WPF Element Binding Kavramı

En basit tanımıyla bir WPF elementinin property'sini set etmek için başka bir WPF elementi ile bağlantı kurmasıdır. Fakat her property'ler bunu sağlayamaz. Bağlantının sağlanabilmesi için ilgili property dependency property olmak zorundadır.

Örnek: Slider Elementi İle **TextBlock** Elementlerini Birbirine Bağlamak

```
1   <Slider Name = "sliderFontSize" Margin="3"
2          Minimum="1" Maximum="40" Value="10"
3          TickFrequency="1" TickPlacement="TopLeft">
4   </Slider>
5   <TextBlock Margin="10" Text="Simple Text" Name="lblSampleText"
6              FontSize="{Binding ElementName=sliderFontSize, Path=Value}" >
7   </TextBlock>
```

Bazı Kavramlar

Source Object: ElementName property'sinin aldığı değer source elementtir. Örneğimizdeki Slider elementinin nesnesini temsil eder. Bu nesne **DataTable**, **DataRow** gibi klasik **.NET** nesnesi de olabilir. Biz bu makalede sadece WPF elementlerine odaklanacağız.

Target: **FontSize** property'dir. **Target** property dependency property olmak zorundadır.

Path: Property yerine Path kullanılarak esneklik sağlanmıştır. Property attribute'sine sadece herhangi bir property referans verilebilirken, Path attribute property'nin de property'sine referans verebilir.

Not: Grid.Row gibi attached property'e referans verebilmek için `Path=(Grid.Row)` şeklinde parantezlerle beraber kullanmak gereklidir.

Binding Errors

WPF'te var olmayan bir property veya bir element kullanırsak herhangi bir hata almayız. Bu sebepten dolayı debug işlemi yaparken hata meydana gelip gelmediğini WPF'in trace information sistemini kullanabiliriz. Bu bilgi Visual Studio'nun output penceresinde görülecektir. Bunun için **.NET 3.5** ile WPF elementlerinde `diag:PresentationTraceSources.TraceLeve` attribute kullanımı sağlanmıştır:

```
1   <StackPanel>
2       <TextBox Name="txtInput" />
3       <Label>
4           <Label.Content>
5               <Binding ElementName="txtInput"
6                       Path="Text"
7                       diag:PresentationTraceSources.TraceLevel="High" />
8           </Label.Content>
9       </Label>
10  </StackPanel>
```

Binding Modes

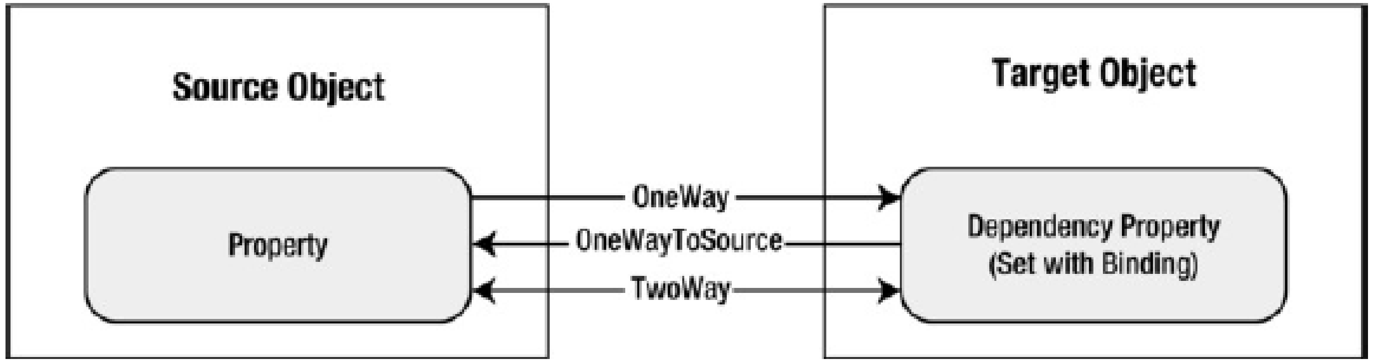
Kaynak elementte yapılan bir değişikliğin hedef elementi etkilemesini aynı şekilde hedef elementin kaynak elementi etkilemesini sağlamak için **Mode** property kullanılır. **Mode** property değeri TwoWay şeklinde belirtildiği zaman çift yönlü etkileşim sağlanmış olur.

```
1   <TextBlock Margin="10" Text="Simple Text" Name="lblSampleText"
2              FontSize="{Binding ElementName=sliderFontSize, Path=Value, Mode=TwoWay}" >
3   </TextBlock>
```

