Окна

Класс Window

Ключевым элементом в системе графического интерфейса в WPF является окно, которое содержит все необходимые элементы управления. Окно в WPF представлено классом **Window**, который является производным от класса ContentControl. Поэтому окно является элементом управления содержимым, и как, к примеру, кнопка, может содержать в себе один дочерний элемент. Как правило, в его качестве выступает один из элементов компоновки, например, Grid.

Класс Window привносит ряд свойств, которые позволяют настроить окно приложения:

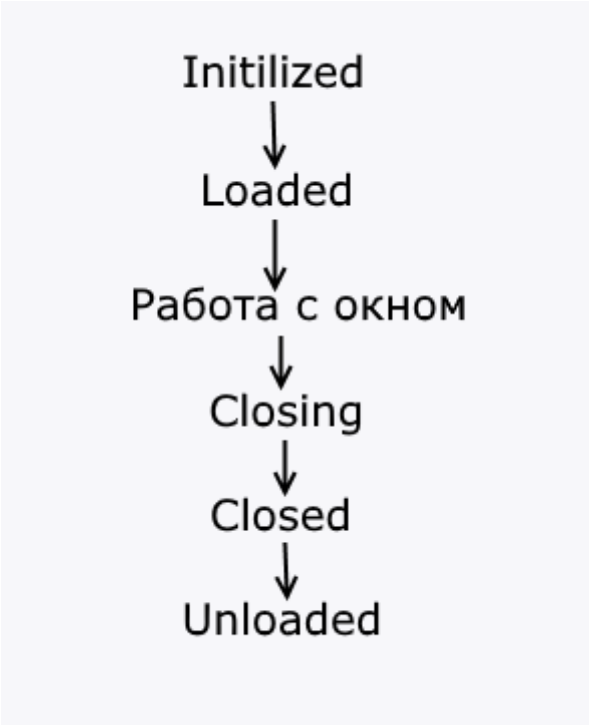
- AllowsTransparency: при значении true позволяет установить прозрачный фон окна
- Icon: представляет иконку, которая отображается в левом верхнем углу окна и в панели задач. Если иконка не установлена, то система будет использовать стандартную иконку по умолчанию.
- Top: устанавливает отступ окна приложения от верхней границы экрана
- Left: устанавливает отступ окна приложения от левой границы экрана
- ResizeMode: задает режим изменения размеров окна. Может принимать следующие значения:
  CanMinimize: окно можно только свернуть
  NoResize: у окна нельзя изменить начальные размеры
  CanResize: у окна можно изменять размеры
  CanResizeWithGrip: в правом нижнем углу окна появляется визуализация того, что у окна можно изменять размеры
- RestoreBounds: возвращает границы окна
- ShowInTaskbar: при значении true иконка окна отображается на панели задач
- SizeToContent: позволяет автоматически масштабировать размеры окна в зависимости от содержимого. Может принимать следующие значения:
  Width: автоматически масштабируется только ширина
  Height: автоматически масштабируется только высота
  WidthAndHeight: автоматически масштабируются высота и ширина
  Manual: автоматическое масштабирование отсутствует
- Title: заголовок окна
- Topmost: при значении true окно устанавливается поверх других окон приложения

- WindowStartupLocation: устанавливает стартовую позицию окна. Может принимать следующие значения:
  CenterOwner: если данное окно было запущено другим окном, то данное окно позиционируется относительно центра запустившего его окна
  CenterScreen: окно помещается в центре экрана
  Manual: позиция устанавливается вручную с помощью свойств Top и Left
- WindowState: состояние окна. Возможные значения:
  Maximized: раскрыто на весь экран
  Minimized: свернуто
  Normal: стандартное состояние

Жизненный цикл

В процессе работы окно в WPF проходит ряд этапов жизненного цикла, которые доступны нам через обработку событий класса Window:

- Initialized: это событие возникает при инициализации окна, когда у него устанавливаются все свойства, но до применения к нему стилей и привязки данных. Это общее событие для всех элементов управления в WPF, поэтому следует учитывать, что сначала возникают события вложенных элементов, а затем их контейнеров. То есть событие Initialized окна приложения генерируется только после того, как отработает событие Initialized для всех вложенных элементов.
- Loaded: возникает после полной инициализации окна и применения к нему стилей и привязки данных. После генерации этого события происходит визуализация элемента, и окно отображается на экране и становится видимым для пользователя
- Closing: возникает при закрытии окна
- Closed: возникает, когда окно становится закрытым
- Unloaded: возникает после закрытия окна при выгрузке всех связанных ресурсов из памяти



