

## **Программные решения для бизнеса**

## Оглавление

1	XAML .....	3
1.1	Элементы компоновки .....	3
1.1.1	Grid .....	4
1.1.2	UniformGrid .....	5
1.1.3	StackPanel .....	6
1.1.4	DockPanel .....	6
1.1.5	WrapPanel .....	7
1.2	Элементы управления .....	7
1.2.1	Button .....	8
1.2.2	CheckBox и RadioButton .....	8
1.2.3	ComboBox .....	9
1.2.4	ListView .....	9
1.2.5	DataGrid .....	10
1.2.6	Calendar и DatePicker .....	11
1.2.7	Image .....	11
2.	Стили .....	12
3	Триггеры .....	12
3.1	Триггеры свойств .....	13
3.2	Триггеры данных .....	13
3.3	Триггеры событий .....	14
3.4	Мультитриггеры .....	14
4	Темы .....	15
5	Работа с данными .....	15
5.1	Привязка данных и контекст данных. ....	21
6	Создание отчётов .....	25
6.1	Excel .....	25
6.2	Word и Pdf .....	26
7	Использование элементов управления WinForms в WPF .....	29
8	Drag and Drop .....	29
9	Мультимедиа .....	32
10	Частичные классы и методы .....	34

XAML — это декларативный язык разметки. С точки зрения модели программирования .NET Core язык XAML упрощает создание пользовательского интерфейса для приложения .NET Core. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов.

Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках. В этом заключается его отличие от большинства других языков разметки, которые, как правило, представляют собой интерпретируемые языки без прямой связи с системой резервных типов. Язык XAML обеспечивает рабочий процесс, позволяющий нескольким участникам разрабатывать пользовательский интерфейс и логику приложения, используя потенциально различные средства.

При представлении в виде текста файлы XAML являются XML-файлами, которые обычно имеют расширение .xaml. Файлы можно сохранять в любой кодировке, поддерживаемой XML, но обычно используется кодировка UTF-8.

## 1.1 Элементы компоновки

Элементы компоновки в WPF, это такие контейнеры, которые могут содержать внутри себя другие контейнеры и различные элементы. Контейнеры компоновки позволяют распределять доступное пространство.

Элементы компоновки: Grid, UniformGrid, StackPanel, WrapPanel, DockPanel

### 1.1.1 Grid

Grid напоминает обычную таблицу, он содержит столбцы и строки, количество которых задает разработчик. Чтобы создать строки нужно воспользоваться свойством RowDefinitions, а столбы ColumnDefinitions.

---

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <TextBlock Grid.Row="0" Grid.Column="0">Hello Grid</TextBlock>
  <TextBlock Grid.Row="0" Grid.Column="1">Hello Grid</TextBlock>
</Grid>
```

---

В данном примере мы создали сетку 2x2, а у дочерних элементов указали свойства Grid.Row, что указывает на то в какой строке будет находиться элемент, а Grid.Column, в каком столбце. Также есть свойства Grid.ColumnSpan и Grid.RowSpan, которые растягивают дочерние элементы на указанное количество строк или столбцов.

Размеры: можно устанавливать автоматические, абсолютные и пропорциональные.

Абсолютные размеры можно задавать в: пикселях (px), дюймах (in), сантиметрах (cm), точках (pt).

---

Автоматические размеры, столбец или строка занимает, столько места, сколько нужно для отображения элементов.

```
<RowDefinition Height="Auto"/>
```

```
<ColumnDefinition Width="Auto"/>
```

Абсолютные размеры, будут использованы заданные размеры.

```
<ColumnDefinition Width="200"/>
```

```
<RowDefinition Height="200"/>
```

Пропорциональные размеры, берется весь размер и делится на указанные части. Если указана просто \*, то столбец или строка занимает оставшуюся часть. Если несколько \*, то место распределяется поровну.

```
<ColumnDefinition Width="*"/>
```

```
<ColumnDefinition Width="0.5*"/>
```

```
<ColumnDefinition Width="1.5*"/>
```

---

### 1.1.2 UniformGrid

Аналогичен контейнеру Grid контейнер UniformGrid, только в этом случае все столбцы и строки одинакового размера и используется упрощенный синтаксис для их определения:

---

```
<UniformGrid Rows="2" Columns="2">
  <TextBlock>Left Top</TextBlock>
  <TextBlock>Right Top</TextBlock>
  <TextBlock>Left Bottom</TextBlock>
  <TextBlock>Right Bottom</TextBlock>
</UniformGrid>
```

---

### 1.1.3 StackPanel

StackPanel, располагает элементы в ряд по выбранной ориентации. По умолчанию ориентация Vertical. При горизонтальной ориентации, если надо поменять расположение элементов, то можно воспользоваться свойством FlowDirection и выбрать, как будут отображаться элементы.

---

```
<StackPanel Orientation="Horizontal" FlowDirection="RightToLeft">
  <TextBlock>Hello StackPanel</TextBlock>
  <TextBlock>Hello StackPanel</TextBlock>
  <TextBlock>Hello StackPanel</TextBlock>
</StackPanel>
```

---

### 1.1.4 DockPanel

DockPanel, контейнер, который прижимает элементы к определенной стороне, а именно: Top, Bottom, Left, Right. Также, есть одно главное свойство LastChildFill, которое позволяет последнему элементу, занять всё оставшееся пространство, по умолчанию это свойство имеет значение true.

---

```
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top" Background="AliceBlue" Content="Верхняя кнопка" />
  <Button DockPanel.Dock="Bottom" Background="BlanchedAlmond" Content="Нижняя кнопка" />
  <Button DockPanel.Dock="Left" Background="Aquamarine" Content="Левая кнопка" />
  <Button DockPanel.Dock="Right" Background="DarkGreen" Content="Правая кнопка" />
  <Button Background="LightGreen" Content="Центр" />
</DockPanel>
```

---

### 1.1.5 WrapPanel

WrapPanel, такой же элемент как и StackPanel, только если элементы не помещаются в строке или столбце, создаются новые столбец или строка для не поместившихся элементов. У WrapPanel можно настроить ориентацию, а также высоту и ширину дочерних элементов, благодаря свойствам ItemHeight и ItemWidth.

---

