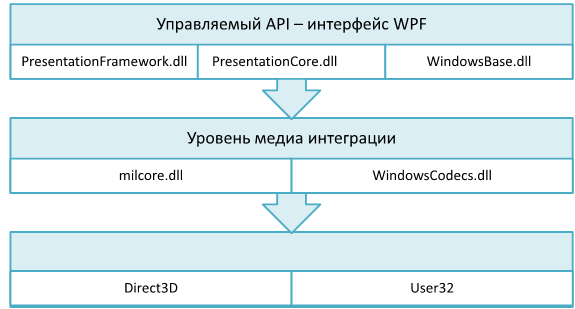
# Intro

* Windows Forms –для визуализации используется User32.dll и GDI/GDI+
* •WPF для визуализации используется DirectX (аппаратная поддержка). В случае со старыми видеокартами используется программное вычисление эффектов.
* Интерфейс в стандартных Windows приложениях не допускает масштабирования. На мониторе с высоким разрешением,размер окна уменьшается и становится менее удобным для работы. (Библиотека User32 не поддерживает полного масштабирования)
* •WPF визуализирует все элементы пользовательского интерфейса самостоятельно, и если кнопка при проектировании в ширину была пять сантиметров, то при выполнении приложения на каком либо разрешении кнопка останется шириной в пять сантиметров. В основу масштабирования ставится системный параметр DPI (DotsPerInch)
* Аппаратно-независимые единицы
* Аппаратно-независимая единица (device-independentunits) составляет 1/96 дюймаи используется для масштабирования окна и его элементов.
* ***[Размер одной единицы] = 1 / 96 дюйма \* [DPI системы]***
* Если создать кнопку с размером 96\*96 то при стандартных настройках системных DPI (96 dotsperinch) кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину ***96***).
* Если DPI системы 120 (дисплей с высоким разрешением)
* ***[Размер одной единицы] = 1 / 96 дюйма \* 120 = 1.24***
* Кнопка в ширину и в высоту будет 1 дюйм (количество пикселей в высоту и ширину ***119***)



System.Threading.DispatcherObject

WPF приложение использует модель STA (Single-ThreadAffinity–однопоточная родственность). Это означает что владельцем всего пользовательского интерфейса является один поток, и любое обращение к элементам управления, созданных в этом потоке, считается не безопасным. DispatcherObject используется для того,что бы координировать обработку сообщений и реализовывать межпоточноевзаимодействие объектов.

System.Windows.DependencyObject

В WPF главный способ взаимодействия с элементами на экране осуществляется черещсвойства. В WPF используется более мощная модель свойств, которая обладает средствами для уведомления об изменениях, наследовании значений по умолчанию, более экономичное хранение свойств. Конечный результат –это свойство зависимостей (dependencyproperty). Классы WPF получают поддержку свойств зависимости за счет порождения от DependencyObject

System.Windows.Media.Visual

Каждый элемент, который отображается в WPF, является производным от класса Visual. Этот класс описывает основную логику для визуализации объекта на окне. Также класс устанавливает связь между управляемыми библиотеками WPF и milcore.dll, которая осуществляет рендеринг.

System.Windows.UIElement

Добавляет поддержку компоновки, ввода, фокуса, событий и команд для объектов.

System.Windows.FrameworkElement

Класс FrameworkElement реализует некоторые члены, которые просто определены в UIElement. Например, UIElementустанавливает фундамент для системы компоновки WPF, но FrameworkElement включает некоторые свойства (такие как HorizontallAlignment и Margin),которые поддерживают его.

System.Windows.Shapes.Shape

От этого класса наследуются базовые фигуры, такие как Rectangle, Polygon, Ellipse, Line, Path.

System.Windows.Controls.Control

Элемент управления –это компонент, который может взаимодействовать с пользователем. Данный класс вводит дополнительные свойства для управлением внешним видом контролов(изменение шрифта, фона и т.д.) Добавляет возможность поддержки шаблонов для быстрого задания стилей элементам управления.

System.Windows.Controls.ContentControl

Это базовый класс для всех элементов управления, которые содержат единственную порцию содержимого. Примеры некоторых производных классов: Label, Button, Window и т.д.

System.Windows.Controls.ItemControl

Класс описывает все элементы управления, которые содержат в себе коллекцию элементов (TreeView, ListBox, ComboBox)

System.Windows.Controls.Panel

Это базовый класс для всех элементов компоновки –объекты способные хранит один и более дочерних элементов.

**XAML**(англ.*e****X****tensible****A****pplication****M****arkup****L****anguage*—расширяемыйязык разметкиприложений произносится (*зэмл)*)—основанный наXMLязык разметки для декларативного программирования приложений, разработанныйMicrosoft.

