

Module 9

"Data Binding [Deeper Dive]"



TEKNOLOGISK
INSTITUT

Agenda

- ▶ **Data Binding Properties in Details**
- ▶ Data Conversion
- ▶ Validation
- ▶ Data Binding Collections in Details

The **Binding** Class

▶ Binding

- ElementName, Source, RelativeSource
- Path
- XPath XML sources
- Mode
- Delay
- NotifyOnSourceUpdated SourceUpdated event
- NotifyOnTargetUpdated TargetUpdated event
- FallbackValue + TargetNullValue

▶ Binding can be set

- Declaratively in XAML
 - Full syntax or markup extension syntax "{}"
- Programmatically in code

▶ Bindings can be cleared (in code) via **BindingOperations.ClearBinding()**

Binding Mode

- ▶ The binding mode specifies the direction of the binding
- ▶ **Binding**
 - **Mode**
 - **Default** determined by the target property
 - **OneTime**
 - **OneWay**
 - **OneWayToSource**
 - **TwoWay**
 - **UpdateSourceTrigger**
 - **Default** determined by the target property
 - **Explicit** when **BindingExpression.UpdateSource()** is called
 - **LostFocus**
 - **PropertyChanged**

Small, Important Features... ☺

- ▶ Bindings can be delayed
 - New feature in .NET 4.5

```
<Slider Name="slider"  
        Value="{Binding ElementName=textbox,  
                        Path=Text,  
                        Mode=TwoWay,  
                        Delay=500}" />
```

- ▶ Applies for updates from target to source in two-way bindings
 - The delay does not apply for both directions!
- ▶ Data binding can be set up "visually" through Visual Studio

Agenda

- ▶ Data Binding Properties in Details
- ▶ **Data Conversion**
- ▶ Validation
- ▶ Data Binding Collections in Details

IValueConverter

- ▶ Value converters
 - Convert, format, localize, combine data
 - Conditionally or unconditionally
 - Usually applied with Data Binding
- ▶ **IValueConverter**
 - **Convert()** Source -> Target
 - **ConvertBack()** Target -> Source
- ▶ Localization proceeds through the **culture** parameter
 - Defaults to **xml:lang**
- ▶ **ValueConversionAttribute**
 - Only conveys the intended meaning to tools
 - Has no semantic meaning!

IMultiValueConverter

- ▶ Multi-value converters
 - Performs similar conversion, but combines multiple values
- ▶ **IMultiValueConverter**
 - **Convert()**
 - Accepts **object[]** instead of just **object**
 - **ConvertBack()**
 - Returns **object[]** instead of just **object**
- ▶ Must use **MultiBinding** to apply multi-value converter!
- ▶ Side note:
 - There is also a **PriorityBinding** for use with asynchronous data binding. We will investigate that in Module 13

Data Binding Special Cases

- ▶ Converters can return special values when conversions are not possible (or not wanted)

- ▶ **Binding.**
 - `FallbackValue`
 - `DoNothing` Ignore update

- ▶ **DependencyProperty.**
 - `UnsetValue` "Errorneous" update

Agenda

- ▶ Data Binding Properties in Details
- ▶ Data Conversion
- ▶ **Validation**
- ▶ Data Binding Collections in Details

Validation Rules

- ▶ Data bindings are subjected to a set of validation rules
 - Exceptions are by default not thrown
- ▶ **ValidationRule**
 - **ExceptionValidationRule** built-in
- ▶ Setting **Binding.ValidatorsOnExceptions** to true
 - is equivalent to adding **ExceptionValidationRule**
- ▶ Custom validation rules
 - Override **ValidationRule.Validate()**
 - **ValidationResult** signals success or failure

Handling Validation Errors

▶ Validation.

