Kurzreferenz („Cheat Sheet“) XAML und WPF

Autor: Dr. Joachim Fuchs (J.Fuchs@IT-Visions.de)

Version 1.2 / Oktober 2013

www.IT-Visions.de
Dr. Holger Schwichtenberg

Layout-Container

Anordnen von Kindelementen

Layout wird durch folgende Eigenschaften der Kindelemente beeinflusst:
- Width, Height
- MinWidth, MaxWidth, MinHeight, MaxHeight
- HorizontalAlignment, VerticalAlignment
- Margin

StackPanel

- Lineare Anordnung der Kindelemente
- Horizontal oder vertikal (Voreinstellung)

Grid

- Anordnung in Zeilen und Spalten
- Festlegung des Grids über Grid.RowDefinitions und Grid.ColumnDefinitions
- Zeile / Spalte kann absolut (Pixel), relativ oder automatisch berechnete Höhe / Breite besitzen
- Einordnen eines Kindelements über Attached Dependency Properties Grid.Row, Grid.Column, Grid.RowSpan, Grid.ColumnSpan

DockPanel

- Andocken oben, unten, links (Voreinstellung) oder rechts
- Steuerung über Attached Dependency Property DockPanel.Dock
- Reihenfolge der Kindelemente wichtig
- Eigenschaft LastChildFill steuert das Verhalten des letzten Kindelements

WrapPanel

- Elemente werden der Reihe nach dargestellt. Es erfolgt ein Umbruch, wenn nicht genügend Platz vorhanden ist (ähnlich Text- oder HTML-Darstellung)
- Eigenschaft Orientation steuert horizontale (Voreinstellung) oder vertikale Ausrichtung
- Sollte möglichst nur mit gleich großen Elementen verwendet werden

UniformGrid

- Grid, bei dem alle Zellen gleich groß sind
- Kindelemente werden der Reihe nach von links nach rechts eingefügt
- Automatische Berechnung des Grids als n x n Matrix
- Alternativ steuerbar über die Eigenschaften Rows und Columns
- Keine Steuerung über Attached Dependency Properties
- Reihenfolge der Elemente für Position entscheidend

Canvas

- Ausrichtung nach x/y-Koordinaten
- Attached Dependency Properties Canvas.Left, Canvas.Right, Canvas.Top, Canvas.Bottom
- Keine dynamische Anpassung

Content-Controls

Z.B. Label, Button, Window, UserControl

- Besitzen **genau ein** Kindelement für die Eigenschaft Content
- Können beliebig komplexen Inhalt haben, wenn das Kindelement beispielsweise ein Layout-Container ist
- Darstellung des Inhalts abhängig vom Typ. Bei UIElement-Ableitungen das gerenderte Objekt, sonst Anzeige des von ToString() gelieferten Textes. Überschreiben des Verhaltens durch DataTemplates möglich
- Einige Content-Controls besitzen mehrere Content-Bereiche (z. B. HeaderedContentControls wie Expander und GroupBox)

Ressourcen

Instanzieren beliebiger Objekte und bereitstellen derselben über eine Key/Value-Liste

Anlegen innerhalb der Eigenschaft Resources eines beliebigen FrameworkElements

```
<Window.Resources>
  <SolidColorBrush Color="Red" x:Key="Füllung1" />
</Window.Resources>
```

Verwenden über MarkupExtensions StaticResource oder DynamicResource

```
<Grid>
  <Ellipse Fill="{StaticResource Füllung1}" …/>
  <Ellipse Fill="{DynamicResource Füllung1}" …/>
</Grid>
```

Unterschied: Wird zur Laufzeit das Objekt, auf das die Key/Value-Liste verweist, ausgetauscht, wird nur die über DynamicResource verknüpfte Eigenschaft geändert. Änderungen des verknüpften Objekts selbst können in beiden Fällen Auswirkungen haben.

Gültigkeitsbereich
Aktueller Knoten und darunter liegende Elemente

Überall in Anwendung gültig:
Oberste Ebene der Einbindung: App.xaml

Auslagerung von Ressourcen in einem ResourceDictionary
Einbinden eines ResourceDictionaries innerhalb eines Resources-Knotens (z. B. in App.xaml):

```
<ResourceDictionary>
  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary Source="EigeneRessourcen.xaml"/>
  </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Datenbindungen

Quellen

- Grundsätzlich beliebige (.NET-) Objekte
- UI-Elemente. Angabe über ElementName
- Automatische Quelle über DataContext
- Explizite Objektreferenz über Source
- Relative Referenz über RelativeSource
- Endpunkt innerhalb der Quelle **muss eine öffentliche** (public) **Eigenschaft** sein
  - Bindung an internal oder private nicht möglich
  - Bindung an Felder nicht möglich

Das **Ziel muss** eine **Dependency Property** sein

Weitere wichtige Eigenschaften der MarkupExtension {Binding}

Path: Spezifikation der Quelle (z. B. Eigenschaftsname, Indexer)
Mode: Richtung der Datenbindung
UpdateSourceTrigger: Zeitpunkt der Änderungsbenachrichtigung der Quelle
Converter: Angabe eines Objekts, das IValueConverter implementiert
ConverterParameter: Beliebiger Wert, der bei der Konvertierung übergeben wird
StringFormat: Formatierungstext, wenn das Ziel vom Typ String ist

Spracheinstellung der Datenbindungsausdrücke

Vorsicht! Der Standard ist die Kultureinstellung im XAML-Code und hier ist die Voreinstellung en-US. Änderung entweder über xml:lang (nicht dynamisch anpassbar) oder dynamisch