Mode property'nin alabileceği değerler şunlardır:

OneWay	Source property değiştiği zaman target property güncellenir
TwoWay	Source property değiştiği zaman target property güncellenir ve target property değiştiği zaman source property güncellenir
OneTime	Target property başlangıçta source property değerine eşit olur fakat source property'deki değişiklikler yansımaz
OneWayToSource	OneWay değerinin yaptığı işlemin tersi yapılır. Yani target property değiştiği zaman source property güncellenir
Default	Binding'in tipi target property bağlıdır. TwoWay veya OneWay olabilir. Tüm binding'ler bu yaklaşımı kullanırlar. Genelde Default değer tam istediğimiz işlemi yapar. Fakat bazı durumlarda spesifik değerlerden birini belirtmemiz gerekir. Örneğin read-only bir text box kullandığımızda OneWay değerini kullanırsak performans kazanmış olur.

Özetle aşağıdaki gibi bir yapı söz konusudur:



Not: OneWay ve **OneWayToSource** seçenekleri aslında aynı işlemi yapmaktadır. Fakat **OneWayToSource** ile WPF'in binding işlemlerinde sadece dependency property kullanılır sınırlandırmasını aşarız. Burada dikkat edilmesi gereken tek nokta değer sağlayan property dependency property olmalıdır.

Kodsal Olarak Bindings Yaratmak

Binding işlemlerini XAML aracılığı ile yapabileceğimiz gibi kodsal olarak ta yapabiliriz.

```
1   Binding binding = new Binding();
2   binding.Source= sliderFontSize;
3   binding.Path= new PropertyPath("Value");
4   binding.Mode= BindingMode.TwoWay;
5   lblSampleText.SetBinding(TextBlock.FontSizeProperty, binding);
```

Ayrıca XAML'de eklemiş olduğumuz bir binding'i kod tarafında silebiliriz. Bunun için **BindingOperations** sınıfının **ClearBinding()** metodunu kullanabiliriz. Eğer tüm binding'leri çıkarmak istiyorsak **ClearAllBindings()** metodu kullanılması gerekir. Bu iki metod tüm elementlerin kalıtım yoluyla **DependencyObject** sınıfından aldığı **ClearValue()** metodu ile temizleme işlemi yapar.

```
1   BindingOperations.ClearAllBindings(lblSampleText);
```

Kod Kısmından Binding Çağırmak

```
1   <TextBlock Margin="10" Text="Simple Text" Name="lblSampleText"
2              FontSize="{Binding ElementName=sliderFontSize, Path=Value}" >
3   </TextBlock>
```

Yukarıdaki örnekte yer alan binding ifadeyi kod kısmında çağırmak için aşağıdaki gibi kod yazmak gerekir:

```
1   Binding binding = BindingOperations.GetBinding(lblSampleText, TextBlock.FontSize);
```

GetBinding() metodu parametre olarak binding ifadenin kullanıldığı element adını, ikinci parametre ise property'i alır. **GetBinding()**

metodu yerine **GetBindingExpression()** metodunu kullanarak daha ayrıntılı bilgi elde edebiliriz:

```
1    BindingExpression expression = BindingOperations.GetBindingExpression(lblSampleText,
2    TextBlock.FontSize);
3
4    // Kaynak element elde edilir
5    Slider boundObject = (Slider)expression.ResolvedSource;
6
7    // Kaynak elementten FontSize property değerini al
8    string boundData = boundObject.FontSize;
```