* + •Каждый элемент в документе ХAML отображается как экземпляр класса .NET. Имя элемента в точности соответствует имени класса.Например элемент <Button> сообщает WPFчто должен быть создан объект Button.
  + •Как и любой XML-документ код XAML допускает вложение одного элемента внутрь другого. Если вы видите элемент Button внутри элемента Grid , то пользовательский интерфейс включает Grid, содержащий внутри себя Button.
  + •Свойства каждого класса можно устанавливать через атрибуты. Тем не менее, в некоторых ситуациях атрибуты не достаточно мощны, чтобы справится с этой работой. В этих случаях понадобятся вложенные дескрипторы со специальным синтаксисом.

Загрузка и компиляция XAML

В общем случае существует три разных стиля кодирования которые могут применятся при создании приложения WPF:

* + - •Только код. Это традиционный подход ,используемы в Visual Studio для приложений Windows Forms. Пользовательский интерфейс в нем генерируется операторами кода.
    - •Код и компилированная разметка (BAML).Это предпочтительный подход для WPF, поддерживаемый Visual Studio.Для каждого окна создается шаблон XAML, и этот код XAML компилируется в BAML,после чего встраивается в конечную сборку. Во время выполнения скомпилированный BAML извлекается и используется для регенерации пользовательского интерфейса.

# Компоновка

* •Окно в WPF может содержать только один элемент.
* •Размеры элементов не должны быть заданы явно.
* •Элементы не отражают свое положение с помощью экранных координат.
* •Контейнеры компоновки разделяют доступное пространство между своими дочерними элементами.
* •Контейнеры компоновки допускают вложения.

Процесс компоновки делится на две стадии:

* + •Измерение -контейнер компоновки просматривает свои дочерние элементы и запрашивает у них предпочтительный размер.
  + •Упорядочивание –контейнер компоновки помещает элементы управления в соответствующие позиции.
* **Grid** – выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки.
*  **StackPanel** –размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна.
*  **DockPanel** – выстраивает элементы по краю контейнера.
*  **WrapPanel** – размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации WrapPanel располагает элементы в строке слева направо, затем переходит к следующей строке. В вертикальной ориентации WrapPanel располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов .
*  **UniformGrid** – помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто.
*  **Canvas** – позволяет элементам позиционироватся по фиксированным координатам. Это неподходящий выбор для окон переменного размера, если только вы не собираететсь взвалить на свои плечи значительный объем работы.
* **InkCanvas** –в WPF также имеется элемент InkCanvas, главное предназначение которого заключается в обеспечении перьевого ввода, а также считывания жестов пользователя.
*  **Z-Порядок** – при наличии более одного перекрывающегося элемента с помощью присоединенного свойства Canvas.Zindex можно управлять их расположением.Обычно все добавляемые элементы имеют одинаковый Zindex-0. Элементы с одинаковым Zindex отображаются в том порядке, в каком они представлены в коллекции Canvas.Children. Который основан на порядке их определения в разметке XAML**.**
*  Вложение контейнеров компоновки. Панели StackPanel, WrapPanel и DockPanel редко используются сами по себе. Вместо этого они применяются для формирования частей интерфейса. Например панель DockPanel можно использовать для размещения разных контейнеров StackPanel и WrapPanel в соответствующих областях окна.

# Биндинг

    Binding binding = new Binding();

    binding.ElementName = "myTextBox"; // элемент-источник

    binding.Path = new PropertyPath("Text"); // свойство элемента-источника

    myTextBlock.SetBinding(TextBlock.TextProperty, binding); // установка привязки для элемента-приемника

# Application

Во время выполнения каждое приложение WPFпредставлено экземпляром класса System.Windows.Application. Этот класс отслеживает все открытые окна в приложении, решает, когда приложение должно быть остановлено, и инициирует события приложения, которые можно обрабатывать для выполнения инициализации и очистки.

В WPF приложение проходит через простой жизненный цикл.

Вскоре после запуска приложения создается объект Application.Во время его выполнения возникают различные события приложения, которые можно отслеживать. И, когда объект приложения освобождается, приложение завершается.

События класса Application

* •Start
* •Exit
* •SessionEnding
* •Activated
* •Deactivated
* •DispatcherUnhandledException

Single Thread Apartment

Однопоточные апартаменты –модель исполнения которая поддерживается в WPF.

* •Элементы WPF обладают *потоковым родством (thread affinity)*. Поток, который создал элементы, владеет ими, а другие потоки не могут с ними взаимодействовать непосредственно.
* •Объекты WPF,которые обладают потоковым родством, порождены от DispatcherObject
* Dispatcher
* Диспетчерэто экземпляр класса System.Windows.Threading.Dispatcher он
* управляет работами, которые возникают во время выполнения WPF приложения. Диспетчер владеет потоком приложения и управляет очередью задач.