ШАБЛОН ПРОЕКТА WPF

Фреймворк декларирует разработчику приложения, какие составляющие необходимы для работы программы, а также как эти составляющие связывать. Не имея готового шаблона необходимо было бы вручную, шаг за шагом создавать файлы и внедрять зависимости. Это очень рутинная задача. Более того, при создании можно допустить нелепые низкоуровневые ошибки. Создатели фреймворка организовали удобный шаблон проекта для `VisualStudio`.

> **Важное отличие:** Между .NET (Core) и .NET Framework имеется существенная разница в предоставляемых функциях. .NET (Core) призван быть универсальной библиотекой, поэтому в ней отсутствуют средства взаимодействия с ОС Windows. Единственная причина, по которой можно использовать WPF-проект на .NET (Core) это необходимость использования core-библиотек.

Из этого следует совет - по возможности выбирать именно `.NET Framework`!

Создание проекта

Создание проекта `WPF` проходит базовую процедуру, подобную для других проектов. Учитывайте особенность платформы и выбирайте `.NET Framework`.

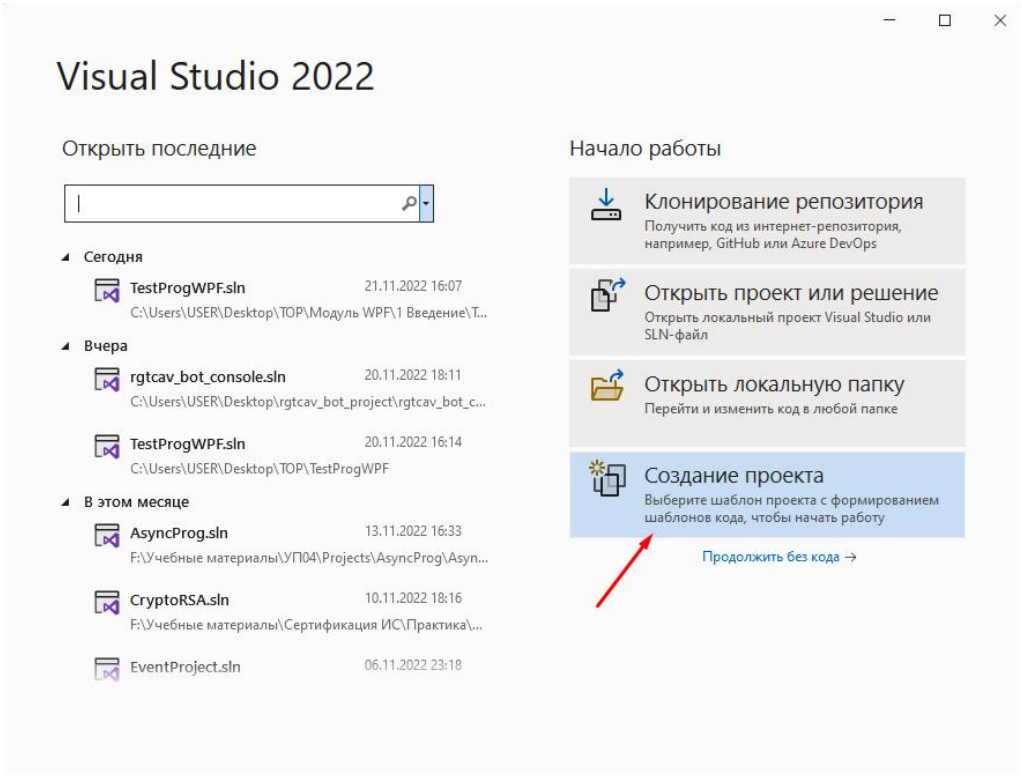

Рисунок 1 - создание проекта

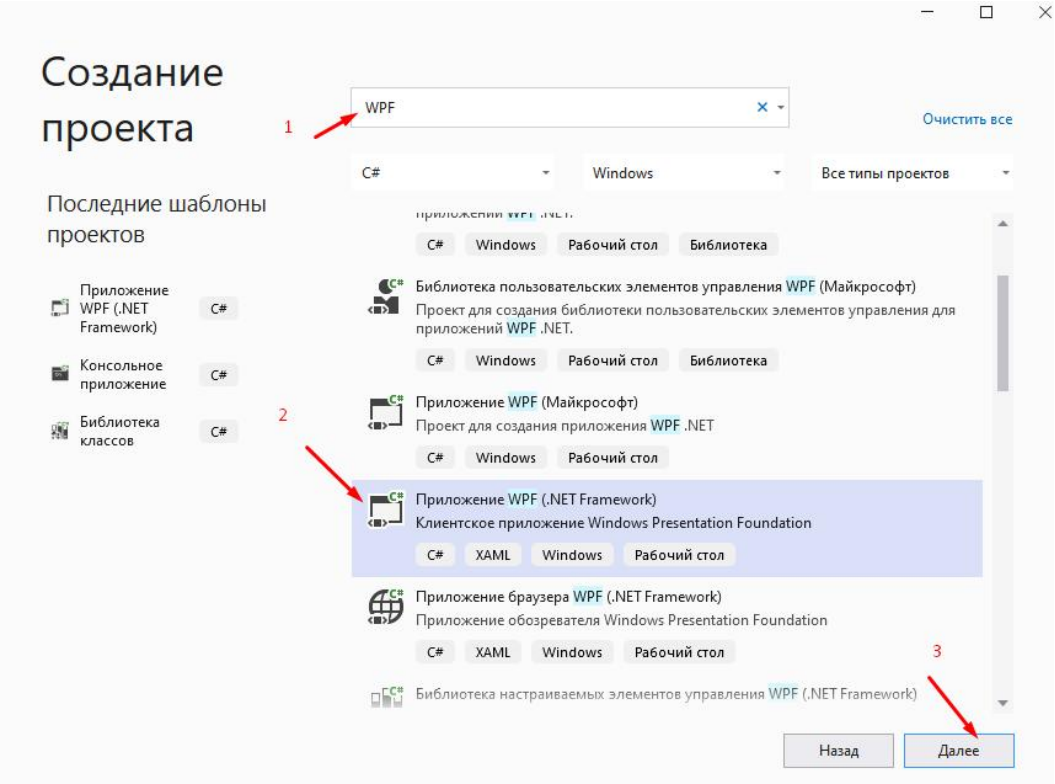

Рисунок 2 - выбор шаблона проекта

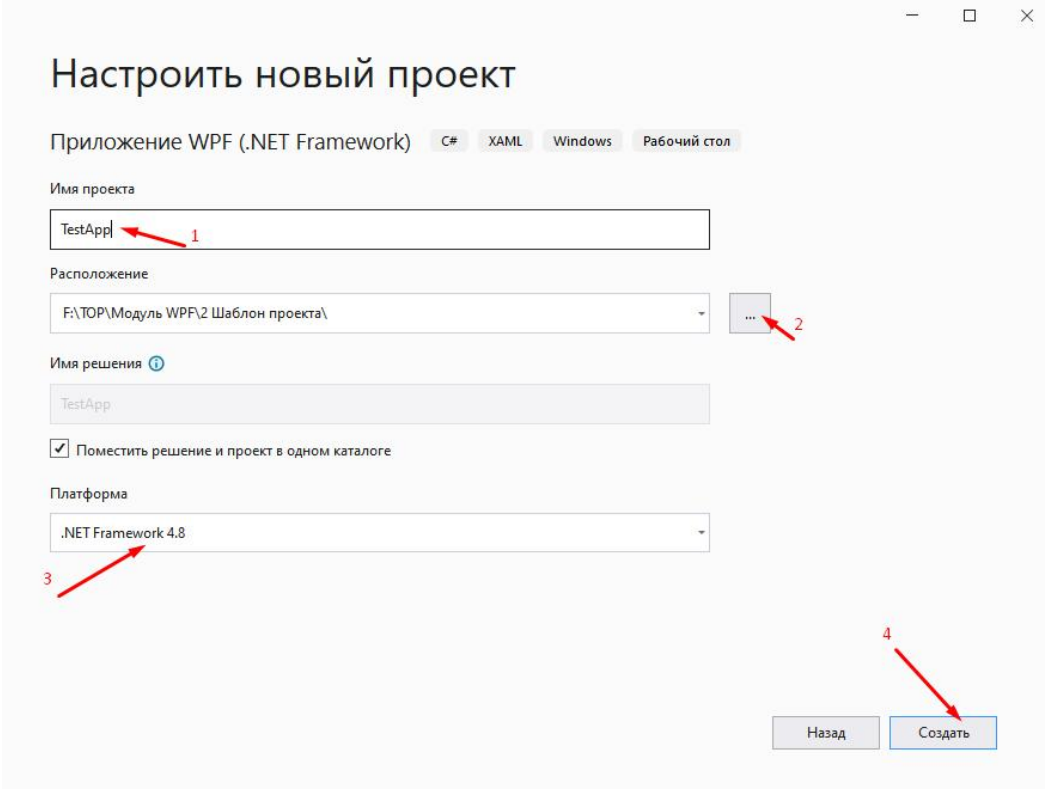

Рисунок 3 - конфигурирование проекта

Обратите внимание на то, что при создании проекта выбирается платформа. Версия платформы выбирается относительно требований к приложению. Например, если вы знаете, что ваше приложение в дальнейшем должно поддерживаться старыми платформами *(Windows 7 стабильно поддерживает версию 4.5, для Windows 10 стабильной является 4.7.2. С требованиями к поддержке фреймворка вы можете ознакомиться в документации* **`Microsoft`***)*.

После создания проекта можно увидеть, что в обозревателе решений уже находятся файлы.

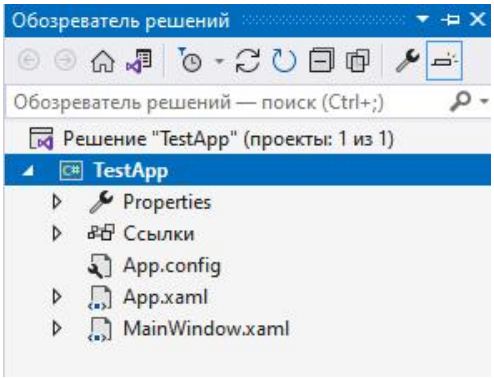

Рисунок 4 - обозреватель решения WPF-проекта

Разбор файлов шаблона

Properties

Во вкладке Properties находятся файлы, представляющие свойства объекта.

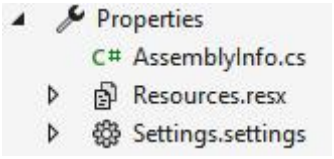

Рисунок 5 - вкладка «Properties»

AssemblyInfo.cs - в этом файле описываются метаданные сборки. Например, компания производителя, мажорная и минорная версия. Примечательно то, что в версии для Core-фремворков большинство свойств отсутствует. Метаданные описываются исходя из конкретной ОС, и в нашем случае, это Windows. Для других ОС WPF не предназначен.

Resources.resx - представляет собой файл, в котором можно хранить ресурсы приложения. Это могут быть ресурсы различных типов данных: строка, изображения и т.п. Эти свойства внедряются в сборку и могут изменяться во время исполнения программы. Более того, после завершения работы приложения, связанные с ним ресурсы сохраняют изменения, и при следующем запуске отображается последнее изменение. Воспринимайте этот файл как банк данных приложения.

Важно отметить, что не стоит хранить неизменяемые элементы файле ресурсов. Это лишняя обёртка, которая будет больше актуальна в приложениях WinForms.Например, храните изображения в файлах проекта, обращаясь к нему по `URI`.