Binding Updates

Kaynaktan hedef elementin property'sinin değişmesi anlık olurken, tersi işlem hemen gerçekleşmeyebilir. Tersinir yani hedef elementten kaynak elemente doğru güncelemeler **Binding.UpdateSourceTrigger** property tarafından kontrol edilir. Bu property aşağıdaki değerlerden birini alır:

PropertyChanged	Source target property değiştiği zaman anında güncellenir.
LostFocus	Target property değiştiğinde ve focusu kaybettiğinde source güncellenir.
Explicit	BindingExpression.UpdateSource() metodunu çağırmadıkça source güncellenmez
Default	Text property has a default behavior of LostFocus. Güncelleme işlemi target property'nin metadata'sı tarafından belirlenir. Bir çok property'nin varsayılan değeri PropertyChanged'tir. **TextBox** için default davranış şekli LostFocus'tur.

```
1    <TextBox Text="{Binding ElementName=txtSampleText, Path=FontSize, Mode=TwoWay,
2    UpdateSourceTrigger=PropertyChanged}" Name="txtFontSize"></TextBox>
```

Yukarıdaki gibi bir ayarlama yapıldığı zaman **TextBox**'a girilen bir değer anında slider değerini değiştirecektir.

Not: Kaynak elementinin güncellenme işlemini **UpdateSourceTriggger.Explicit** mod kullanarak yaparsak, dilediğimiz zaman güncelleme işlemini sağlamış oluruz. Örneğin, Apply butonu içerisinde **BindingExpression.UpdateSource()** metodunu çağırıp, yapılan değişikliklerin veritabanına kaydedilmesini sağlayabiliriz:

```
1    // Get the binding that's applied to the text box.
2    BindingExpression binding = txtFontSize.GetBindingExpression(TextBox.TextProperty);
3
4    // Update the linked source (the TextBlock).
5    binding.UpdateSource();
```

Binding Delays

Bazı durumlarda belli zaman geçtikten sonra kaynağın güncellenmesi gerekebilir. Böyle durumlarda Delay property'sini kullanmalıyız:

```
1    <TextBox Text="{Binding ElementName=txtSampleText, Path=FontSize, Mode=TwoWay,
2              UpdateSourceTrigger=PropertyChanged, Delay=500}"
3             Name="txtFontSize">
4    </TextBox>
```

WPF Elementi Olmayan Nesnelerin Kullanılması

Şimdiye kadar anlatılanlarda iki elementin birbirine bağlanmasından bahsettik. Bu kısımda WPF elementi olmayan bir nesnenin binding işlemlerinde kullanılmasından bahsedeceğiz. Element olmayan bir nesneyi kullanabilmek için bu nesnenin **public property'lere** sahip olması gerekir. **private field ve public field** binding işlemlerinde **kullanılmaz.**

Element olmayan bir nesnenin bağlanmasında aşağıdaki üç farklı property'lerden birini kullanabiliriz:

Source: Direkt olarak source nesnesine point etmek için kullanılır.

RelativeSource: Genelde template ve data template için kullanılır.

DataContext: Eğer **Source** veya **RelativeSource** property'leri kullanmazsak, WPF, o anki elementten başlayarak, element ağacı içerisinde elementleri tarar. **DataContext** property değeri **null** olmayan ilk elementin değeri alınır. Bu property, bir nesnenin birden çok

property'sini farklı elementlere bağlamayı sağladığı için oldukça kullanışlıdır.

Bu üç property'i detaylı incelleyelim..

Source

Kullanımı oldukça basittir.

```
1    <TextBlock Text="{Binding Source={x:Static SystemFonts.IconFontFamily},Path=Source}">
2    </TextBlock>
```

Bir başka örnek:

Resource olarak oluşturduğumuz bir nesne:

```
1    <Window.Resources>
2        <FontFamily x:Key="CustomFont">
3            Calibri
4        </FontFamily>
5    </Window.Resources>
```

Bu nesnenin kullanımı:

```
1    <TextBlock Text="{Binding Source={StaticResource CustomFont},Path=Source}">
2    </TextBlock>
```

RelativeSource

Bir elementi kendisine bağlamak için veya parent elemente bağlamak için kullanılır. Örneğin aşağıdaki uygulamada **Window**'un başlığı **TextBlock** elementinde gösterilmesi sağlanmıştır:

```
1    <TextBlock>
2        <TextBlock.Text>
3            <Binding Path="Title">
4                <Binding.RelativeSource>
5                    <RelativeSource Mode="FindAncestor" AncestorType="{x:Type Window}" />
6                </Binding.RelativeSource>
7            </Binding>
8        </TextBlock.Text>
9    </TextBlock>
```

FindAncestor modu ile element ağacı içinde **AncestorType** property'e bakılarak elementin tipinin aranması sağlanır.