```
<WrapPanel ItemHeight="30" ItemWidth="80" Orientation="Horizontal">
  <Button Background="AliceBlue" Content="1" />
  <Button Background="Blue" Content="2" />
  <Button Background="Aquamarine" Content="3"/>
  <Button Background="DarkGreen" Content="4"/>
  <Button Background="LightGreen" Content="5"/>
  <Button Background="AliceBlue" Content="6" />
  <Button Background="Blue" Content="7" />
</WrapPanel>
```

---

## 1.2 Элементы управления

Все элементы управления могут быть условно разделены на несколько подгрупп:

Элементы управления содержимым, например кнопки (Button), метки (Label)

Специальные контейнеры, которые содержат другие элементы, но в отличие от элементов Grid или Canvas не являются контейнерами компоновки - ScrollView, GroupBox

Декораторы, чье предназначение создание определенного фона вокруг вложенных элементов, например, Border или Viewbox.

Элементы управления списками, например, ListBox, ComboBox.

Текстовые элементы управления, например, TextBox, RichTextBox.

Элементы, основанные на диапазонах значений, например, ProgressBar, Slider.

Элементы для работ с датами, например, DatePicker и Calendar.

Остальные элементы управления, которые не вошли в предыдущие подгруппы, например, Image.

Далее будет разбор элементов управления и описаны свойства и события, наиболее используемые в рамках чемпионатов.

### 1.2.1 Button

Button, обычная кнопка, которая унаследована от ButtonBase.

Обработчики событий:

Click — происходит по нажатию кнопки.

Свойства:

IsDefault — позволяет вызывать обработчик событий по нажатию кнопки Enter.

IsCancel — позволяет вызывать обработчик событий по нажатию кнопки Esc.

IsEnabled — делает кнопку недоступную к нажатию, если имеет значение false.

### 1.2.2 CheckBox и RadioButton

Элементы представляют из себя обычный флажок, являются производными от ToggleButton.

Свойства:

IsThreeState — разрешает использование элемента три состояния: Checked, Unchecked, Indeterminate.

IsChecked — устанавливает флажок.



Обработчики событий:

Checked — происходит, когда IsChecked=true.

Unchecked — происходит, когда IsChecked=false.

Indeterminate — происходит, когда IsChecked=null.

### 1.2.3 ComboBox

ComboBox содержит коллекцию элементов и образует выпадающий список.

Свойства:

DisplayMemberPath — задаёт путь к значению исходного объекта (значение, которое будет отображено).

ItemsSource — возвращает или задаёт коллекцию.

SelectedIndex — возвращает или задаёт индекс объекта ComboBox.

Обработчики событий:

SelectionChanged — происходит при выборе элементов из ComboBox.

### 1.2.4 ListView

ListView — элемент управления отображает информацию на множестве строк и столбцов. Он унаследован от класса ListBox, поэтому может вести себя простой список.

Но чтобы создать более сложные по структуре данные используется свойство View. Это свойство принимает в качестве значения объект GridView, который управляет отображением данных. GridView определяет коллекцию определений столбцов - GridViewColumn, которое с помощью свойства Header определяет название столбца, а с помощью свойства DisplayMemberBinding

можно определить привязку столбца к определенному свойству добавляемого в ListView объекта.

---

```
<ListView>
  <ListView.View>
    <GridView>
      <GridViewColumn Header="HeaderTitle" DisplayMemberBinding="{Binding
Path}"></GridViewColumn>
      <GridViewColumn Header="HeaderTitle" DisplayMemberBinding="{Binding
Path}"></GridViewColumn>
    </GridView>
  </ListView.View>
</ListView>

<ListView>
  <ListView.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Path}"></TextBlock>
      </StackPanel>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

---

Все элементы управления списками имеют свойство `ItemTemplate`, которое позволяет задать свой шаблон отображения данных. В качестве значения оно принимает объект `DataTemplate`.

### 1.2.5 DataGrid

`DataGrid` во многом похож на `ListView`, но более сложный по характеру и допускает редактирование содержимого таблицы.

Свойства:

`AutoGenerateColumns` — Значение `true`, если столбцы должны создаваться автоматически; в противном случае — значение `false`. Значение по умолчанию — `true`.

`ItemsSource` — возвращает или задаёт коллекцию.

`HeadersVisibility` — Задаёт видимость заголовков.

`DataGrid.Columns` — Коллекция столбцов для отображения

`DataGridTextColumn` — Отображает элемент `TextBlock` или `TextBox` при редактировании.

`DataGridHyperlinkColumn` — Представляет гиперссылку и позволяет переходить по указанному адресу.

`DataGridCheckBoxColumn` — Отображает элемент `CheckBox`.

`DataGridComboBoxColumn` — Отображает выпадающий список - элемент `ComboBox`.

`DataGridTemplateColumn` — Позволяет задать специфичный шаблон для отображения столбца.

`RowDetailsTemplate` — Шаблон, для дополнительной информации строки.

`IsReadOnly` — Получает или задает значение, позволяющее определить, можно ли пользователю вносить изменения в значения в `DataGrid`.

### 1.2.6 Calendar и DatePicker

`Calendar` представляет собой элемент в виде календаря, тогда как `DatePicker` - текстовое поле для ввода даты с выпадающим календарем после ввода.

### 1.2.7 Image

Элемент `Image` предназначен для работы с изображениями. Свойство `Source` позволяет задать путь к изображению

## 2. Стили

Стили позволяют определить набор некоторых свойств и их значений, которые потом могут применяться к элементам в xaml. Стили хранятся в ресурсах и отделяют значения свойств элементов от пользовательского интерфейса.

