나를 넘는다.

WPF Async Programming 관련 MSDN 매거진 posted by msdnmag0414

https://msdn.microsoft.com/en-us/magazine/msdnmag0414

MSDN Async Programming 관련 포스트 모음 posted by Stephen cleary

https://msdn.microsoft.com/ko-kr/magazine/mt149362?author=stephen+cleary

Stephen Cleary 의 Asynchronous MVVM 어플리케이션 패턴 – Commands 편

https://msdn.microsoft.com/en-us/magazine/dn630647.aspx

WPF ICommandSource 구현의 Leak Memory 문제를 다룬 Article

https://blogs.msdn.microsoft.com/nathannesbit/2009/05/29/wpf-icommandsource-implementations-leak-memory/

**Introduction to XAML**

XAML(Extensible Application Markup Language) 은 XML 기반의 계층 관계를 통해 .NET 객체를 생성 및 초기화 하는 언어이다. WPF를 지원하기 위한 목적으로 작성되었지만 오늘 날에는 WPF 뿐만 아니라 Silverlight, WF 에서도 사용된다. WPF에 사용되는 모든 클래스들은 파라미터가 없는 생성자를 가지고 있으며 지나치게 많은 생성자를 사용하고 있는데, 이는 XAML 과 같은 XML 언어에 꼭 들어 맞기 위해서이다.

**XAML 의 이점**

XAML 에서 가능한 것은 모두 코드상에서 가능한 것이다. XAML 은 객체를 생성하고 초기화하기 위한 또 하나의 방법이다.

* XAML 코드는 짧고 분명하다.
* 디자인과 로직을 분리
* Graphical Design Tool 은 XAML 을 요구
* 디자이너와 개발자의 역할을 분리

**XAML 과 Code 비교**

단순한 TextBlock 을 포함하는 StackPanel 을 작성하는 두 가지 예시이다. 아래의 예시에서 보다시피 XAML 버전이 매우 간결하고 가독성이 뛰어나다.

우선 XAML

|  |
| --- |
| <StackPanel>  <TextBlock Margin="20">Welcome to the World of XAML</TextBlock>  <Button Margin="10" HorizontalAlignment="Right">OK</Button>  </StackPanel> |

다음은 C#

|  |
| --- |
| // Create the StackPanel  StackPanel stackPanel = [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) StackPanel();  **this**.Content = stackPanel;    // Create the TextBlock  TextBlock textBlock = [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) TextBlock();  textBlock.Margin = [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) Thickness(10);  textBlock.Text = "Welcome to the World of XAML";  stackPanel.Children.Add(textBlock);    // Create the Button  Button button = [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) Button();  button.Margin= [**new**](http://www.google.com/search?q=new+msdn.microsoft.com) Thickness(20);  button.Content = "OK";  stackPanel.Children.Add(button); |

**Element 로서 Properties**

Properties 는 일반적으로 XML 와 같이 inline 형태로 작성되었다. 하지만, 객체의 Content 에 이미지를 삽입한 복잡한 객체의 경우 property element syntax 를 사용할 수 있다. 이를 통해 property 를 마치 element 처럼 추출하여 표기가 가능하다.

|  |
| --- |
| <Button>  <Button.Content>  <Image Source="Images/OK.png" Width="50" Height="50" />  </Button.Content>  </Button> |

Implicit Type Conversion

매우 강력한 WPF 의 구조는 암시적인 타입 변환에 있다. Background 에서 동작하는 이러한 작업은 BorderBrush를 선언 후 Blue 를 문자열로 전달하였을 경우 BrushConverter 는 Blue 라는 문자열을 System.Windows.Media.Brushes.Blue 로 변경해준다. 이와 동일한 견지에서 BorderThickness 는 Tickness 객체로 변환된다. WPF 는 매우 많은 type converter 를 제공해주며 custom type Converter를 작성하는 것을 도와준다.

|  |
| --- |
| <Border BorderBrush="Blue" BorderThickness="0,10">  </Border> |

**Markup Extensions**

XAML 의 attribute 값에 대한 동적인 placeholder로서 runtime 상에서 property 를 변경해 준다.