Settings.settings - это файл наследует функционал ресурсов. Он всего лишь семантично отличает используемые ресурсы приложения от ресурсов, представляющих настройки приложения. Если необходимо устанавливать настройку ширины запускаемого окна, лучше воспользоваться этим файлом.

Ссылки

Пункт Ссылки не предоставляет особого интереса для `WPF`. В нём просто указываются зависимости с внешними библиотеками.

App.config

Этот файл представляет собой файл конфигурации приложения **WPF**. В этом файле указывается необходимая для запуска информация, в частности, поддерживаемая версия фреймворка и т.п. Этот файл не компилируется в сборку, а находится рядом с исполняемым файлом. Файл конфигурации так же используют для хранения строки подключения к БД.

App.xaml

Этот файл представляет стартовую точку приложения. Обычно, её предоставляет метод **Main()**. **App.xaml** в очередной раз предоставляет возможность хранить ресурсы приложения. Обычно, в этом файле оформляют подключение словарей стилей. В этом файле так же указываются атрибуты проекта, такие как **StartupUri**, указывающий путь к окну, которое будет запускаться.

```
<Application x:Class="TestProgWPF.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:TestProgWPF"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

Рисунок 6 - структура файла App.xaml

Класс **App.xaml** является ***частичным***. Связанный с ним класс **App.xaml.cs** тоже предоставляет возможность взаимодействия с программой.

```
namespace TestProgWPF
{
    /// <summary>
    /// Логика взаимодействия для App.xaml
    /// </summary>
    Ссылок: 3
    public partial class App : Application
    {

    }
}
```

Рисунок 7 - частичный класс App.xaml

Класс **Application** – это класс, запускающий поток приложения. Как можно было помнить, в приложениях **WPF** используется модель **STA** – *single-thread affinity*.

Посмотрите определение класса **Application** *(для этого, удерживайте* ***ctrl*** *и кликните на класс* ***Application****)*. В нём можно заметить методы **Run()**, **OnStartup()**. Прочитав их описание можно понять, что **Run()** отвечает за запуск **WPF**-приложения, а **OnStartup()** вызывает событие **Startup**, которое возникает при запуске приложения с помощью метода **Run()**.