Пример простых стилей:

---

```
<Style x:Key="DefaultWindow" TargetType="Window">
    <Setter Property="FontFamily" Value="Georgia"></Setter>
    <Setter Property="FontSize" Value="14"></Setter>
    <Setter Property="MinHeight" Value="480"></Setter>
    <Setter Property="MinWidth" Value="640"></Setter>
</Style>
<Style x:Key="DefaultPage" TargetType="Page">
    <Setter Property="FontFamily" Value="Georgia"></Setter>
    <Setter Property="FontSize" Value="14"></Setter>
</Style>
<Style x:Key="DefaultButton" TargetType="Button">
    <Setter Property="Height" Value="30"></Setter>
</Style>
    <Style x:Key="BackButton" TargetType="Button">
        <Setter Property="Height" Value="30"></Setter>
        <Setter Property="Width" Value="150"></Setter>
        <Setter Property="Margin" Value="10"></Setter>
    </Style>
```

---

Свойства:

TargetType — Тип элемента, для которого будут написаны стили.

BasedOn — Позволяет наследовать и расширять дочерние стили.

## 3 Триггеры

Триггеры позволяют декларативно задать некоторые действия, которые выполняются при изменении свойств стиля. Существует следующие виды триггеров:

1. Триггеры свойств: вызываются в ответ на изменения свойствами зависимостей своего значения
2. Триггеры данных: вызываются в ответ на изменения значений любых свойств (они необязательно должны быть свойствами зависимостей)
3. Триггеры событий: вызываются в ответ на генерацию событий
4. Мультитриггеры: вызываются при выполнении ряда условий

### 3.1 Триггеры свойств

---

```
<Button
Height="30"
Width="150"
Name="BtnSample"
Click="BtnSample_OnClick"
Content="Нажми">
  <Button.Style>
    <Style>
      <Style.Triggers>
        <Trigger Property="Button.IsMouseOver" Value="True">
          <Setter Property="Button.FontSize" Value="14" />
          <Setter Property="Button.Foreground" Value="Red" />
        </Trigger>
      </Style.Triggers>
    </Style>
  </Button.Style>
</Button>
```

---

### 3.2 Триггеры данных

С помощью свойства Binding триггер данных устанавливает привязку к отслеживаемому свойству.

---

```
<DataGrid.RowStyle>
  <Style TargetType="DataGridRow">
    <Style.Triggers>
      <DataTrigger Binding="{Binding Active}" Value="False">
        <Setter Property="Background" Value="Red"></Setter>
      </DataTrigger>
      <DataTrigger Binding="{Binding RoleID}" Value="1">
```

```
        <Setter Property="Background" Value="Green"></Setter>
    </DataTrigger>
</Style.Triggers>
</Style>
</DataGrid.RowStyle>
```

---

### 3.3 Триггеры событий

---

```
<Style.Triggers>
    <EventTrigger RoutedEvent="Click">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <!--Анимация-->
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
</Style.Triggers>
```

---

### 3.4 Мультитриггеры

---

```
<DataGrid.RowStyle>
    <Style TargetType="DataGridRow">
        <Style.Triggers>
            <MultiDataTrigger>
                <MultiDataTrigger.Conditions>
                    <Condition Binding="{Binding Active}" Value="False"></Condition>
                    <Condition Binding="{Binding RoleID}" Value="1"></Condition>
                </MultiDataTrigger.Conditions>
                <MultiDataTrigger.Setters>
                    <Setter Property="Background" Value="Red"></Setter>
                </MultiDataTrigger.Setters>
            </MultiDataTrigger>
        </Style.Triggers>
    </Style>
</DataGrid.RowStyle>
```

---

## 4 Темы

В менеджере ссылок, есть некоторые темы для WPF приложений.

Например Aero. После добавление ссылки, нужно добавить стили в App.xaml.

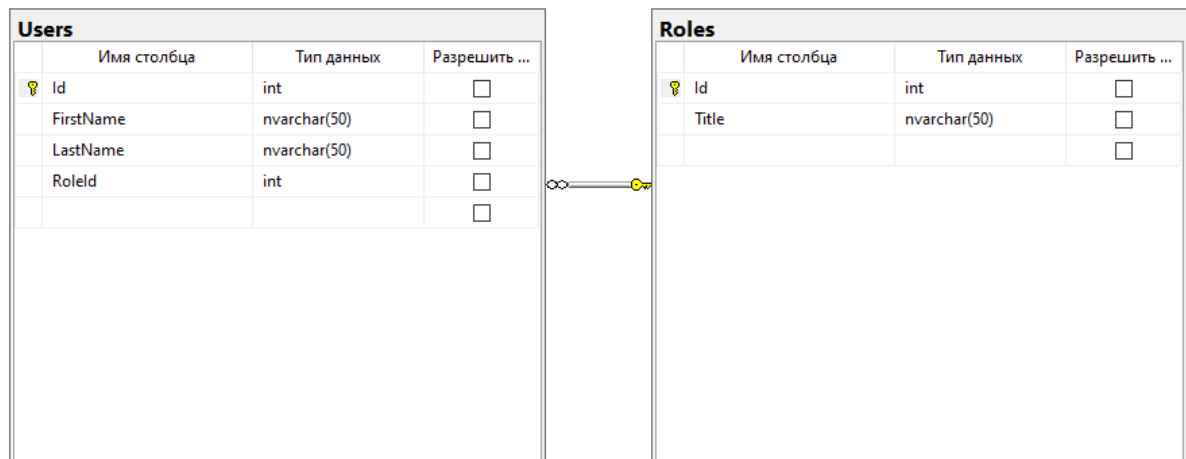
---

```
<ResourceDictionary>
  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
      Source="/PresentationFramework.Aero,Version=4.0.0.0,Culture=neutral,PublicKeyToken=31bf38
      56ad364e35;component/themes/Aero.NormalColor.xaml"/>
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
```

