лишенные внешнего вида; это значит, что можно подставлять собственные визуальные представления элементов вместо стандартных. Чтобы сделать это, необходимо создать собственный шаблон элемента управления, который представляет собой не что иное, как блок разметки XAML;

Написание выражений привязки данных. Привязка данных позволяет извлекать данные из объекта и отображать их в привязанном элементе. Чтобы установить это отношение и конфигурировать его работу, необходимо добавить выражение привязки данных к коду разметки XAML;

Определение анимаций. Анимации – распространенный компонент приложений XAML. Обычно они определяются как ресурсы, конструируются с использованием разметки XAML, а затем привязываются к другим элементам управления (или инициируются в коде).

Большинство разработчиков WPF используют комбинацию приемов, разрабатывая часть пользовательского интерфейса с помощью инструмента проектирования (VisualStudio или ExpressionBlend), а затем проводя тонкую настройку посредством редактирования кода разметки вручную.

WPF сама рисует каждую кнопку, текст, рамку, фон. В основе новых возможностей **WPF** лежит мощная инфраструктура, основанная на **DirectX** и аппаратно ускоренной графике. Это означает, что вы можете использовать богатые графические эффекты без ущерба для производительности.

WPF не зависит от разрешения экрана, все измеряется в аппаратно-независимых единицах:

[Размер физического элемента] = **[Размер аппаратно-независимой единицы]** X **[DPI системы]** = 1/96 дюйма X 96 dpi = 1 пиксель.

Если значение DPI системы равно 120:
1/96 дюйм X 120 dpi = 1,25 пикселей.

Пример создания окна:

```
<Window x:Class="WPFTest.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">
<Grid>

</Grid>
</Window>
```

Первый элемент это Window – созданное окно. **x:Class="WPFTest.Window1"** указывает на класс, в котором находится реализация окна.

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml — подключается два пространства имен **WPF** в которых находятся все базовые типы. **Title** — имя отображаемое в заголовке окна, **Height** и **Width**высота и ширина окна и закрывающий тег. Далее это содержимое окна или **Content**. **Grid** - это контейнер для всего что будет находится в окне. Заменим **<Grid></Grid>** и на**<StackPanel></StackPanel>**. Теперь контейнером всего содержимого окна будет **StackPanel**(это не обычная панель которую вы привыкли видеть в **WinForms**).

Внутри **StackPanel** создадим кнопку и надпись:

```
<StackPanel>
<Button>Click Me!</Button>
<TextBlock>Text</TextBlock>
</StackPanel>
```

Запустите и попробуйте растянуть и сжать окно. Все содержимое растягивается и масштабируется вместе с окном. В **WPF** содержимым кнопки или любого другого элемента может быть что угодно. В кнопку можно вложить еще одну кнопку или чекбокс.

```
<StackPanel>
<Button>
<StackPanel>
<TextBlock>Text In Button</TextBlock>
<CheckBox>ChecckBoxIn Button</CheckBox>
<Button>Button In Button</Button>
</StackPanel>
</Button>
</StackPanel>
```

В **XAML** можно создать кнопку разными способами:

```
<Button Content="MyButton" />
```

Возможно и так:

```
<Button>
<Button.Content>
<TextBlock>TextIn Button</TextBlock>
</Button.Content>
</Button>
```

Введение в компоновку

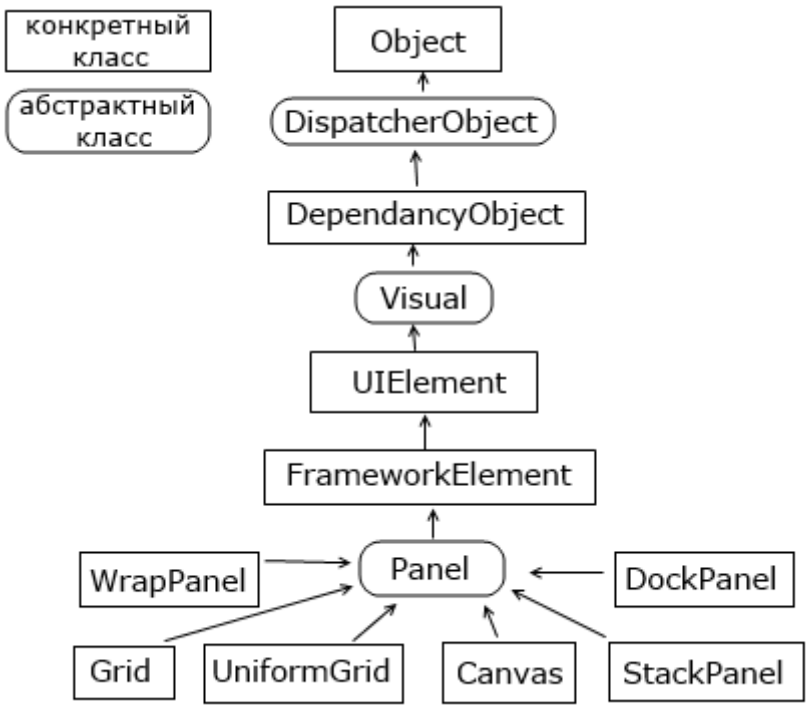
Чтобы перейти уже непосредственно к созданию красивых интерфейсов и их компонентов, сначала необходимо познакомиться с компоновкой. Компоновка (layout) представляет собой процесс размещения элементов внутри контейнера. В большинстве своем программы используют жестко закодированные в коде размеры элементов управления. WPF уходит от такого подхода в пользу так называемого "резинового дизайна", где весь процесс позиционирования элементов осуществляется с помощью компоновки.