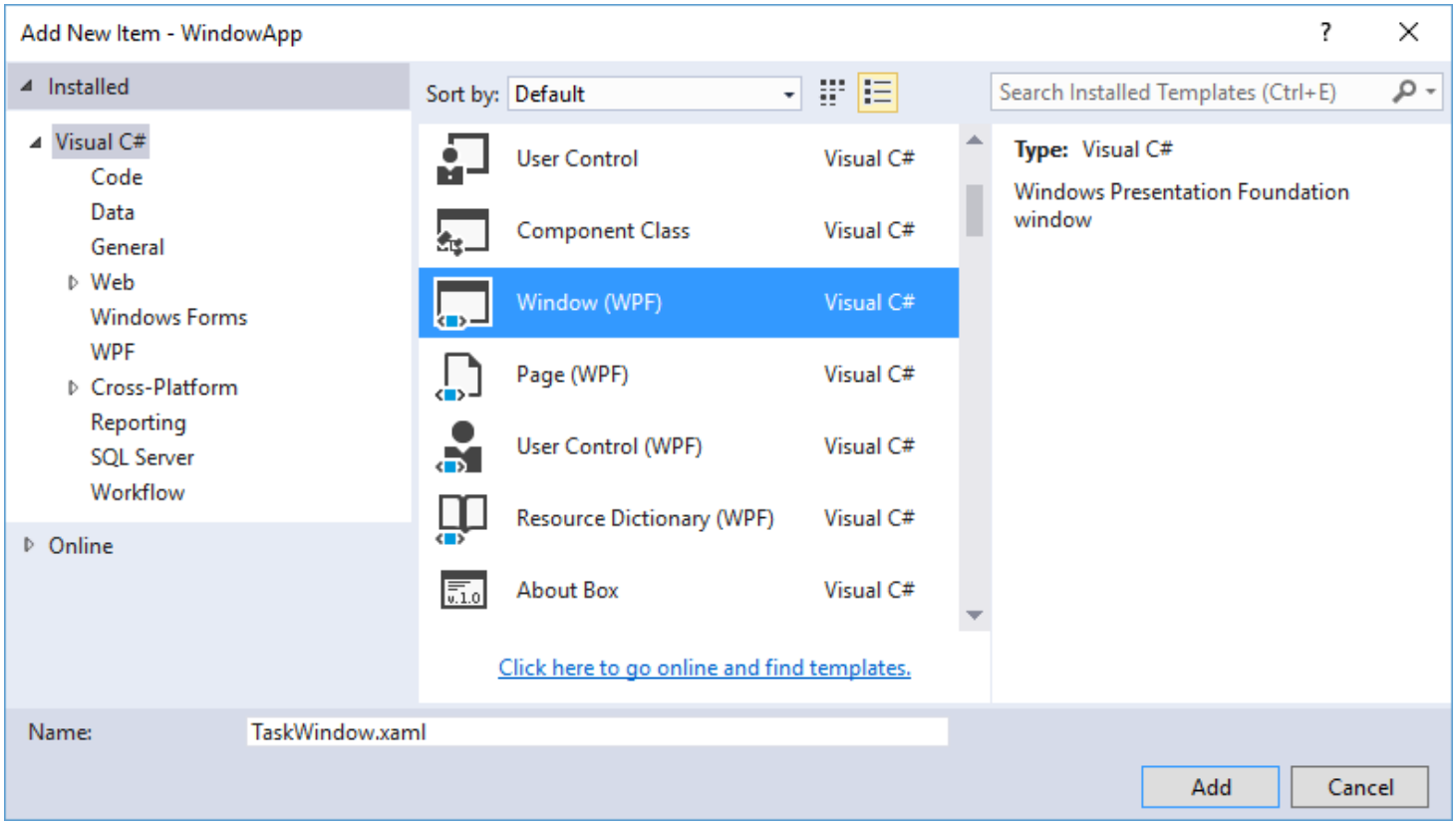
Соответственно, если нам надо выполнить некоторые действия при загрузке или при закрытии окна, мы можем обработать события Loaded и Closing/Closed. Например, запишем в текстовый лог события жизненного цикла:

```
using System;
using System.Windows;
using System.Windows.Media;
using System.IO;

namespace WindowApp
{
    public partial class MainWindow : Window
    {
        string path = "log.txt";
        public MainWindow()
        {
            InitializeComponent();

            this.Loaded += MainWindow_Loaded;
            this.Closing += MainWindow_Closing;
            this.Closed += MainWindow_Closed;
        }

        private void MainWindow_Loaded(object sender, RoutedEventArgs e)
        {
            Log("Loaded");
        }

        private void MainWindow_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            Log("Closing");
        }

        private void MainWindow_Closed(object sender, EventArgs e)
        {
            Log("Closed");
        }

        private void Log(string eventName)
        {
            using (StreamWriter logger = new StreamWriter(path, true))
            {
                logger.WriteLine(DateTime.Now.ToLongTimeString() + " - " + eventName);
            }
        }
    }
}
```

Взаимодействие между окнами

Рассмотрим, как мы можем взаимодействовать с несколькими окнами в WPF. Для этого создадим новый проект. По умолчанию он уже содержит одно главное окно MainWindow. Теперь

добавим еще одно окно. Для этого в окне добавления нового элемента нам надо выбрать тип "Window (WPF)":



Назовем его **TaskWindow**.