---

## 5 Работа с данными

Создадим базу данными по следующей схеме:

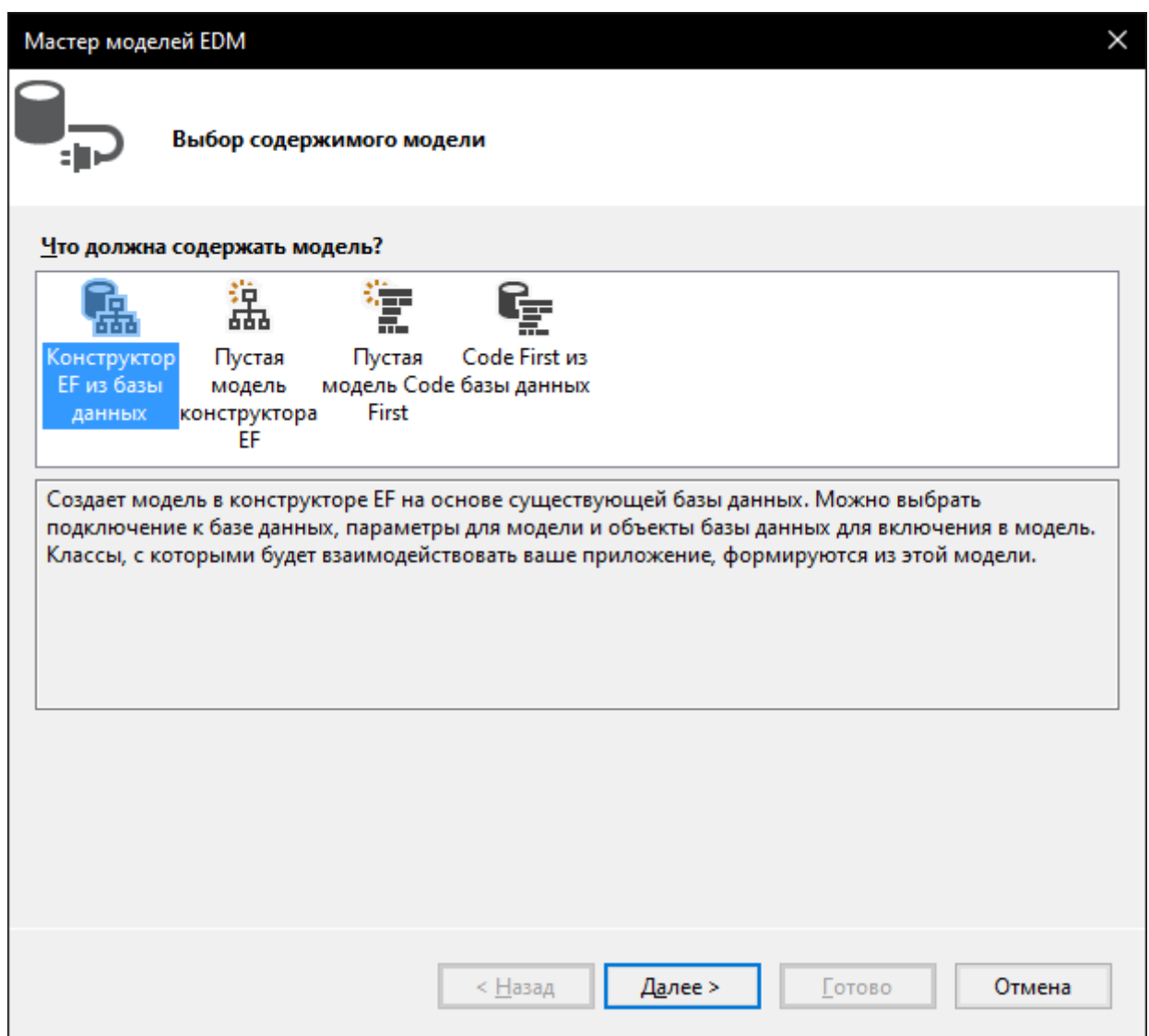


Добавим тестовые данные, и напомним CRUD приложение используя привязку данных и контекст данных.

Чтобы обрабатывать данных из базы данных, есть ORM (Object Relational Mapping), Entity Framework, всё сделает за нас, лишь надо настроить подключение к базе данных.

Есть и другие методы проектирования бд с использованием EF и подхода Code First — когда создаются сущности на основе классов, описывается контекст и т.д, а потом на основе этого контекста создается база данных.

Для того, чтобы подключить модель, надо Проект —>Добавить—>Создать элемент—>Данные—>Модель ADO.NET EDM.



Выбираем конструктор EF из базы данных, этот конструктор создаст классы и настроит DataContext со строкой подключения.



Мастер моделей EDM

Выбор подключения к данным

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

desktop-ob3vg27.Black.dbo

Создать соединение...

Возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

```
metadata=res://*/Model1.csdl|res://*/Model1.ssdl|
res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data
source=DESKTOP-OB3VG27;initial catalog=Black;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Сохранить параметры соединения в App.Config как:

BlackEntities

< Назад    Далее >    Готово    Отмена

Выбираем создать соединение.

Свойства подключения

Введите данные для подключения к выбранному источнику данных или нажмите кнопку "Изменить", чтобы выбрать другой источник данных и (или) поставщик.

Источник данных:  
Microsoft SQL Server (SqlClient) Изменить...

Имя сервера:  
DESKTOP-OB3VG27 Обновить

Вход на сервер

Проверка подлинности: Проверка подлинности Windows

Имя пользователя:

Пароль:

☐ Сохранить пароль

Подключение к базе данных

☒ Выберите или введите имя базы данных:  
Sample

☐ Прикрепить файл базы данных:  
 Обзор...

Логическое имя:


Дополнительно...

Проверить подключение ОК Отмена

Указываем имя сервера, настраиваем проверку подлинности, если такая имеется и выбираем имя бд.

Мастер моделей EDM

✕



Выбор подключения к данным

**Какое подключение к данным будет использоваться приложением для подключения к базе данных?**

desktop-ob3vg27.Sample.dbo ▾

Создать соединение...

Возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☐ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

metadata=res://\*/Model1.csdl|res://\*/Model1.ssdl|  
res://\*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data  
source=DESKTOP-OB3VG27;initial catalog=Sample;integrated  
security=True;MultipleActiveResultSets=True;App=EntityFramework"

☒ Сохранить параметры соединения в App.Config как:

SampleEntities

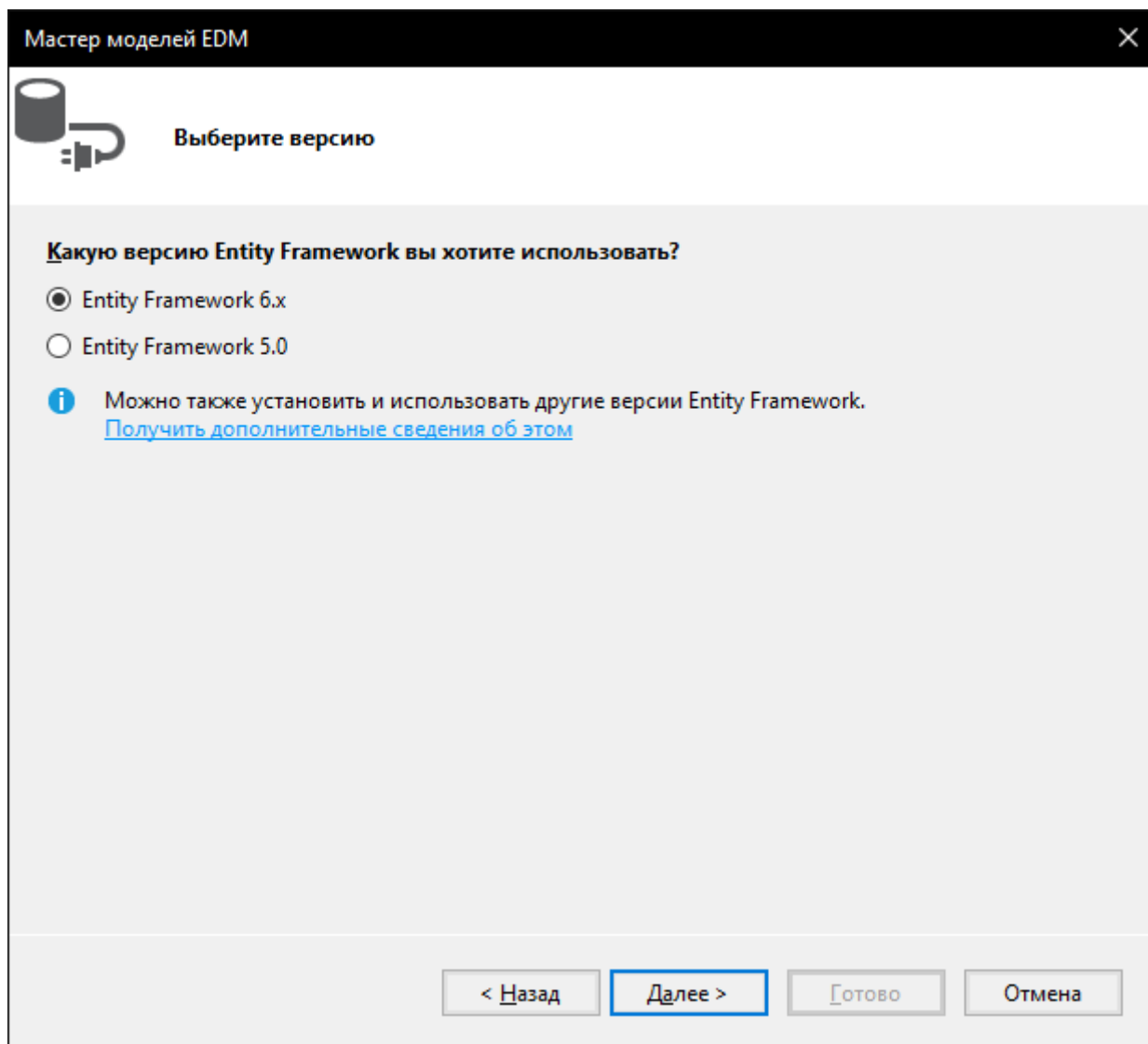
< Назад

Далее >

Готово

Отмена

Нажимаем далее.



Выбираем актуальную версию EF.

Мастер моделей EDM

Выберите параметры и объекты базы данных

Какие объекты базы данных нужно включить в модель?

- ☒ Таблицы
  - ☒ dbo
    - ☒ Roles
    - ☐ sysdiagrams
    - ☒ Users
  - ☐ Представления
  - ☐ Хранимые процедуры и функции

☐ Формировать имена объектов во множественном или единственном числе

☒ Включить столбцы внешних ключей в модель

☒ Импортировать выбранные хранимые процедуры и функции в модель сущностей

Пространство имен модели:

SampleModel

< Назад    Далее >    Готово    Отмена

Выбираем нужные таблицы, без диаграммы.

## 5.1 Привязка данных и контекст данных.

Для создания привязки применяется элемент Binding и его свойства:

ElementName — имя элемента, к которому идет привязка. Если мы говорим о привязке данных, то данное свойство задействуется редко за исключением тех случаев, когда данные определены в виде свойства в определенном элементе управления

Path — ссылка на свойство объекта, к которому идет привязка

Source — ссылка на источник данных, который не является элементом управления

Свойства элемента Binding помогают установить источник привязки. Для установки источника или контекста данных в элементах управления WPF предусмотрено свойство DataContext.

Пример приложения:

Helper.cs

---

