#15 TECHARTIKEL

WPF DATA BINDING

10.02.2025

Nadine Gießler

Inhalt

WPF Data Binding

Die Datenbindung stellt eine Verbindung zwischen der Benutzeroberfläche der Anwendung und den dort angezeigten Daten her. Das bedeutet, dass die mit den Daten verknüpften Elemente bzw. dass bei Änderungen an der Benutzeroberfläche die verknüpften Daten, automatisch aktualisiert werden.

Daten können in Form von .NET-Objekten und XML-Daten gebunden werden.

Unabhängig davon, welches Element gebunden werden soll und welcher Art die Datenquelle ist, erfolgt die Bindung immer nach dem in der folgenden Abbildung dargestellten Modell.

Abbildung 1

Wie in der Abbildung dargestellt, ist die Datenbindung die Brücke zwischen dem Bindungsziel und der Bindungsquelle.

Eine Datenbindung besteht normalerweise aus vier Komponenten: Zielobjekt, Zieleigenschaft, Bindungsquelle und Eigenschaft der Bindungsquelle.

Wenn man z.B. den Inhalt einer TextBox an die Eigenschaft **„Customer.Name“** bindet, sieht das wie folgt aus:

- Zielobjekt: **TextBox**
- Zieleigenschaft: **Text**
- Bindungsquelle: **Customer**
- Eigenschaft der Bindungsquelle: **Name**

Die verwendeten Eigenschaften des Quellobjekts müssen öffentliche Eigenschaften der Klasse sein. Explizit definierte Interface-Eigenschaften sowie geschützte, private, interne und virtuelle Eigenschaften können nicht für eine Bindung verwendet werden.

Bindungen werden in der Regel im XAML definiert, aber auch eine Bindung im Code ist möglich.

1. Datenkontext

Der Datenkontext ist das Quellobjekt für die Auswertung des Pfades zum Wert des Quellobjekts. Man kann ihn selbst definieren oder er wird vom übergeordneten Objekt geerbt.

Verknüpfungen können zu einem bestimmten Objekt aufgelöst werden. So kann z.B. die Vordergrundfarbe eines Objekts an die Hintergrundfarbe eines anderen Objekts gebunden werden. Dazu ist kein DataContext erforderlich.

Wenn sich die DataContext-Eigenschaft ändert, werden alle Bindungen, die vom Datenkontext betroffen sein könnten, neu ausgewertet.

```
<Window x:Class="Test.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Test"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
        <local:Customer/>
    </Window.DataContext>
    <StackPanel>
        <TextBlock Text="Name:"/>
        <TextBlock Text="{Binding Name}"/>
    </StackPanel>
</Window>
```

Abbildung 2

Wie im Beispiel zu sehen, kann die Eigenschaft „Name“ nun direkt als Binding gesetzt werden, wenn die Klasse „Customer“ als Datenkontext gesetzt wird.

2. Binding Modus

Der Datenfluss von der Quelle zum Ziel wird durch den Binding.Mode definiert. Die verschiedenen Datenflussarten sind hier dargestellt:

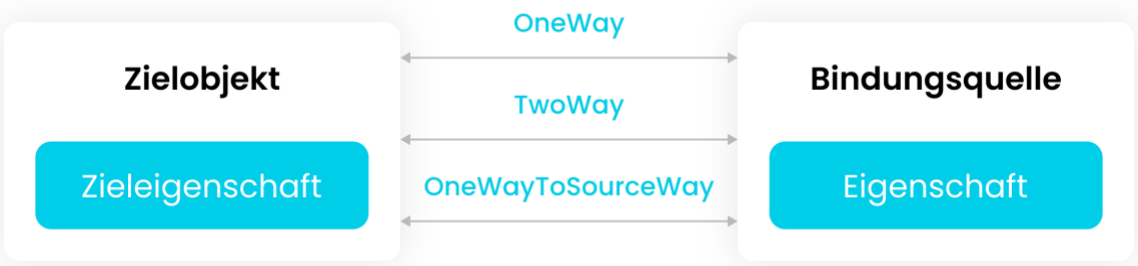

Abbildung 3

OneWay-Binding:

Bei Änderungen an der Quelleigenschaft wird die Zieleigenschaft automatisch aktualisiert, ohne dass Änderungen an der Zieleigenschaft zurück an die Quelleigenschaft übertragen werden. Dies ist vor allem bei schreibgeschützten Elementen sinnvoll.

TwoWay-Bindung:

Bei Änderungen an der Quell- oder Zieleigenschaft wird, die jeweils andere, automatisch aktualisiert. Dies eignet sich besonders für interaktive Elemente wie z.B. Formulare.