- HasErrors
- Errors ValidationError collection
- Error event
 - Raised if `Binding.NotifyOnValidationError == true`

▶ ValidationErrorEventArgs

- Action
 - Added
 - Removed
- Error
 - ErrorContent
 - Exception
 - RuleInError
 - BindingInError

Validation Error Templates

- ▶ Controls can set attached **Validation.ErrorTemplate** property
- ▶ Note:
 - This is a **ControlTemplate** (not a **DataTemplate**)
 - **AdornedElementPlaceholder** refers back to control being validated!

IDataErrorInfo

- ▶ **IDataErrorInfo** in **System.ComponentModel**

```
public interface IDataErrorInfo
{
    string Error { get; }
    string this[ string columnName ] { get; }
}
```

- ▶ **DataErrorValidationRule**
 - Checks for errors raised by classes implementing **IDataErrorInfo**
- ▶ Setting **Binding.ValidatesOnDataErrors** to true
 - is equivalent to adding **DataErrorValidationRule**
- ▶ WPF 4.5 adds **INotifyDataErrorInfo** for more advanced uses
 - There is a similar **Binding.ValidatesOnNotifyDataErrors**

Agenda

- ▶ Data Binding Properties in Details
- ▶ Data Conversion
- ▶ Validation
- ▶ **Data Binding Collections in Details**

Design-time Data

- ▶ XAML allows setting specific properties applying only to design-time

```
<Window ...  
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
  xmlns:mc="http://schemas.openxmlformats.org/markup-  
compatibility/2006"  
  mc:Ignorable="d">  
    <d:Window.DataContext>  
      <clr:Participants>  
        <clr:Participant FirstName="Jesper"  
                          LastName="Gulmann Henriksen"  
                          Company="Wincubate ApS" />  
      </clr:Participants>  
    </d:Window.DataContext>  
    ...  
</Window>
```


Hierarchical Data Templates

► **HierarchicalDataTemplate**

- Excellent for hierarchical data
 - e.g. file system etc.

► Use

- **HierarchicalDataTemplate** for internal nodes
- "regular" DataTemplate for leaf nodes

► Note:

- A "non-recursive" example is supplied in Lab 3.3

Sorting

- ▶ Bound data can be sorted through **ICollectionView**
- ▶ **ICollectionView**
 - **SortDescriptions**
 - **SortDescription** collection
- ▶ Specifically for **ListCollectionView**
 - **CustomSort**
 - **IComparer** implementation
- ▶ Note
 - The **SortDescription** class is defined in the **System.ComponentModel** namespace in **WindowsBase.dll**.

Grouping

- ▶ Bound data can be grouped through **ICollectionView**
- ▶ **ICollectionView**
 - **GroupDescriptions**
 - **PropertyGroupDescription** collection
- ▶ **ItemsControl**
 - **GroupStyle**
 - **HeaderTemplate**, **HeaderTemplateSelector**
 - **ContainerStyle**, **ContainerStyleSelector**
 - **Panel**
- ▶ Custom Grouping
 - Can be performed by implementing **IValueConverter** appropriately

Filtering

- ▶ Bound data can be filtered through **ICollectionView**
- ▶ **ICollectionView**
 - **Filter**
 - should be set to filtering predicate (of **object!**) in code
- ▶ This approach does not work for ADO.NET objects
 - These views are **BindingListCollectionView**
 - **CustomFilter** = *"ColumnName Operator Value"*

CollectionViewSource

- ▶ Collection views can similarly be created in XAML
 - Define a **CollectionViewSource** instance bound to data
 - Bind ItemsControl to the **CollectionViewSource** instance

```
<Window.Resources>  
    <clr:Participants x:Key="participants" />  
    <CollectionViewSource x:Key="cvs"  
        Source="{Binding Source={StaticResource participants}}" />  
</Window.Resources>
```

```
<ListBox ItemsSource="{Binding Source={StaticResource cvs}}"  
    DisplayMemberPath="FullName"/>
```

- ▶ Sorting can also be applied in XAML

Summary

- ▶ Data Binding Properties in Details
- ▶ Data Conversion
- ▶ Validation
- ▶ Data Binding Collections in Details



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Ringgårdsvej 4A

8270 Højbjerg

Denmark