```
static App()
{
  FrameworkElement.LanguageProperty.OverrideMetadata(
    typeof(FrameworkElement), new FrameworkPropertyMetadata(
      XmlLanguage.GetLanguage(CultureInfo.CurrentCulture.Name)
));
}
```

Benachrichtigung bei Änderungen

- Automatisch durch Dependency-Property-Infrastruktur bei vom Typ DependencyObject abgeleiteten Klassen
- Implementierung von INotifyPropertyChanged in anderen Datentypen
- Implementierung von INotifyCollectionChanged bei Auflistungen (z. B. ObservableCollection)

Beispiele

Bindung an den aktuellen Datenkontext
```
<TextBlock Text="{Binding}"/>
```

Bindung an die Eigenschaft Vorname des aktuellen Datenkontextes
```
<TextBlock Text="{Binding Vorname}"/>
```
oder

Kurzreferenz („Cheat Sheet“) XAML und WPF

Autor: Dr. Joachim Fuchs (J.Fuchs@IT-Visions.de)

Version 1.2 / Oktober 2013

www.IT-Visions.de
Dr. Holger Schwichtenberg

```
<TextBlock Text="{Binding Path=Vorname}"/>
```

Bindung an ein Ressourcen-Objekt mit dem Key DiePerson

```
<TextBlock Text="{Binding Source={StaticResource DiePerson}}"/
>
```

Bindung an eine TextBox mit dem Namen TextBox1

```
<TextBlock Text="{Binding Text, ElementName=TextBox1}"/>
```

Relative Bindung an das übergeordnete Window-Objekt

```
<TextBlock Text="{Binding Title,  RelativeSource={
RelativeSource Mode=FindAncestor, AncestorType=Window}}"/>
```

Umrechnen des gebundenen Wertes über einen Konverter

```
<TextBlock Foreground="{Binding Wert, Converter=
{StaticResource ZahlZuFüllungConverter}}"/>
```

Vorgabe der Bindungsrichtung

```
<TextBlock Text="{Binding Mode=OneWay}"/>
```

Benachrichtigung der Quelle bei jeder Änderung der Eigenschaft Text

```
<TextBlock Text="{Binding Path=Vorname, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
```

Formatierung von Textausgaben

```
<TextBlock Text="{Binding Path=Betrag, StringFormat=000.00}"/>
```

Data Template

Beschreibt die visuelle Repräsentation für ein Objekt

- Überschreibt das Standardverhalten (Textdarstellung von ToString())
- Besitzt genau ein Unterelement
- Kann bei Definition als Ressource als Standard für einen bestimmten Datentyp vorgesehen werden, wenn statt des Keys die Eigenschaft DataType gesetzt wird
- Typische Eigenschaften, denen DataTemplates zugewiesen werden können: ContentTemplate, ItemTemplate

Beispiel:

```
<ListBox ItemsSource="{Binding}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Vorname}" Width="100"/>
        <TextBlock Text="{Binding Nachname}"/>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Beispiel für typgebundenes DataTemplate innerhalb eines Ressourcenknotens:

```
<DataTemplate DataType="{x:Type local:Person}">
  <StackPanel Orientation="Horizontal">    …
  </StackPanel>
</DataTemplate>
```

Style

Definiert zentral Eigenschaftswerte von Objekten

- Es können nur Dependency Properties gesetzt werden
- Kann als Standard für bestimmte Objekttypen festgelegt werden (TargetType=…)
- Styles können voneinander erben (BasedOn=…)

Explizite Zuordnung eines Styles:

```
<Window.Resources>
  <Style TargetType="Label" x:Key="Stil1">
    <Setter Property="Foreground" Value="Aqua"/>
    <Setter Property="Background" Value="DarkBlue"/>
  </Style>
</Window.Resources>
<Grid>
  <Label Style="{StaticResource Stil1}" …/>
</Grid>
```

Implizite Zuordnung eines Styles:

```
<Window.Resources>
  <Style TargetType="Label" > …  </Style>
</Window.Resources>
```

Vererbung von Styles:

```
<Style TargetType="Label" x:Key="Stil1">
  …
</Style>
<Style TargetType="Label" x:Key="Stil2"
BasedOn="{StaticResource Stil1}">
  …
</Style>
```

Wichtige Schnittstellen

ICommand
Allgemeine Verknüpfung mit Controls wie Button oder MenuItem

INotifyPropertyChanged
Benachrichtigung von Bindungsausdrücken bei Änderung von Eigenschaften mittels Event PropertyChanged

IValueConverter
Klasse, die einen Wert, der über eine Datenbindung ermittelt wird, in einen anderen umrechnen kann

IDataErrorInfo
Interface, das für die Validierung bei Datenbindungen benutzt werden kann.

INotifyDataErrorInfo (ab .NET 4.5)
Ereignisbasiertes Interface für die Validierung bei Datenbindungen

TextBlock

Wichtige Eigenschaften

Text, TextWrapping

Statt einfachem Text sind auch Unterelemente möglich

```
<TextBlock>
  <Span Foreground="AliceBlue" Background="DarkBlue">
    <Run Text="Hallo " FontSize="15"/>
    <Bold><Run Text="{Binding Ausgabetext}"/></Bold>
    <Run Text="Hier geht's zu"/>
    <Hyperlink NavigateUri="http://www.it-visions.de"
               Command="{Binding NavigateToCommand}">
      <Run Text="{Binding LinkText}"/>
    </Hyperlink>
  </Span>
</TextBlock>
```

Wichtige Markup-Extensions

Binding
Datenbindungsausdrücke

StaticResource, DynamicResource
Einbinden von Resourcen

x:Static
Verweis auf statische Variablen

x:Null
Null-Referenz

x:Type
Typobjekt (zwingend bei DataTemplate.DataType)

Links

www.dotnet45.de

www.wpftutorial.net

www.windowsclient.net/wpf

www.dotnetframework.de