|  |  |
| --- | --- |
| Binding | 두 개의 properties 의 값을 binding |
| StaticResource | Resource entry 를 한번 참조 |
| DynamicResource | 자동으로 Resource Entry 를 참조 |
| TemplateBinding | 1)Control 템플릿의 프로퍼티를 dependency property of the control 를 바인딩 |
| x:Static | 2)Static property 의 값을 resolve 해준다. |
| x:Null | Null 을 반환 |

1) To bind a property of a control template to a dependency property of the control

2) Resolve the value of a static property

Markup Extensions 는 curly braces 의 내에 문자열을 작성함으로써 XAML 의 구문분석기에 의해 해석된다고 하였다. Curly braces 내의 구문에 대해 자세히 언급하고자 한다. 우선, 첫번째 식별자는 extension 의 이름을 나타낸다(Binding, StaticResource, 등). 이 식별자 이후에 등장하는 식별자들은 모두 명명된 파라미터들이다. 즉, Property=Value 와 같은 형태로 기술되어 진다.

아래의 예시는 Label 의 컨텐츠가 TextBox 의 Text 와 바인딩 됨을 보여준다. 여러분이 Text Box 에 입력하는 값은 고스란히 Label 의 컨텐츠로 입력된다.

|  |
| --- |
| <TextBox x:Name="textBox"/>  <Label Content="{Binding Text, ElementName=textBox}"/> |

**Namespaces**

모든 XAML 문서의 서두에는 두 개의 namespace 를 반드시 입력해야 한다.

1. http://schemas.microsoft.com/winfx/2006/xaml/presentation

-> WPF Control 과 연결해주는 역할 (System.Windows.Controls)

2. http://schemas.microsoft.com/winfx/2006/xaml

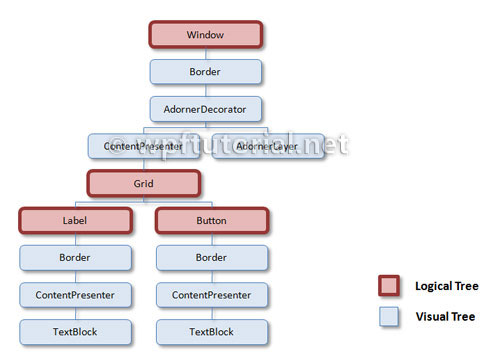
-> XAML 키워드를 정의하는 System.Windows.Makrup과 연결

XML 네임스페이스와 CLR 네임스페이스를 공유하는 것또한 XmlnsDefinition 을 통해 가능하다. 이는 assembly 레벨에서 이루어진다. 즉, 빌드가 되어야 공유 가능. 이러한 문제를 XAML 파일 작성 중에 namespace가 인식이 되지 않는 현상을 통해 이미 경험한 바 있다.

|  |
| --- |
| <Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">  </Window> |

**Logical And Visual Tree (https://www.wpftutorial.net/LogicalAndVisualTree.html)**

<https://www.wpftutorial.net/images/trees.jpg>



**Introduction**

WPF 의 Element 들은 계층적 구조를 가진다. 이러한 관계를 LogicalTree 라고 한다. 하나의 Element 템플릿을 구성하는 여러가지 시각적 element 들이 있다. 이를 VisualTree 라고 부른다. WPF 는 이러한 두가지 Tree를 구별한다.

|  |
| --- |
| <Window>  <Grid>  <Label Content="Label" />  <Button Content="Button" />  </Grid>  </Window> |

**Why do we need two different kind of trees?**

WPF control 은 다양한 서로 다른 원시 control 로 구성된다. 예를 들어 button 의 경우 border, rectangle, content presenter 로 구성되어 있다. 이러한 control 들은 button 의 시각적 element 이다. WPF 는 button 을 rendering 할 때 button 의 시각적 정보를 알지 못한다. Button 의 visualTree 를 iteration 하면서 Button Element 의 visual children element 를 렌더링해 나간다. 이러한 계층적 관계는 hit-testing, layout 을 하면서 사용될 수 있다.

**The Logical Tree**

Logical tree 는 user interface 의 element 관계를 표현한다. Logical Tree 는 다음의 항목을 책임진다.

- Inherit DependencyProperty values

- Resolving DynamicResources references

- Looking up element names for bindings

- Forwarding RoutedEvents

**The Visual Tree**

Visual Tree 는 다음의 항목을 책임진다.