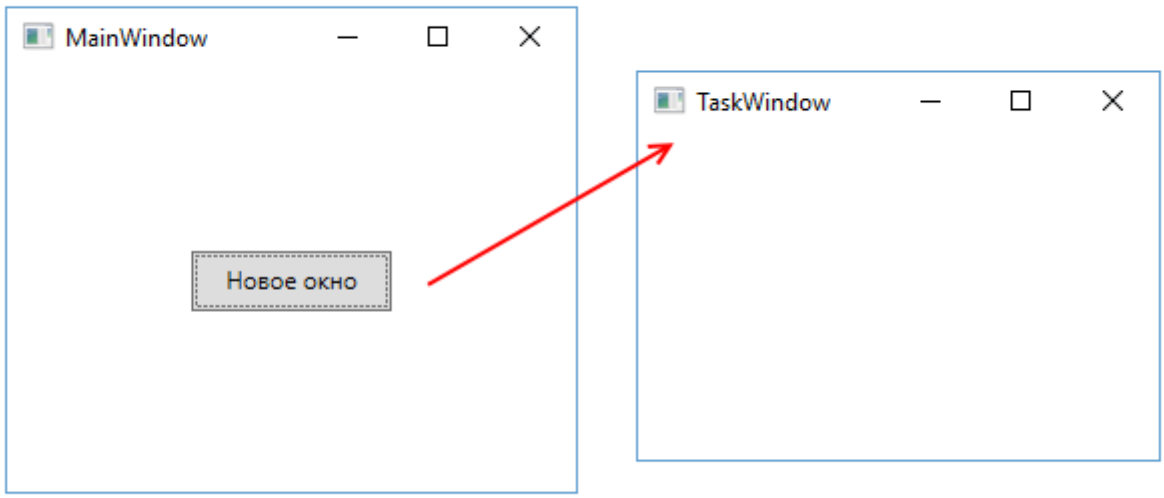
Теперь определим на главном окне MainWindow кнопку для открытия нового окна:

```
<Window x:Class="WindowApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WindowApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Grid>
        <Button Width="100" Height="30" Content="Новое окно" Click="Button_Click" />
    </Grid>
</Window>
```

Обработчик нажатия кнопки Button_Click будет выглядеть так:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    TaskWindow taskWindow = new TaskWindow();
    taskWindow.Show();
}
```

Для открытия нового окна создаем его объект и затем вызываем метод **Show()**.



При нажатии на кнопку открывается окно TaskWindow.

Используя ссылку на окно, мы можем взаимодействовать с ним, например, передавать ему данные из главной формы или вызывать его методы. Например, изменим код C# класса TaskWindow:

```
using System.Windows;

namespace WindowApp
{
    public partial class TaskWindow : Window
    {
        public string ViewModel { get; set; }

        public TaskWindow()
        {
            InitializeComponent();
        }

        public void ShowViewModel()
        {
            MessageBox.Show(ViewModel);
        }
    }
}
```

Здесь добавлено свойство ViewModel и метод, который отображает его содержимое. Теперь изменим обработчик Button_Click в главном окне MainWindow:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    TaskWindow taskWindow = new TaskWindow();
    taskWindow.ViewModel = "ViewModel";

    taskWindow.Show();

    taskWindow.ShowViewModel();
}
```

Здесь у окна TaskWindow устанавливается свойство ViewModel и вызывается его метод.

Важно отметить, что после открытия эти окна существуют независимо друг от друга. Мы можем закрыть главное окно MainWindow, и второе окно TaskWindow все равно продолжит свою работу. Однако мы можем задать и другое поведение.