```
public class Helper
{
    public static SampleEntities context = new SampleEntities();
}
```

---

Содержит экземпляр контекста.

MainWindow.xaml

---

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="50"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal">
        <Button
            Height="30"
            Width="150"
            Name="BtnAdd"
            Click="BtnAdd_OnClick"
            Margin="10">Добавить</Button>
        <Button
            Height="30"
            Width="150"
            Name="BtnUpdate"
            Click="BtnUpdate_OnClick"
            Margin="10">Обновить</Button>
        <Button
            Height="30"
            Width="150"
            Name="BtnRemove"
            Click="BtnRemove_OnClick"
            Margin="10">Удалить</Button>
    </StackPanel>
    <DataGrid
        Grid.Row="1"
        Name="UsersDataGrid"
        AutoGenerateColumns="False"
        ColumnWidth="*"
        IsReadOnly="True">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Фамилия" Binding="{Binding
LastName}"></DataGridTextColumn>
            <DataGridTextColumn Header="Имя" Binding="{Binding
FirstName}"></DataGridTextColumn>
```

```
                <DataGridTextColumn Header="Роль" Binding="{Binding
Roles.Title}"></DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
```

---

## MainWindow.xaml.cs

---

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        Load();
    }

    private void Load()
    {
        UsersDataGrid.ItemsSource = Helper.context.Users.ToList();
    }

    private void BtnAdd_OnClick(object sender, RoutedEventArgs e)
    {
        new AddOrUpdateWindow().ShowDialog();
        Load();
    }

    private void BtnUpdate_OnClick(object sender, RoutedEventArgs e)
    {
        if (UsersDataGrid.SelectedItem is Users user)
        {
            new AddOrUpdateWindow(user).ShowDialog();
            Load();
        }
    }

    private void BtnRemove_OnClick(object sender, RoutedEventArgs e)
    {
        if (UsersDataGrid.SelectedItem is Users user)
        {
            Helper.context.Users.Remove(user);
            Helper.context.SaveChanges();
            Load();
        }
    }
}
```

---

Здесь реализован вывод в DataGrid, переходы на формы добавления и обновления информации о пользователях, а также удаление пользователей.

## AddOrUpdateWindow.xaml

---

```

<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <TextBlock
        HorizontalAlignment="Right"
        Margin="10"
        VerticalAlignment="Center">Фамилия</TextBlock>
    <TextBlock
        Grid.Row="1"
        HorizontalAlignment="Right"
        Margin="10"
        VerticalAlignment="Center">Имя</TextBlock>
    <TextBlock
        Grid.Row="2"
        HorizontalAlignment="Right"
        Margin="10"
        VerticalAlignment="Center">Роль</TextBlock>
    <TextBox
        Grid.Column="1"
        VerticalAlignment="Center"
        Margin="10"
        Text="{Binding LastName}"></TextBox>
    <TextBox
        Grid.Column="1"
        Grid.Row="1"
        VerticalAlignment="Center"
        Margin="10"
        Text="{Binding FirstName}"></TextBox>
    <ComboBox
        Grid.Column="1"
        Grid.Row="2"
        Margin="10"
        VerticalAlignment="Center"
        Name="CmbRoles"
        DisplayMemberPath="Title"
        SelectedIndex="0"
        SelectedItem="{Binding Roles}"
        Text="{Binding Roles.Title}"></ComboBox>
    <Button
        Grid.Column="1"
        Grid.Row="3"
        Height="30"
        Width="150"
        Name="BtnSubmit"
        Click="BtnSubmit_OnClick">Подтвердить</Button>
</Grid>

```

---

## AddOrUpdateWindow.xaml.cs

---

```

public partial class AddOrUpdateWindow : Window
{

```



```

public AddOrUpdateWindow()
{
    InitializeComponent();
    Load();
    DataContext = new Users();
}

public AddOrUpdateWindow(Users user)
{
    InitializeComponent();
    Load();
    DataContext = user;
}

private void Load()
{
    CmbRoles.ItemsSource = Helper.context.Roles.ToList();
}

private void BtnSubmit_OnClick(object sender, RoutedEventArgs e)
{
    if (DataContext is Users user && user.Id == 0)
    {
        Helper.context.Users.Add(user);
    }
    Helper.context.SaveChanges();
    Close();
}
}

```

---

Здесь используется перегрузка конструкторов, один без параметров для добавления, а второй для обновления информации о передаваемом пользователе. Событие кнопки подтвердить, выполняет проверку, является ли объект привязки Users и равен ли Id 0, если эти условия удовлетворяются, то объект добавляется к коллекции Users, после сохраняются изменения. Если обновляем информацию, то Id у объекта, не равен 0, происходит просто сохранения привязанной информации.

## 6 Создание отчётов

### 6.1 Excel

Для того, чтобы использовать встроенные библиотеки, переходим в менеджер ссылок—> COM—> Microsoft Excel 16.0 Object Library.

P.S 16.0 Версия файла, может не совпадать.

Добавим зависимость, чтобы было проще работать.

`using Excel = Microsoft.Office.Interop.Excel;`

Пример простого Excel отчёта.

---

```
SaveFileDialog saveFile = new SaveFileDialog();
saveFile.FileName = "report";
saveFile.Filter = "Excel files |*.xlsx";
if (saveFile.ShowDialog() == true)
{
    Excel.Application app = new Excel.Application();
    Excel.Workbook workbook = app.Workbooks.Add();
    Excel.Worksheet worksheet = app.Worksheets[1];
    worksheet.Name = "Пользователи";
    worksheet.Range["A1"].Value = "Фамилия";
    worksheet.Range["B1"].Value = "Имя";
    worksheet.Range["C1"].Value = "Роль";

