XAML

Extensible Application Markup Language (расширяемый язык разметки приложений) – представляет собой язык, использования и создания экземпляров объектов .NET, главный назначением которого является – конструирование пользовательского интерфейса.

Особенности XAML

Жесткое разделение между графической и логической частью. Что позволяет осуществлять проектирование по отдельности

Графический интерфейс пользователя до WPF

В традиционных технологиях отображения не существовала простого способа отделения графического содержимого от кода программы. Ключевая проблема WinForms, состоит в том, что каждая форма, которую вы создаете, целиком определяется в коде. Из-за чего вытекают следующие проблемы:

- Каждый графический элемент должен экспортироваться как отдельное растровое изображение. Это огранивает возможности их комбинирования и применения динамических эффектов, такое как сглаживание, тени и т.д.

- Значительная часть логики пользовательского интерфейса должна быть встроена в код разработчиком. Сюда относятся размеры кнопок, позиционирование, эффекты от перемещения курсора, анимация и т.д.

- Не существует внутренней связи между разными графическими элементами, так что легко создать не соответствующие друг другу наборы изображений. Отслеживание всех этих элементов приносит доп.сложность.

Графический интерфейс пользователя до WPF

- Растровые изображения не могут изменять в размерах без потери качества. По этой причине пользовательский интерфейс на основе растрового изображения зависит от разрешения, следовательно не может адаптироваться к большому разрешению экрана.

Обычно дизайнеры приходилось подготавливать макет, которые позднее мучительно внедрялся в проект

Разновидности XAML

- WPF XAML включает элементы, описывающее содержимое WPF

- XPS XAML часть WPF XAML , определяющая XML-представление форматирования электронных документов

- SilverLightXAML – подмножество WPF XAML, SilverLight-приложений. SilverLight – межплатформенных браузерный подключаемый модуль, который позволяет создавать расширенное веб-содержимое с двумерной графикой, анимацией, аудио и видео.

Компиляция XAML

XAML нужен не только для решения проблемы совместного проектирования, он также должен быть быстрым. Но XML задуман как непротиворечий, читабельный, прямолинейный, но не компактный формат.

В WPF этот недостаток преодолевает по средствам BAML (Binary Application Markup language - язык двоичной разметки приложений). Когда вы компилируете приложение WPF в VS, все файлы XAML преобразуются в BAML, и этот код позже встраивается в виде ресурса в финальную сборку DLL или EXE. Язык BAML поддерживает лексемы, т.е. длительный фрагменты XAML заменены короткими лексемами. И код BAML не только существенно меньше, но и оптимизирован, чтобы быстрее интерпретироваться во время выполнения

Основы XAML

Стандарт XAML достаточно очевиден, если понять несколько его основополагающих правил:

- Каждый элемент в документе XAML отображается на экземпляр класса .NET. Имя элемента в точности соответствует имени класса. Например, элемент <Button> сообщает WPF, что нужно создать объект Button.

- XAML допускает вложение одного элемента внутрь другого.

- Свойства каждого класса можно установить через атрибуты.

Пространство имен XAML

Ясно, что не достаточно просто указать имя класса. Анализаторы XAML также нужно знать пространство имен, котором находится данных класс.

- xmlns – специальный атрибут в мире XML, в который зарезервирован для объявления пространств имен

- http://schemas.microsoft.com/winfx/2006/xaml/presentaion - основное пространство имен WPF. Оно охватывает все классы WPF. Включая элементы управления, которые применяется для управления пользовательского интерфейса. Данное пространство имен объявляется без префикса, поэтом данное пространство имен устанавливается по умолчанию, для данного элемента. Другими словами, каждый элемент автоматически помещается в это пространство имен, если только не указано другие.

Пространство имен XAML

- http://schemas.miicrosoft.com/winfx/2006/xaml - пространство имен XAML. Оно включает различные служебные свойства XAML, которые позволяют влиять на то, как интерпретируется атрибут. Данное пространство имен отображается на префикс **x:**. То есть будет осуществлять поддержка дополнительных встроенных функций.

Пространства имен XML не соответствуют какому-либо конкретному пространству имен .NET. Существует несколько причин, по которым создатели XML выбрали такое проектное решение. По существенному соглашению XML часто имеет форму URI. Формат Uri используется потому, что он делает маловероятным ситуацию, когда разные организации непреднамеренно создают разные языки XML с одинаковыми пространствами имен.

Пространство имен XAML

Другая причина отсутствие отображения «один к одному» между пространствами имен XML и пространствами имен .NET заключается в том, что это могло бы значительно усложнить документы XAML. WPF содержит свыше десятка пространств имен (все начинается с System.Windows). Если каждое пространство имен .NET отображалось на отдельное пространство имен XML, происходила бы путаница и пришлось бы указывать конкретное пространство имен для каждого элемента.

Поэтому разработчики сочли нужно скомбинировать все эти пространства имен .NET в единое пространство XML.