```
        public static void SetCookie(Uri uri, string value);
        public object FindResource(object resourceKey);
        public int Run(Window window);
        //
        // Сводка:
        //     Запускает Windows Presentation Foundation (WPF) приложения.
        //
        // Возврат:
        //     System.Int32 Приложения код выхода, возвращаемый операционной системе при завершении
        //     работы приложения. По умолчанию код выхода равен 0.
        //
        // Исключения:
        //   T:System.InvalidOperationException:
        //     System.Windows.Application.Run вызывается из приложения, размещенные в браузере
        //     (например, Приложение обозревателя XAML (XBAP)).
        public int Run();
        public void Shutdown();
        public void Shutdown(int exitCode);
        public object TryFindResource(object resourceKey);
        protected virtual void OnActivated(EventArgs e);
        protected virtual void OnDeactivated(EventArgs e);
        protected virtual void OnExit(ExitEventArgs e);
        protected virtual void OnFragmentNavigation(FragmentNavigationEventArgs e);
        protected virtual void OnLoadCompleted(NavigationEventArgs e);
        protected virtual void OnNavigated(NavigationEventArgs e);
        protected virtual void OnNavigating(NavigatingCancelEventArgs e);
        protected virtual void OnNavigationFailed(NavigationFailedEventArgs e);
        protected virtual void OnNavigationProgress(NavigationProgressEventArgs e);
        protected virtual void OnNavigationStopped(NavigationEventArgs e);
        protected virtual void OnSessionEnding(SessionEndingCancelEventArgs e);
        //
        // Сводка:
        //     Вызывает событие System.Windows.Application.Startup.
        //
        // Параметры:
        //   e:
        //     Объект System.Windows.StartupEventArgs, содержащий данные события.
        protected virtual void OnStartup(StartupEventArgs e);
    }
}
```

Рисунок 8 - определение класса Application, от которого наследуется App

Методы запуска приложения можно переопределить в случае необходимости *(например, если ранее надо выполнить какую-нибудь обработку, данные от которой понадобятся для отрисовки формы)*.

Класс **App** изначально пустой. Он формируется при сборке приложения *(из-за его частичной структуры)*, и все конструкторы и доп. Методы добавляются исходя из шаблона автоматически.

Так же, из-за частичной структуры мы не видим метод Main, который можно заметить в обозревателе решений, раскрыв структуру класса.

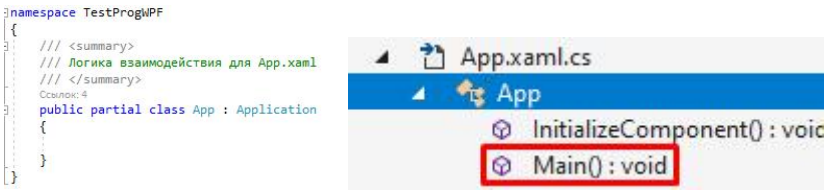

Рисунок 9 - Из-за частичной структуры, метод Main не определён явно в классе App.

MainWindow.xaml

Последний файл – `MainWindow.xaml` и связанный с ним `MainWindow.xaml.cs` – это файлы окна приложения. Окон может быть множество. Этот файл добавляется автоматически. При желании, его можно удалить, изменить его имя и проделать другие манипуляции.

Важно заметить, что к этому окну прописан `StartapUri` в классе `App`. Учитывайте это при разработке форм.

В обозревателе решений, для каждого окна создаётся вложенная структура, где корневой файл это `MainWindow.xaml`, а вложенный - `MainWindow.xaml.cs`. В такой структуре эти файлы отображаются только в обозревателе решений `VisualStudio`.

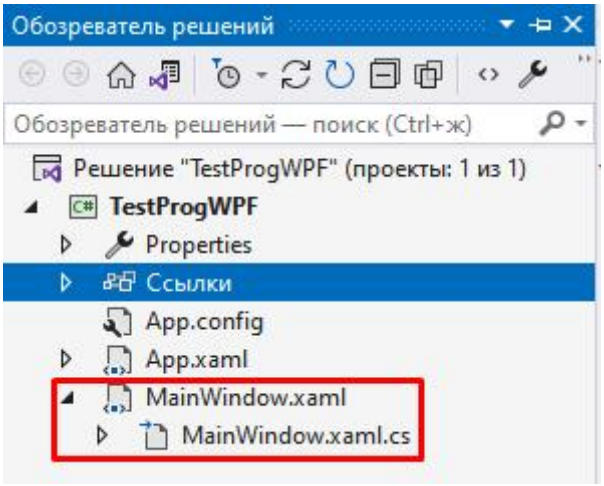

Рисунок 10

Но, как и в случае с любыми ***элементами*** *(не словарями)*, оформленными в файле `*.xaml`, в файловой системе эти файлы отображаются как независимые, существующие отдельно.

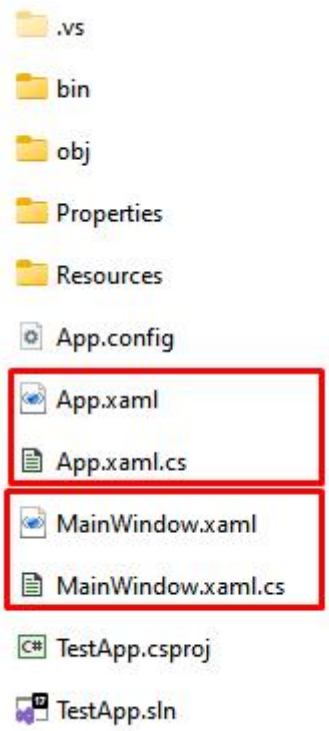

Рисунок 11

Задача указания связи между файлами возложена на фреймворк `WPF`.