    List<Users> users = Helper.context.Users.ToList();
    for (int i = 0; i < users.Count; i++)
    {
        if (users[i] != null)
        {
            worksheet.Range["$A{i + 2}"].Value = users[i].FirstName;
            worksheet.Range["$B{i + 2}"].Value = users[i].LastName;
            worksheet.Range["$C{i + 2}"].Value = users[i].Roles.Title;
        }
    }
    workbook.SaveAs(saveFile.FileName);
    workbook.Close();
    app.Quit();
}
```

---

## 6.2 Word и Pdf

Для того, чтобы использовать встроенные библиотеки, переходим в менеджер ссылок—> COM—> Microsoft Word 16.0 Object Library.

P.S 16.0 Версия файла, может не совпадать.

Добавим зависимость, чтобы было проще работать.

`using Word = Microsoft.Office.Interop.Word;`

---

```
SaveFileDialog saveFile = new SaveFileDialog();
saveFile.FileName = "report";
saveFile.Filter = "Docx files |*.docx";
if (saveFile.ShowDialog() == true)
{
    Word.Application app = new Word.Application();
    Word.Document document = app.Documents.Add();
    List<Users> users = Helper.context.Users.ToList();
    Word.Paragraph paragraph = document.Paragraphs.Add();
    Word.Range range = paragraph.Range;
    Word.Table table = document.Tables.Add(range, users.Count+1, 3);
}
```

```

Word.Range cellRange;
cellRange = table.Cell(1, 1).Range;
cellRange.Text = "Фамилия";

cellRange = table.Cell(1, 2).Range;
cellRange.Text = "Имя";

cellRange = table.Cell(1, 3).Range;
cellRange.Text = "Роль";
for (int i = 0; i < users.Count; i++)
{
    cellRange = table.Cell(i + 2, 1).Range;
    cellRange.Text = users[i].LastName;

    cellRange = table.Cell(i + 2, 2).Range;
    cellRange.Text = users[i].FirstName;

    cellRange = table.Cell(i + 2, 3).Range;
    cellRange.Text = users[i].Roles.Title;
}
document.SaveAs2(saveFile.FileName);
document.Close();
app.Quit();
}

```

---

Для того, чтобы сохранить word файл в pdf формате, надо просто добавить формат сохранения.

---

```

SaveFileDialog saveFile = new SaveFileDialog();
saveFile.FileName = "report";
saveFile.Filter = "Pdf files |*.pdf";
if (saveFile.ShowDialog() == true)
{
    Word.Application app = new Word.Application();
    Word.Document document = app.Documents.Add();
    List<Users> users = Helper.context.Users.ToList();
    Word.Paragraph paragraph = document.Paragraphs.Add();
    Word.Range range = paragraph.Range;
    Word.Table table = document.Tables.Add(range, users.Count + 1, 3);

    Word.Range cellRange;
    cellRange = table.Cell(1, 1).Range;
    cellRange.Text = "Фамилия";

    cellRange = table.Cell(1, 2).Range;
    cellRange.Text = "Имя";

    cellRange = table.Cell(1, 3).Range;
    cellRange.Text = "Роль";
    for (int i = 0; i < users.Count; i++)
    {
        cellRange = table.Cell(i + 2, 1).Range;
        cellRange.Text = users[i].LastName;

        cellRange = table.Cell(i + 2, 2).Range;
        cellRange.Text = users[i].FirstName;

        cellRange = table.Cell(i + 2, 3).Range;
    }
}

```

```
        cellRange.Text = users[i].Roles.Title;
    }
    document.SaveAs2(saveFile.FileName, Word.WdSaveFormat.wdFormatPDF);
    document.Close();
    app.Quit();
}
```

---

## 7 Использование элементов управления WinForms в WPF

Для того, чтобы использовать WinForms элементы, переходим в менеджер ссылок—> Ссылки—> WindowsFormsIntegration.

Для большинства элементов управления, достаточно добавить ссылку System.Windows.Forms.

---

```
<WindowsFormsHost>
  <forms:DataGridView></forms:DataGridView>
</WindowsFormsHost>
```

---

Также надо добавить в xaml разметку, зависимость, для использования System.Windows.Forms.

```
xmlns:forms="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

Если в приложение требуется, использование графиков, то есть такая ссылка —> System.Windows.Forms.DataVisualization.

## 8 Drag and Drop

Drag and drop — способ оперирования элементами интерфейса в интерфейсах пользователя при помощи манипулятора «мышь» или сенсорного экрана.

Сделаем примеры использования drag and drop с использованием Image и ListView.

ListView:

---

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <ListView
    Name="ProductsList"
```

```

PreviewMouseDown="Productslist_OnPreviewMouseDown">
<ListView.ItemTemplate>
    <DataTemplate>
        <StackPanel Orientation="Horizontal">
            <TextBlock
                Text="{Binding Title}"
                Margin="5"></TextBlock>
            <TextBlock
                Text="{Binding Price}"
                Margin="5"></TextBlock>
        </StackPanel>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>

```

```

<ListView
    Name="BagList"
    Grid.Column="1"
    AllowDrop="True"
    Drop="BagList_OnDrop">
    <ListView.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock
                    Text="{Binding Title}"
                    Margin="5"></TextBlock>
                <TextBlock
                    Text="{Binding Price}"
                    Margin="5"></TextBlock>
                <TextBlock
                    Text="{Binding Count}"
                    Margin="5"></TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

```

