Binding Declarations Overview

.NET Framework 4

Updated: September 2010

This topic discusses the different ways you can declare a binding.

This topic contains the following sections.

Prerequisites

Before reading this topic, it is important that you are familiar with the concept and usage of markup extensions. For more information about markup extensions, see Markup Extensions and WPF XAML.

This topic does not cover data binding concepts. For a discussion of data binding concepts, see Data Binding Overview.

Declaring a Binding in XAML

This section discusses how to declare a binding in XAML.

Markup Extension Usage

Binding is a markup extension. When you use the binding extension to declare a binding, the declaration consists of a series of clauses following the Binding keyword and separated by commas (,). The clauses in the binding declaration can be in any order and there are many possible combinations. The clauses are *Name*=*Value* pairs where *Name* is the name of the Binding property and *Value* is the value you are setting for the property.

When creating binding declaration strings in markup, they must be attached to the specific dependency property of a target object. The following example shows how to bind the TextBox.Text property using the binding extension, specifying the Source and Path properties.

XAML

```
<TextBlock Text="{Binding Source={StaticResource myDataSource}, Path=PersonName}"/>
```

You can specify most of the properties of the Binding class this way. For more information about the binding extension as well as for a list of Binding properties that cannot be set using the binding extension, see the Binding Markup Extension overview.

Object Element Syntax

Object element syntax is an alternative to creating the binding declaration. In most cases, there

is no particular advantage to using either the markup extension or the object element syntax. However, in cases which the markup extension does not support your scenario, such as when your property value is of a non-string type for which no type conversion exists, you need to use the object element syntax.

The following is an example of both the object element syntax and the markup extension usage:

XAML

```
<TextBlock Name="myconvertedtext"
  Foreground="{Binding Path=TheDate,
                       Converter={StaticResource MyConverterReference}}">
  <TextBlock.Text>
    <Binding Path="TheDate"
             Converter="{StaticResource MyConverterReference}"/>
  </TextBlock.Text>
</TextBlock>
```

The example binds the Foreground property by declaring a binding using the extension syntax. The binding declaration for the Text property uses the object element syntax.

For more information about the different terms, see XAML Syntax In Detail.

MultiBinding and PriorityBinding

MultiBinding and PriorityBinding do not support the XAML extension syntax. Therefore, you must use the object element syntax if you are declaring a MultiBinding or a PriorityBinding in XAML.

Creating a Binding in Code

Another way to specify a binding is to set properties directly on a Binding object in code. The following example shows how to create a Binding object and specify the properties in code. In this example, TheConverter is an object that implements the IValueConverter interface.

VB

```
        Private Sub OnPageLoaded(ByVal sender As Object, ByVal e As EventArgs)
                ' Make a new source, to grab a new timestamp
                Dim myChangedData As New MyData()

                ' Create a new binding
        ' TheDate is a property of type DateTime on MyData class
                Dim myNewBindDef As New Binding("TheDate")

                myNewBindDef.Mode = BindingMode.OneWay
                myNewBindDef.Source = myChangedData
                myNewBindDef.Converter = TheConverter
                myNewBindDef.ConverterCulture = New CultureInfo("en-US")

        ' myDatetext is a TextBlock object that is the binding target object
                BindingOperations.SetBinding(myDateText, TextBlock.TextProperty, myNew
                BindingOperations.SetBinding(myDateText, TextBlock.ForegroundProperty,



...
```

```
        End Sub
```

If the object you are binding is a FrameworkElement or a FrameworkContentElement you can call the SetBinding method on your object directly instead of using BindingOperations.SetBinding. For an example, see How to: Create a Binding in Code.

Binding Path Syntax

Use the Path property to specify the source value you want to bind to:

- In the simplest case, the Path property value is the name of the property of the source object to use for the binding, such as Path=PropertyName.
- Subproperties of a property can be specified by a similar syntax as in C#. For instance, the clause Path=ShoppingCart.Order sets the binding to the subproperty Order of the object or property ShoppingCart.
- To bind to an attached property, place parentheses around the attached property. For example, to bind to the attached property DockPanel.Dock, the syntax is Path=(DockPanel.Dock).
- Indexers of a property can be specified within square brackets following the property name where the indexer is applied. For instance, the clause Path=ShoppingCart[0] sets the binding to the index that corresponds to how your property's internal indexing handles the literal string "0". Nested indexers are also supported.
- Indexers and subproperties can be mixed in a Path clause; for example, Path=ShoppingCart.ShippingInfo[MailingAddress,Street].
- Inside indexers you can have multiple indexer parameters separated by commas (,). The type of each parameter can be specified with parentheses. For example, you can have Path="[(sys:Int32)42,(sys:Int32)24]", where sys is mapped to the System namespace.
- When the source is a collection view, the current item can be specified with a slash (/). For example, the clause Path=/ sets the binding to the current item in the view. When the source is a collection, this syntax specifies the current item of the default collection view.
- Property names and slashes can be combined to traverse properties that are collections. For example, Path=/Offices/ManagerName specifies the current item of the source collection, which contains an Offices property that is also a collection. Its current item is an object that contains a ManagerName property.
- Optionally, a period (.) path can be used to bind to the current source. For example, Text="{Binding}" is equivalent to Text="{Binding Path=.}".

Escaping Mechanism

- Inside indexers ([ ]), the caret character (^) escapes the next character.
- If you set Path in XAML, you also need to escape (using XML entities) certain characters that are special to the XML language definition:
  - Use & to escape the character "&".
  - Use > to escape the end tag ">".

- Additionally, if you describe the entire binding in an attribute using the markup extension syntax, you need to escape (using backslash \) characters that are special to the WPF markup extension parser:
  - Backslash (\) is the escape character itself.
  - The equal sign (=) separates property name from property value.
  - Comma (,) separates properties.
  - The right curly brace (}) is the end of a markup extension.

Default Behaviors

The default behavior is as follows if not specified in the declaration.

- A default converter is created that tries to do a type conversion between the binding source value and the binding target value. If a conversion cannot be made, the default converter returns null.
- If you do not set ConverterCulture, the binding engine uses the Language property of the binding target object. In XAML, this defaults to "en-US" or inherits the value from the root element (or any element) of the page, if one has been explicitly set.
- As long as the binding already has a data context (for instance, the inherited data context coming from a parent element), and whatever item or collection being returned by that context is appropriate for binding without requiring further path modification, a binding declaration can have no clauses at all: {Binding} This is often the way a binding is specified for data styling, where the binding acts upon a collection. For more information, see the "Entire Objects Used as a Binding Source" section in the Binding Sources Overview.
- The default Mode varies between one-way and two-way depending on the dependency property that is being bound. You can always declare the binding mode explicitly to ensure that your binding has the desired behavior. In general, user-editable control properties, such as TextBox.Text and RangeBase.Value, default to two-way bindings, whereas most other properties default to one-way bindings.
- The default UpdateSourceTrigger value varies between PropertyChanged and LostFocus depending on the bound dependency property as well. The default value for most dependency properties is PropertyChanged, while the TextBox.Text property has a default value of LostFocus.

See Also

Reference

PropertyPath XAML Syntax

Concepts

Data Binding Overview
Optimizing Performance: Data Binding

Other Resources

Data Binding How-to Topics

Change History

Date	History	Reason
September 2010	Explained TheConverter in "Creating a Binding in Code."	Customer feedback.