Yukarıdaki örnek aşağıdaki gibi de yazılabilir:

```
1    <TextBlock Text="{Binding Path=Title,RelativeSource={RelativeSource FindAncestor,
2    AncestorType={x:Type Window}} }">
3    </TextBlock>
```

FindAncestor mod yerine aşağıdaki modlardan biri de kullanılabilir

Self	Elementin kendisini referans vererek, bir property'nin değerini başka bir property'sinde kullanabilmesi sağlanır.
PreviousData	Data-bound listede bir önceki data verisine bağlanmayı sağlar. Genelde list item gibi işlemlerde kullanılır.
TemplatedParent	Binding sadece control template veya data template içindeyken kullanılır. Bu mod ile template'in uygulandığı elemente bağlanırız.

DataContext

Bazı durumlarda, birden fazla element tek bir nesneye bağanması gerekebilir. Örneğin **User** isimli bir sınıftan yaratılan bir nesnenin property'lerinin **TextBox**'lar içinde gösterilmesi için, her bir **TextBox** elementinde ayrı ayrı belirtmek yerine **DataContext** property kullanılabilir. Bu property ile **TextBox** elementlerini bir **StackPanel** elementinin yaptığı gibi gruplayıp, bu element içinde **User** nesnesine

referans verip bu nesnenin property'lerini daha etkili kullanılmasını sağlayabiliriz.

Örnek olarak **SystemFonts.IconFontFamily** property'nin daha verimli kullanılmasını **DataContext** ile şu şekilde sağlarız:

```
1   <StackPanel>
2       <TextBlock Text="{Binding Source={x:Static SystemFonts.IconFontFamily},Path=Source}">
3       </TextBlock>
4       <TextBlock Text="{Binding Source={x:Static SystemFonts.IconFontFamily},Path=LineSpacing}">
5       </TextBlock>
6       <TextBlock Text="{Binding Source={x:Static
7   SystemFonts.IconFontFamily},Path=FamilyTypefaces[0].Style}">
8       </TextBlock>
9       <TextBlock Text="{Binding Source={x:Static
10  SystemFonts.IconFontFamily},Path=FamilyTypefaces[0].Weight}">
11      </TextBlock>
12  </StackPanel>
```

Yukarıdaki örneği şimdi **DataContext** ile yapalım:

```
1   <StackPanel DataContext="{x:Static SystemFonts.IconFontFamily}">
2       <TextBlock Text="{Binding Path=Source}">
3       </TextBlock>
4       <TextBlock Text="{Binding Path=LineSpacing}">
5       </TextBlock>
6       <TextBlock Text="{Binding Path=FamilyTypefaces[0].Style}">
7       </TextBlock>
8       <TextBlock Text="{Binding Path=FamilyTypefaces[0].Weight}">
9       </TextBlock>
10  </StackPanel>
```

Örnekte görüldüğü gibi **TextBlock** elementlerinin **Source** property'leri yazılmamıştır. Bunun yerine parent elementi içinde yazılmıştır. WPF element ağacı içinde **TextBlock** elementinden başlayarak **StackPanel**'e gider. Eğer **StackPanel** içinde **DataContext** property set edilmeseydi, **Window** elementine gelinceye kadar element ağacı taranırdı. **StackPanel** elementinde set edildiği için tarama işlemi devam etmez.

Özet

Bu makalede element binding işlemlerinin nasıl yapıldığından ve kullanım amacından bahsettik. Element binding, WPF uygulamaları geliştirirken oldukça kullanışlı bir yöntemdir.