Die meisten Eigenschaften sind standardmäßig auf OneWay-Binding eingestellt, einige Abhängigkeitseigenschaften (z.B. TextBox.Text und CheckBox.IsChecked) sind standardmäßig auf TwoWay-Binding eingestellt.

OneWayToSource-Bindung:

Hier wird die Quelleigenschaft aktualisiert, wenn die Zieleigenschaft geändert wird. Ein Beispielszenario hierfür ist, wenn nur der Quellwert von der Benutzerschnittstelle neu bewertet werden muss.

Um Änderungen an der Quelle zu erkennen – sowohl bei der OneWay- als auch bei der TwoWay-Bindung –, muss die Quelle einen geeigneten Mechanismus zur Benachrichtigung über Eigenschaftsänderungen implementieren, beispielsweise durch die Verwendung von INotifyPropertyChanged.

Hier ein Beispiel für ein solche Implementierung:

```
public class Customer : INotifyPropertyChanged
{
    0 references
    public string Name
    {
        get => _name;
        set
        {
            if (_name == value) return;

            _name = value;
            OnPropertyChanged();
        }
    }

    private string _name;

    public event PropertyChangedEventHandler? PropertyChanged;

    1 reference
    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke( sender: this,  e: new PropertyChangedEventArgs(propertyName));
    }
}
```

Abbildung 4

Beim Setzen des Namens wird ein OnPropertyChanged Event aufgerufen, über das die Aktualisierung in der UI erfolgt.

3. UpdateSourceTrigger

Das Binding.UpdateSourceTrigger bestimmt, wodurch die Aktualisierung der Quelle ausgelöst wird. Die folgende Abbildung veranschaulicht die Rolle der Binding.UpdateSourceTrigger-Eigenschaft:

Abbildung 5

PropertyChanged:

Hier wird die Quelleigenschaft aktualisiert, sobald sich die Zieleigenschaft ändert.

LostFocus:

Mit LostFocus wird die Quelleigenschaft nur dann mit dem neuen Wert aktualisiert, wenn die Zieleigenschaft den Fokus verliert.

Der Standardwert für die meisten Abhängigkeitseigenschaften ist PropertyChanged. Bei Textfeldern kann eine Aktualisierung nach jeder Tastatureingabe jedoch die Performance beeinträchtigen und führt außerdem dazu, dass der Benutzer Tippfehler nicht wie gewohnt durch Drücken der Rücktaste korrigieren kann, bevor der neue Wert übergeben wird. Die TextBox.Text-Eigenschaft verwendet beispielsweise standardmäßig LostFocus, um diese Probleme zu vermeiden.

Im folgenden Beispiel wird die Eigenschaft „Age“ bei der Tastatureingabe aktualisiert und die Anzeige der Textbox kann über die TwoWay-Bindung auch aus dem Code heraus aktualisiert werden.

```
12    <StackPanel>
13        <TextBlock Text="Age:"/>
14        <TextBox Text="{Binding Age, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" Width="120" HorizontalAlignment="Left"/>
15    </StackPanel>
```

Abbildung 6

4. Konvertierung

Es wird ein Standardkonverter erzeugt, der versucht, eine Typkonvertierung zwischen dem Wert der Bindungsquelle und dem Wert des Bindungsziels durchzuführen. Wenn keine Konvertierung durchgeführt werden kann, gibt der Standardkonverter Null zurück. Deswegen kann es sinnvoll sein, einen eigenen Konverter zu erstellen.

5. Binden von Datenauflistungen

Neben der Bindung an einzelne Objekte können auch Datenauflistungen (z.B. Ergebnisse einer Datenbankabfrage etc.) gebunden werden. Für die Anzeige solcher Datenauflistungen empfiehlt es sich, ein ItemsControl wie eine ListBox, ListView oder TreeView zu verwenden.

Jede Auflistung, die das IEnumerable Interface implementiert, kann angezeigt werden. Um dynamische Bindungen zu erstellen, bei denen die Benutzerschnittstelle automatisch aktualisiert wird, wenn Elemente in die Auflistung eingefügt oder aus ihr gelöscht werden, muss die Auflistung die INotifyCollectionChanged Schnittstelle implementieren.

WPF stellt die ObservableCollection-Klasse zur Verfügung, die eine integrierte Implementierung einer Datenauflistung ist, die die INotifyCollectionChanged-Schnittstelle zur Verfügung stellt. Um eine vollständige Unterstützung für die Übertragung von Datenwerten von Quell- zu Zielobjekten zu gewährleisten, muss jedes Objekt in der Auflistung, welches diese zu bindende Eigenschaften unterstützt, auch die INotifyPropertyChanged-Schnittstelle implementieren.