В WPF компоновка осуществляется при помощи специальных контейнеров. Фреймворк предоставляет нам следующие контейнеры: Grid, UniformGrid, StackPanel, WrapPanel, DockPanel и Canvas.

Различные контейнеры могут содержать внутри себя другие контейнеры. Кроме данных контейнеров существует еще ряд элементов, такие как TabPanel, которые могут включать другие элементы и даже контейнеры компоновки, однако на саму компоновку не столь влияют, в отличие от выше перечисленных. Кроме того, если не хватает стандартных контейнеров, можно определить свои с нужной функциональностью.

Контейнеры компоновки позволяют эффективно распределить доступное пространство между элементами, найти для него наиболее предпочтительные размеры.

Элементы компоновки в WPF



Процесс компоновки проходит два этапа: измерение (measure) и расстановка (arrange). На этапе измерения контейнер производит измерение

предпочтительного для дочерних элементов места. Однако не всегда контейнер имеет достаточно места, чтобы расставить все элементы по их предпочтительным размером, поэтому их размеры приходится усекать. Затем происходит этап непосредственной расстановки дочерних элементов внутри контейнера.

Grid

Это наиболее мощный и часто используемый контейнер, напоминающий обычную таблицу. Он содержит столбцы и строки, количество которых задает разработчик. Для определения строк используется свойство **RowDefinitions**, а для определения столбцов - свойство **ColumnDefinitions**:

```
    <Grid.RowDefinitions>
  <RowDefinition></RowDefinition>
  <RowDefinition></RowDefinition>
  <RowDefinition></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
  <ColumnDefinition></ColumnDefinition>
  <ColumnDefinition></ColumnDefinition>
  <ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
```

Каждая строка задается с помощью вложенного элемента RowDefinition, который имеет открывающий и закрывающий тег. При этом задавать дополнительную информацию необязательно. То есть в данном случае у нас определено в гриде 3 строки.

Каждая столбец задается с помощью вложенного элемента ColumnDefinition. Таким образом, здесь мы определили 3 столбца,то есть в итоге у нас получится таблица 3x3.

StackPanel

Это более простой элемент компоновки. Он располагает все элементы в ряд либо по горизонтали, либо по вертикали в зависимости от ориентации. Например,

```
    <Window x:Class="Layout.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="300" Width="300">
  <Grid>
    <StackPanel>
      <Button Background="Blue" Height="30" Content="1" />
      <Button Background="White" Height="30" Content="2" />
      <Button Background="Red" Height="30" Content="3" />
    </StackPanel>
```

```
    </Grid>
</Window>
```

В данном случае для свойства Orientation по умолчанию используется значение Vertical, то есть StackPanel создает вертикальный ряд, в который помещает все вложенные элементы сверху вниз. Мы также можем задать горизонтальный стек. Для этого нам надо указать свойство Orientation="Horizontal":

```
    <Window x:Class="Layout.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="300" Width="300">
  <Grid>
    <StackPanel Orientation="Horizontal">
      <Button Background="Blue" Width="30" Content="1" />
      <Button Background="White" Width="30" Content="2" />
      <Button Background="Red" Width="30" Content="3" />
    </StackPanel>
  </Grid>
</Window>
```

Задания

1 Изучить теоретические сведения и задание к работе

2 В соответствии с вариантом задания разработать варианты пользовательского интерфейса по технологии WPFдля модуля с использованием разных типов компоновки. Изменить цветовую гамму для используемых элементов управления.

3 Построить отлаженный модуль.

Порядок выполнения работы

…

Содержание отчета

1 Название работы
2 Цель работы
3 Технические средства обучения
4 Задания (условия задач)

5 Порядок выполнения работы
6 Листинг отлаженной программы в соответствии с вариантом задания
7 Ответы на контрольные вопросы
8 Вывод

Варианты заданий

Варианты заданий представлены в практической работе 28.

Используемая литература

1. Гниденко, И. Г. Технология разработки программного обеспечения: учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М.: Издательство Юрайт, 2017.
2. Шарп Джон Ш26 Microsoft Visual C#. Подробное руководство. 8-е изд. — СПб.: Питер, 2017.
3. Васильев А.Н. Программирование на C# для начинающих. Основные сведения. – Москва: Эксмо, 2018.
4. Васильев А.Н. Программирование на C# для начинающих. Особенности языка. – Москва: Эксмо, 2019.
5. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx.
6. Мэтью Макдональд WPF: WindowsPresentationFoundation в .NET 4.0 с примерами на C# 2010 для профессионалов, М:Вильямс, 2011