```

</Grid>

```

---

```

public partial class MainWindow : Window
{
    private List<Product> Products = new List<Product>();
    private List<Product> BagProducts = new List<Product>();
    public MainWindow()
    {
        InitializeComponent();
        GetProducts();
        ProductsList.ItemsSource = Products;
    }

    public void GetProducts()
    {
        Products.Add(new Product() { Title = "Apple", Price = 132.0, Count = 0 });
        Products.Add(new Product() { Title = "Banana", Price = 100.0, Count = 0 });
        Products.Add(new Product() { Title = "Potato", Price = 56.0, Count = 0 });
        Products.Add(new Product() { Title = "Pineapple", Price = 250.0, Count = 0
    });
}

```

```

e) private void ProductsList_OnPreviewMouseDown(object sender, MouseButtonEventArgs
{
    ListView listView = sender as ListView;
    if (listView?.SelectedItem != null)
    {
        DragDrop.DoDragDrop(listView, listView.SelectedItem as Product,
DragDropEffects.Copy);
    }
}

private void BagList_OnDrop(object sender, DragEventArgs e)
{
    Product product = (Product)e.Data.GetData(e.Data.GetFormats()[0]);
    bool isExists = BagProducts.Exists(x => x == product);
    if (isExists)
    {
        BagProducts.Find(x => x == product).Count += 1;
    }
    else
    {
        BagProducts.Add(product);
    }
    BagList.ItemsSource = BagProducts;
    BagList.Items.Refresh();
}

private class Product
{
    public string Title { get; set; }
    public double Price { get; set; }
    public int Count { get; set; }
}
}

```

---

Image:

```

<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Image Source="cat.jpg" PreviewMouseDown="UIElement_OnPreviewMouseDown"></Image>
    <Image Grid.Column="1" Name="Img" Source="cat2.jpg" AllowDrop="True"
Drop="Img_OnDrop"></Image>
</Grid>

```

---

```

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

    private void Img_OnDrop(object sender, DragEventArgs e)
    {
        //Пример с переносом картинки с рабочего стола
    }
}

```

```

        foreach (string pic in (string[])e.Data.GetData(DataFormats.FileDrop))
        {
            Img.Source = new BitmapImage(new Uri(pic));
        }
        //

        //Пример с переносом из дрыого Image
        //Img.Source = (BitmapSource)e.Data.GetData(DataFormats.Text);
    }

    private void UIElement_OnPreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Image image = e.Source as Image;
        DataObject data = new DataObject(DataFormats.Text, image.Source);
        DragDrop.DoDragDrop((DependencyObject)e.Source, data, DragDropEffects.Copy);
    }
}

```

---

## 9 Мультимедиа

MediaElement — это элемент WPF, служащий оболочкой функциональности класса MediaPlayer. Подобно всем элементам, MediaElement помещается непосредственно в пользовательский интерфейс. В случае применения MediaElement для воспроизведения аудио его расположение не имеет значения, но если воспроизводится видео, он должен быть размещен там, где планируется отображаться видео-окно.

---

```

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>
    <MediaElement Name="media"></MediaElement>
    <StackPanel HorizontalAlignment="Center" Orientation="Horizontal" Grid.Row="1">
        <Button Height="30" Width="100" Margin="5" Name="BtnOpen"
Click="BtnOpen_Click">Open</Button>
        <Button Height="30" Width="100" Margin="5" Name="BtnPlay"
Click="BtnPlay_Click">Play</Button>
        <Button Height="30" Width="100" Margin="5" Name="BtnPause"
Click="BtnPause_Click">Pause</Button>
        <Button Height="30" Width="100" Margin="5" Name="BtnStop"
Click="BtnStop_Click">Stop</Button>
    </StackPanel>
    <StackPanel Grid.Row="2" Orientation="Horizontal" HorizontalAlignment="Center">
        <TextBlock Name="TbTime"></TextBlock>
    </StackPanel>

```



```

        <Slider Grid.Row="3" Minimum="0" Maximum="10"
ValueChanged="mediaSlider_ValueChanged" Name="mediaSlider"></Slider>
    </Grid>

```

---

```

public partial class MainWindow : Window
{
    private DispatcherTimer timer;
    public MainWindow()
    {
        InitializeComponent();
        media.LoadedBehavior = MediaState.Manual;
        timer = new DispatcherTimer
        {
            Interval = TimeSpan.FromSeconds(1)
        };
        timer.Tick += Timer_Tick;
    }

    private void Timer_Tick(object sender, EventArgs e)
    {
        if (media.Source != null)
        {
            if (media.NaturalDuration.HasTimeSpan)
            {
                TbTime.Text =
                $"{media.Position:mm\\:ss}/{media.NaturalDuration.TimeSpan:mm\\:ss}";
            }
        }
    }

    private void BtnOpen_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        if (openFileDialog.ShowDialog() == true)
        {
            media.Source = new Uri(openFileDialog.FileName);
            mediaSlider.Value = media.Volume;
        }
    }

    private void BtnPlay_Click(object sender, RoutedEventArgs e)
    {
        media.Play();
        timer.Start();
    }

    private void BtnPause_Click(object sender, RoutedEventArgs e)
    {
        if (media.CanPause)
        {
            media.Pause();
            timer.Stop();
        }
    }

    private void BtnStop_Click(object sender, RoutedEventArgs e)
    {
        media.Stop();
        timer.Stop();
    }
}

```

```
private void mediaSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
{
    if (media.Source != null)
    {
        media.Volume = e.NewValue;
    }
}
```

---

## 10 Частичные классы и методы

Частичные классы позволяют иметь определения одного и того же класса в разных файлах, но после компиляции они все определяются в один файл.

Это хорошо использовать, когда используется конструктор EF из базы данных, потому что, если мы будем добавлять методы и свойства в сгенерированные классы, то после обновления или пересоздания модели, добавленное будет утеряно.

---

```
public partial class Users
{
    public void GiveMoney()
    {
        Console.WriteLine($"{FullName} give 100$");
    }

    public string FullName => $"{LastName} {FirstName}";
}
```

---