“Databinding in WPF” Gill Cleeren

May 22, 2009
[MS Techdays 2009] Summary “Databinding in WPF” Gill Cleeren

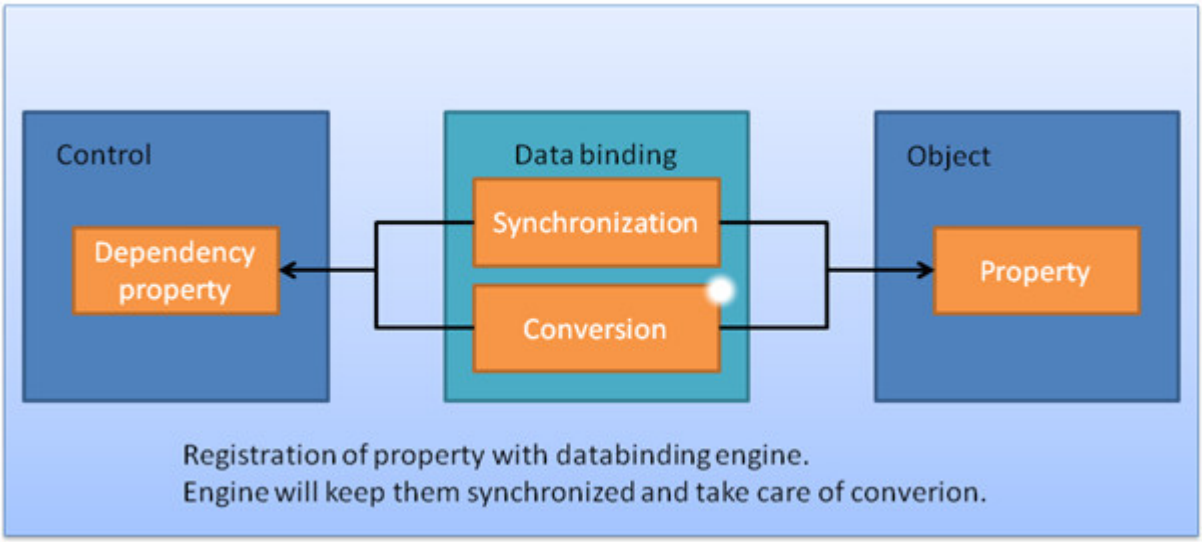
This was one of the presentations I didn’tt find very interesting so I haven’t a lot of notes. For details you should have a look at the conference slides.

Why Databinding

Databinding is not unique to WPF it appeared already in ASP.Net and Winforms applications. Not using databinding means writing a lot of code: code to get controls in an initial state, code to handle state update events (data object’s state) and code to update the visual controls (representation of the object on the GUI). This plumbing code is taken care of by databinding. A common mistake is thinking that databinding requires a DB. Other sources are supported as well.

What is Databinding

The basic concepts can be summarized as follows: properties of a control are bound to properties of a data object. Databinding, in other words, glues properties together.



Databinding requires 4 things:

- a source object
- a target object
- a source Property
- a target Property

The SetBinding method links them together:

```
public MainWindow()
{
    InitializeComponent();
    Binding binding = new Binding();
    binding.Source = treeView;
    binding.Path = new PropertyPath("SelectedItem.Header");
    currentFolder.SetBinding(TextBlock.TextProperty, binding);
}
```

```
BindingOperations.SetBinding
        (currentFolder, TextBlock.TextProperty, binding);
```

Binding can also be done by means XAML through Dependency Properties:

```
<TextBlock x:Name="currentFolder" DockPanel.Dock="Top"
Text="{Binding ElementName=treeView, Path=SelectedItem.Header} "
Background="AliceBlue" FontSize="16" />
```

The *ElementName* is the source of the binding. For XAML you?e required to add code responsible to handle change events by implementing *INotifyPropertyChanged* or *INotifyCollenctionChanged*. *DataTemplates* allow you to apply different visual templates to a control based on the data's content.

Types of Binding

- **OneWay**: Target updates when source changes
- **TwoWay**: Change to either changes the other
- **OneTime**: Similar to OneWay, changes aren't reflected in Target (snapshot view)
- **OneWayToSource**: Source updates when Target changes

The update events fire when the control loses focus, when a property in the source or target has changed or whenever an explicit call to the *UpdateSource* method is made.

Value Conversions, Grouping and Sorting

Value Conversion, before-after:

Value conversions allow you to morph source values to different target values. This is done through the use of the ValueConversion attribute and the *Convert* and *ConvertBack* methods.

Bound collections can be grouped and sorted, see examples below, through views on the collection by implementing the *ICollectionView*:

```
view.SortDescriptions.Add(new SortDescription("DateTime",
ListSortDirection.Descending));
view.SortDescriptions.Add(new SortDescription("Name",
ListSortDirection.Ascending));
```

```
ICollectionView view = CollectionViewSource.GetDefaultView(
this.FindResource("photos"));
view.GroupDescriptions.Clear();
view.GroupDescriptions.Add(new PropertyGroupDescription("DateTime"));
```

Validation

Two types of validation are supported:

- Exception validation rules: used when your data objects already have validation in them
- Custom validation rules: if validation is added for the first time.

You can modify the presentation of a control when the validation has failed through setting the Validation.ErrorTemplate.

Data Providers

Two generic data providers are available:

- XmlDataProvider: to databind XML, the data is gathered through XPath expressions
- ObjectDataProvider: objects can be anything for example XAML declarations

Summary

The goal of databinding is to reduce the tedious coding needed to glue the interface controls to data objects

holding the state. Syntax can be daunting to begin with but allows for a high degree of customization.
