leren. durven. doen.

SYNTRA WEST

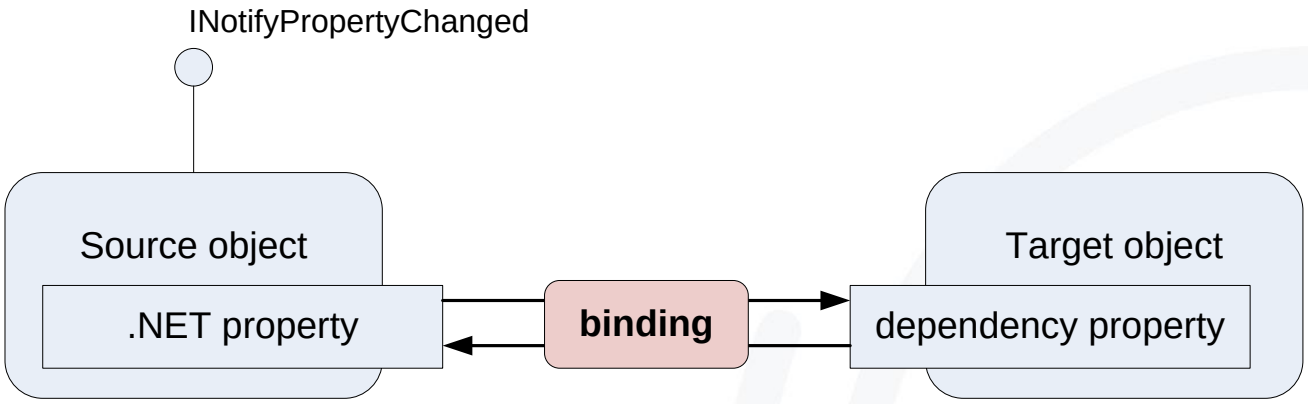
C# FUNDAMENTALS

INLEIDING WPF EN MVVM

west-vlaanderen POM

Binding

- **Properties van controls kunnen automatisch aangepast worden door properties van andere controls of model objects**
- **Updates kunnen be ‘one-way’ of ‘two way’ zijn**



Binding controls

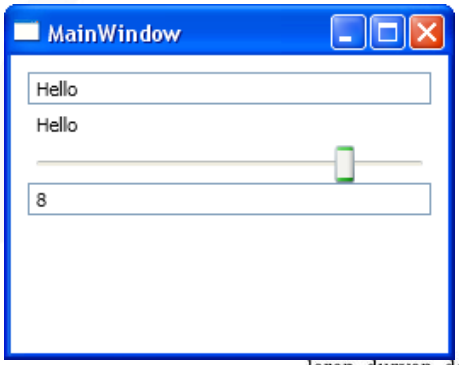
```
<StackPanel Margin="10,10,10,10">
    <TextBox x:Name="txtInput" />
    <Label Content="{Binding ElementName=txtInput,
        Path=Text}" />

    <Slider x:Name="sliderSize" Value="5" />
    <TextBox x:Name="txtSize" Text="{Binding ElementName=sliderSize,
        Path=Value,
        Mode=TwoWay,
        UpdateSourceTrigger=PropertyChanged}" />
</StackPanel>
```

Content property of Label (target) wordt gebonden aan Text property of TextBox (source)

Text property van TextBox (target) wordt gebonden aan de Value property van Slider (source)

Binding mode – wijzigingen zullen updates in 2-richtingen veroorzaken



Binding modes

- **One time**
  Source property updates de target property slechts éénmalig
- **One way**
  Source property updates de target property
- **Two way**
  Source en target properties updaten elkaar – waneer de ene verandert, verandert de andere eveneens
- **One way to source**
  Target property zal de source property updaten

Binding aan een object

Model class – voorbeeld Employee class

```
public class Employee : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void Changed(string propertyName)...

    private string name;

    public string Name
    {
        get { return name; }
        set {
            if (name != value)
            {
                name = value;
                Changed("Name:" + name);
            }
        }
    }
}
```

extra code om wijzigingen in property values voor binding aan te geven

Binding to an object

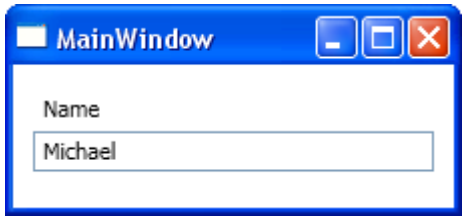
```
<TextBox x:Name="txtName"  Text="{Binding Path=Name,
    Mode=TwoWay}" />
```

- **XAML – TextBox is gebonden aan Name property, gespecifieerd door binding Path**

- **Hier wordt de source niet gespecifieerd – dit zal de data context van de window zijn**
- **Code-behind – creatie van model object zet deze als data context voor de window**

```
public MainWindow()
{
    InitializeComponent();

    Employee emp = new Employee { Name = "Michael" };
    DataContext = emp;
}
```



Binding aan een object

- Voor 2-way binding van objecten aan een UI control, moeten objecten INotifyPropertyChanged interface implementeren
- Properties moeten de PropertyChanged event oproepen
- Collections moeten van ObservableCollection zijn, deze moeten INotifyPropertyChanged en INotifyCollectionChanged implementeren

Binding aan data sources

- **Source object voor binding kan een data source zijn, bv**
  Objecten
  Entity data
  XML
- **Er zijn verschillende manieren om een binding source voor een element te specifiëren:**
  - Door gebruik van de **DataContext** property op een parent element
    Nuttig wanneer je meerdere properties wil binden aan dezelfde source
  - Door de binding **Source** property te specifiëren voor de individuele binding declaraties
  - Door de binding **ElementName** property te specifiëren om te binden aan een andere control