- Rendering visual elements

- propagate element opacity

- propagate Layout- and RenderTransforms

- propagate the IsEnabled property

- Do Hit-Testing

- RelativeSource (FindAncestor)

**Programmatically Find an Ancestor in the Visual Tree**

아래의 코드는 Child Element에서 Parent Element를 찾는 코드이다.

|  |
| --- |
| **public** **static** **class** VisualTreeHelperExtensions  {  **public** **static** T FindAncestor<T>(DependencyObject dependencyObject)  where T : **class**  {  DependencyObject target = dependencyObject;  **do**  {  target = VisualTreeHelper.GetParent(target);  }  **while** (target != **null** && !(target [**is**](http://www.google.com/search?q=is+msdn.microsoft.com) T));  **return** target **as** T;  }  } |

아래의 코드는 위의 helper class 를 어떻게 사용하는 지를 보여준다.

|  |
| --- |
| var grid = VisualTreeHelperExtensions.FindAncestor<Grid>(**this**); |

**Dependency Properties**

https://www.wpftutorial.net/DependencyProperties.html#valueres

**Introduction**

WPF 개발 시작하기에 앞서 DenpendcyProperties 에서 난관을 겪을 것이다. 겉으로 보기에는 일반 .NET properties 와 일견 다를 것이 없어 보이지만, 이면의 개념은 복잡하면서도 강력하다.

1)주요한 차이점은 일반 properties 는 읽기 전용인 반면, DependencyProperties 는 동적으로 Resolution 된다는 점이다. DependencyObject 로부터 상소된 GetValue 메서드를 호출시에 이러한 동적 Resolve가 발생한다.

1) The main difference is, sthat the value of a normal .NET Property is read directly from a private member in your class, whereas the value of a DependencyProperty is resolved dynamically when calling the GetValue() method that is inherited from DependencyObject.

Dependency property 의 값을 설정할 때 이 값은 object 의 필드에 저장되지 않는다. Base class 인 DependencyObject 에 의해 제공된 Dictionary에 저장된다. Dictionary 의 Entry 에서 Key 는 Property 의 이름이며, Value 는 이 Property 의 실제 값이다.

이러한 dependency properties의 장점은 아래와 같다 .

- Reduced memory footprint

It’s huge dissipation to store a field for each property when you think that over 90% of the properties of a UI control typically stay at its initial values. Dependency properties solve these problems by only store modified properties in the instance. The default values are stored once within the dependency property.

- Value inheritance

When you access a dependency property the value is resolved by using a value resolution strategy. If no local value is set, the dependency property navigates up the logical tree until it finds a value. When you set the FontSize on the root element it applies to all textblocks below except you override the value.

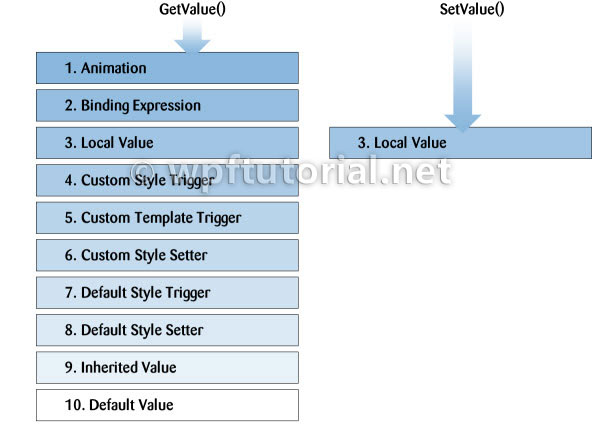
- Change notification

Dependency properties have a build-in change notification mechanism. By registering a callback in the property metadata you get notified, when the value of the property has been changed. This is also used by the databinding.

**Value resolution strategy**

Dependency object에 접근할 때 마다 내부적으로 다음의 우선순위에 기반하여 값을 Resolving 한다. 만약 아래의 순서로 값을 찾을 때 찾지못하는 경우 언제나 설정되어 있는 Default 값을 택하게 된다.

<https://www.wpftutorial.net/images/valueresolution.jpg>



**The magic behind it**