У всех окон есть свойство **Owner**, которое указывает на главное окно, владеющее текущим окном. Так, изменим обработчик Button_Click в главном окне:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    TaskWindow taskWindow = new TaskWindow();

    //Теперь MainWindow главное окно для taskWindow
    taskWindow.Owner = this;

    taskWindow.Show();
}
```

Теперь текущий объект MainWindow является владельцем taskWindow. Если, к примеру, мы закроем MainWindow, то закроется и TaskWindow.

Кроме того, мы можем обращаться из TaskWindow к своему владельцу:

```
public partial class TaskWindow : Window
{
    public void ChageOwnerBackground()
    {
        this.Owner.Background = new SolidColorBrush(Colors.Red);
    }

    // остальной код
}
```

С другой стороны все зависимые окна доступны в главном окне-владельце через свойство **OwnedWindows**:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    TaskWindow taskWindow = new TaskWindow();
    taskWindow.Owner = this;
    taskWindow.Show();

    foreach(Window window in this.OwnedWindows)
    {
        window.Background = new SolidColorBrush(Colors.Red);

        if (window is TaskWindow)
            window.Title = "Новый заголовок!";
    }
}
```

Класс App и свойство Windows

Еще одним способ для взаимодействия с окнами предоставляет класс App - главный класс приложения. Он содержит свойство **Windows**, которое хранит информацию обо всех открытых окнах приложения. И в любом месте программы мы можем получить эту информацию:

```
foreach(Window window in App.Current.Windows)
{
    window.Background = new SolidColorBrush(Colors.Red);

    // если окно - объект TaskWindow
    if (window is TaskWindow)
        window.Title = "Новый заголовок!";
}
```

Диалоговые окна

WPF поддерживает возможность создания модальных диалоговых окон. При вызове модальное окно блокирует доступ к родительскому окну, пока пользователь не закроет модальное окно.

Для работы добавим в проект новое окно, которое назовем **PasswordWindow**. Это окно будет выполнять роль модального.

Изменим интерфейс PasswordWindow:

```
<Window x:Class="WindowApp.PasswordWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WindowApp"
        mc:Ignorable="d"
        Title="Авторизация" SizeToContent="WidthAndHeight" WindowStartupLocation="CenterScreen">
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="20" />
            <RowDefinition Height="20" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <TextBlock>Введите пароль:</TextBlock>
        <TextBox Name="passwordBox" Grid.Row="1" MinWidth="250">Пароль</TextBox>

        <WrapPanel Grid.Row="2" HorizontalAlignment="Right" Margin="0,15,0,0">
            <Button IsDefault="True" Click="Accept_Click" MinWidth="60" Margin="0,0,10,0">OK</Bu
            <Button IsCancel="True" MinWidth="60">Отмена</Button>
        </WrapPanel>

    </Grid>
</Window>
```

Здесь определено текстовое поле для ввода пароля и две кнопки. Вторая кнопка с атрибутом IsCancel="True" будет выполнять роль отмены. А первая кнопка будет подтверждать ввод.

Для подтверждения ввода и успешного выхода из модального окна определим в файле кода PasswordWindow определим обработчик первой кнопки Accept_Click:

```
using System.Windows;

namespace WindowApp
{
    public partial class PasswordWindow : Window
    {
        public PasswordWindow()
        {
            InitializeComponent();
        }

        private void Accept_Click(object sender, RoutedEventArgs e)
        {
            this.DialogResult = true;
        }

        public string Password
        {
            get { return passwordBox.Text; }
        }

    }
}
```

Для успешного выхода из модального диалогового окна нам надо для свойства **DialogResult** установить значение true. Для второй кнопки необязательно определять обработчик, так как у нее установлен атрибут IsCancel="True", следовательно, ее нажатие будет эквивалентно результату this.DialogResult = false;. Этот же результат будет при закрытии диалогового окна на крестик.

Кроме того, здесь определяется свойство Password, через которое мы можем извне получить введенный пароль.

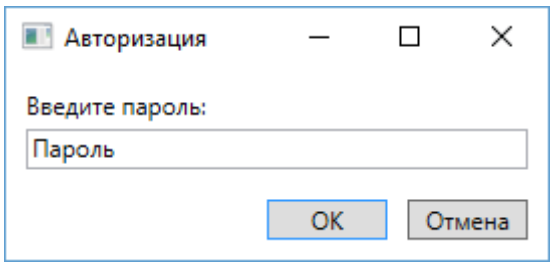
И изменим главную форму MainWindow, чтобы из нее запускать диалоговое окно. Во-первых, определим кнопку:

```
<Window x:Class="WindowApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WindowApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Grid>
        <Button Width="100" Height="30" Content="Авторизация" Click="Login_Click" />
    </Grid>
</Window>
```

И определим обработчик для этой кнопки:

```
private void Login_Click(object sender, RoutedEventArgs e)
{
    PasswordWindow passwordWindow = new PasswordWindow();

    if(passwordWindow.ShowDialog()==true)
    {
        if(passwordWindow.Password=="12345678")
            MessageBox.Show("Авторизация пройдена");
        else
            MessageBox.Show("Неверный пароль");
    }
    else
    {
        MessageBox.Show("Авторизация не пройдена");
    }
}
```

В итоге при нажатии на кнопку будет отображаться следующее диалоговое окно:



И в зависимости от результатов ввода будет отображаться то или иное сообщение.