6. Validierung

Die meisten Anwendungen, die Benutzereingaben erfordern, benötigen Validierungslogik, um sicherzustellen, dass der Benutzer die erwarteten Informationen eingegeben hat.

Das WPF-Datenbindungsmodell ermöglicht die Zuordnung von ValidationRules zum Binding-Objekt.

Eine Validierung findet normalerweise statt, wenn der Wert eines Ziels an die der Eigenschaft der Bindungsquelle übergeben wird. Diese Übergabe erfolgt bei TwoWay- und OneWayToSource-Bindungen. Der Auslöser für die Aktualisierung einer Quelle hängt also vom Wert der UpdateSourceTrigger-Eigenschaft ab.

Ein ValidationRule-Objekt prüft, ob der Wert einer Eigenschaft gültig ist. WPF bietet zwei Arten von integrierten ValidationRule-Objekten:

ExceptionValidationRule:

Dieses ValidationRule-Objekt überprüft, ob während der Aktualisierung der Eigenschaft der Bindungsquelle Ausnahmen ausgelöst wurden. Im Falle einer ungültigen Eingabe wird eine Exception ausgelöst, wodurch die Bindung als ungültig markiert wird.

DataErrorValidationRule:

Hier wird geprüft, ob Fehler vorliegen, die von Objekten ausgelöst wurden, die das IDataErrorInfo Interface implementieren.

Es können auch eigene Validierungsregeln erstellt werden, indem man die ValidationRule-Klasse ableitet und die Validate-Methode implementiert, wie das folgende Beispiel zeigt:

```
14        <TextBox>
15            <TextBox.Text>
16                <Binding Path="Age">
17                    <Binding.ValidationRules>
18                        <local:AgeValidationRule/>
19                    </Binding.ValidationRules>
20                </Binding>
21            </TextBox.Text>
22        </TextBox>
```

Abbildung 7

Im XAML wird eine entsprechend selbst erstellte ValidationRule gebunden. Diese erbt von der ValidationRule und überschreibt das ValidationResult. Dort steht die entsprechende Prüfung und Fehlermeldung.

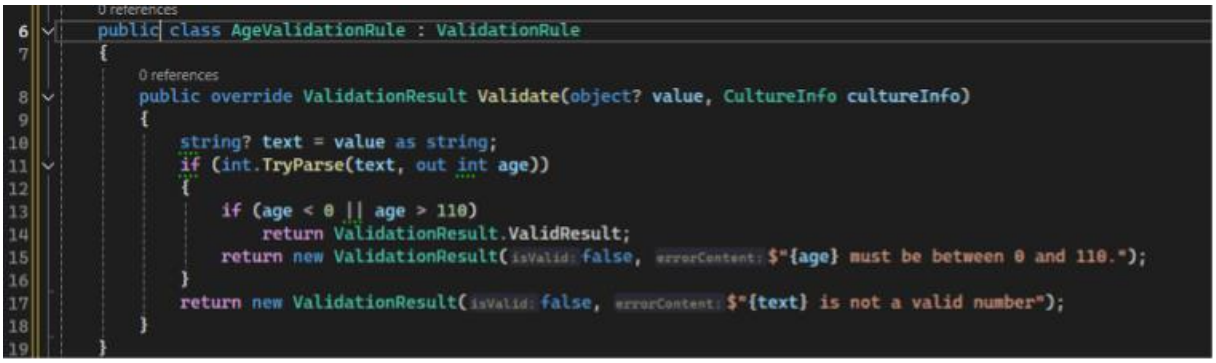

Abbildung 8

Wenn der Benutzer einen ungültigen Wert eingibt, kann es nützlich sein, eine Rückmeldung über den Fehler auf der Benutzeroberfläche der Anwendung bereitzustellen. Eine Möglichkeit der Rückmeldung besteht darin, die angehängte Eigenschaft Validation.ErrorTemplate auf ein benutzerdefiniertes ControlTemplate zu setzen.

7. Fazit

Datenbindung ist eine der Grundlagen von WPF und der empfohlene Weg, um Daten dem Benutzer zu präsentieren. Es ist eine wirkungsvolle Technik bei der Entwicklung von Benutzerschnittstellen, da sie hilft, die Darstellungslogik von der Geschäftslogik zu trennen und den resultierenden Code unabhängig davon zu testen.

Durch die verschiedenen Arten von **Binding Mode** und **UpdateSourceTrigger** können unterschiedliche Anwendungsfälle abgedeckt werden. Datenkonvertierung und -validierung erweitern die Anwendungsmöglichkeiten und bilden somit ebenfalls eine Grundlage im Umgang mit WPF.