Информация ПИ позволяет анализатору XAML находить правильный класс. Например, когда он просматривает элементы Windows и Grid , то видит, что они помещены в ПИ WPF по умолчанию. Затем он ищет соответствующее пространство имен .NET – до тех пор, пока не находит System.Windows.Window и System.Windows.Controls.Grid

Класс отдельного кода

Язык XAML позволяет конструировать пользовательский интерфейс, но для создания функционального приложения необходим способ подключения обработчиков событий. XAML позволяет легко это сделать с помощью атрибута Class. Префикс **:X** помещает Class в ПИ XAML. Фактически он сообщает анализатору, чтобы он сгенерировал новый класс

Метод InitializeComponent()

Класс Windows* не содержит реальной функциональности, однако он вызывает метод InitializeComponent() по умолчанию, который генерирует все компоненты по умолчанию т.е. вызов метода LoadComponent() класса System.Windows.Application. Метод LoadComponent() извлекает код BAML из сборки и использует его для построение пользовательского интерфейса.

Именование элементов

В классе отдельного кода часто требуется манипулировать элементами управления. Например, необходимо читать либо изменять свойства отдельного элемента. Чтобы использовать такую возможность, элемент управления должен включать атрибут **Name**

Простые свойства и конвертеры типы

Рассмотрим следующий пример

```
<TextBox Name="txtQuestion"
  VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
  FontFamily="Verdana" FontSize="24" Foreground="Green" ... >
```

Чтобы все это заработало, класс System.Windows.Controls.TextBox дожжён предоставить следующий свойства VerticalAligment, HorizontalAligment, FontFamily, FontSize и Foreground.

Чтобы преодолеть зазор между строковыми значениями и не строковыми свойствами, анализатор XAML должен выполнить преобразование, осуществляемое *конвертерами типов*

Простые свойства и конвертеры типы

Конвертеры типов играет лишь одну роль – предоставляют служебные методы, которые могут преобразовывать определенные типы .NET в другие типы .NET, например, как в данном случае в строку. При поиске нужного конвертера типов анализатор выполняет следующие действия:

- Проверяет объявление свойства в поисках атрибута *TypeConverter*, если атрибут присутствует он указывает класс, выполняющий преобразование)
- Если же атрибут отстуствует, то анализатор проверяет объявление класса соотвествующего типа данных. Например, свойство Foreground используется объект Brush. Класс Brush и его наследники, используют BrushConverter
- Если в объявлении свойства или объявления класса не оказывается ассоциированного конвертера, то анализатор XAML генерирует ошибку.

Служебные свойства

Конвертеры типов не подходят для всех сценариев. Например, некоторые свойства являются полноценными объектами с собственными наборами свойств. Однако XAML предоставляет другой выбор. С помощью которого можно добавить дочерний элемент с именем в форме, используя следующий синтаксис <РодительскийЭлемент.ИмяСвойства>

```
<Grid Name="grid1">
  <Grid.Background>
    ...
  </Grid.Background>
  ...
</Grid>
```

Служебные свойства

Однако, проблеме не решена полностью, остается еще один вопрос: как установить сложное свойство после его идентификации?

Трюк заключаете в следующем. Внутрь вложенного объекта можно добавить другой дескриптор, чтобы создать экземпляр определенного класса

```
<Grid Name="grid1">
  <Grid.Background>
    <LinearGradientBrush>

    </LinearGradientBrush>
  </Grid.Background>
  ...
</Grid>
```

Служебные свойства

Однако, просто создать LinearGradientBrush недостаточно: нужно также указать цвета градиентов. Это делается свойством LinearGradienBrush.GradientStops, но опять-таки, оно слишком сложное, поэтому требуется положить еще одно свойство GradientStops

```
<Grid Name="grid1">
  <Grid.Background>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0.00" Color="Red" />
        <GradientStop Offset="0.50" Color="Indigo" />
        <GradientStop Offset="1.00" Color="Violet" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Grid.Background>
  ...
</Grid>
```

Служебные свойства

Любой набор дескрипторов XAML может быть заменен набором операторов кода, реализующих следующую задачу

```
LinearGradientBrush brush = new LinearGradientBrush();

GradientStop gradientStop1 = new GradientStop();
gradientStop1.Offset = 0;
gradientStop1.Color = Colors.Red;
brush.GradientStops.Add(gradientStop1);

GradientStop gradientStop2 = new GradientStop();
gradientStop2.Offset = 0.5;
gradientStop2.Color = Colors.Indigo;
brush.GradientStops.Add(gradientStop2);
GradientStop gradientStop3 = new GradientStop();
gradientStop3.Offset = 1;
gradientStop3.Color = Colors.Violet;
brush.GradientStops.Add(gradientStop3);

grid1.Background = brush;
```

Расширение разметки

В некоторых случаях не возможно жестко закодировать значения свойства. Например, значения должно быть установлено в уже существующий объект или может понадобиться установить значения свойства динамически , привязывая его к свойству другого элемента. В таких случаях необходимо использовать *расширение разметки* (markup extension) - специализированный синтаксис, устанавливающий

```
<Button ... Foreground="{x:Static SystemColors.ActiveCaptionBrush}" >
```

Все РР реализованы классами производными от System.Windows.Markup.MarkupExtension. Базовый класс чрезвычайно прост – включает единственный метод ProvideValue(), получающий необходимое значение