MVVM pattern

- Model-View-ViewModel
- WPF/Silverlight equivalent voor MVC (Model-View-Controller) en MVP (Model-View-Presenter) patterns die worden o.a. gebruikt in ASP.NET
- Geschikt voor Soc (separation of concerns) in Interactieve user interfaces
- Is te gebruiken in combinatie met WPF binding, command and templating infrastructuren

MVVM pattern (Herhaling)

- **Model**
  Domein class(es)
- **View**
  XAML die elementen bevat om data te tonen en om te reageren op user input
- **ViewModel**
  Class die het volgende bevat:
  Properties die de inhoud of status van de UI elementen bijhouden
  Commands die overeenkomen met de acties die nodig zijn om gebruikers-invoer via UI elements af te handelen
  Commands gebruiken domein-objects om business logica en de UI content (status) aan te passen

MVVM binding

- MVVM is voornamelijk geschikt voor WPF/Silverlight vanwege zijn krachtig binding mechanism
- View elementen binden aan properties en/of commands die gedefineerd worden in een ViewModel
- ViewModel hoeft niets te weten van de View
- ViewModel definieert de UI status/gedrag, maar is onafhankelijk van een specifieke UI
- UI logica in ViewModel kan worden getest via unit tests
- WPF templating voorziet controle over hoe de View de properites van de ViewModel zal presenteren in een GUI

MVVM binding

```
<ListBox Grid.Row="0"
         Grid.Column="0"
         Grid.ColumnSpan="2"
         HorizontalAlignment="Stretch"
         Margin="10"
         Name="listBox1"
         VerticalAlignment="Stretch"
         ItemsSource="{Binding Path=Persons}" >
    <ListBox.ItemTemplate>
        <DataTemplate DataType="mo:Person">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding FirstName}" />
                <TextBlock Text=" " />
                <TextBlock Text="{Binding LastName}" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

```
private ObservableCollection<Person> _persons;
public ObservableCollection<Person> Persons
{
    get { return _persons; }
    set { _persons = value; }
}
```

ItemSource wordt gebonden met *Persons* property van een ViewModel, deze bevat een collection van Person domain objecten

ItemTemplate definieert hoe properties van elk item, van type *Person*, in de ListBox moeten worden getoond – hier worden de FirstName en LastName properties getoond als text blocks in een stack panel

Visual Studio en EF data sources

- **Visual Studio laat toe om object data sources or EF entity sets te draggen-and-droppen op de WPF designer**
- **Zet de Resources element op om data sources in XAML te binden**
- **Genereert code-behind om data op te halen van de data source**
- **Elementen kunnen worden gebonden aan een source via een StaticResource**

```
<Window.Resources>
    <CollectionViewSource x:Key="customersViewSource"
                          d:DesignSource="{d:DesignInstance
                          my:Customer, CreateList=True}" />
    <CollectionViewSource x:Key="customersOrdersViewSource"
                          Source="{Binding Path=Orders,
                            Source={StaticResource customersViewSource}}" />
</Window.Resources>
<Grid DataContext="{StaticResource customersViewSource}">
```

XAML routed events

- Een WPF applicatie bevat meerdere elementen binnen elkaar
- Elementen staan in een boom-relatie tegenover elkaar
- Een ‘routed event’ is een soort event dat meerdere eventhandlers kan aanroepen in een reeks elementen die binnen elkaar zijn gedefineerd. Elk element kan zijn eigen event-listener met –handler definiëren voor een bepaald event
- De event-route vertrekt meestal van het source element en gaat dan naar boven ("bubbles“) door de element-tree tot het buitenste element (element-tree root) (meestal een window)

XAML routed events

- **De Button click event handlers kunnen op verschillend plaatsen worden gespecifieerd:**

```
<Grid Height="92" Width="396" Button.Click="CommonClickHandler">
    <Border Height="50" Width="300" BorderBrush="Gray"
            BorderThickness="2">
        <StackPanel Background="LightGray" Orientation="Horizontal"
                    Button.Click="CancelClickHandler">
            <Button Name="YesButton" Width="95" >Yes</Button>
            <Button Name="NoButton" Width="95" >No</Button>
            <Button Name="CancelButton" Width="95" >Cancel</Button>
        </StackPanel>
    </Border>
</Grid>
```

```
private void CommonClickHandler(object sender, RoutedEventArgs e)
{
    FrameworkElement feSource = e.Source as FrameworkElement;
    switch (feSource.Name)
    {
        case "YesButton":
            MessageBox.Show("Yes:Grid");
            break;
        case "NoButton":
            MessageBox.Show("No:Grid");
            break;
        case "CancelButton":
            MessageBox.Show("Cancel:Grid");
            break;
    }
    e.Handled = true;
}

private void CancelClickHandler(object sender, RoutedEventArgs e)
{
    FrameworkElement feSource = e.Source as FrameworkElement;
    switch (feSource.Name)
    {
        case "CancelButton":
            MessageBox.Show("Cancel:StackPanel");
            e.Handled = true;
            break;
    }
}
```

andere button events 'bubble up' naar Grid en worden hier eveneens afgehandeld

Dit zal niet voorkomen aangezien de Cancel button event already reeds de afhandeling heeft gedaan

Event-Handler van de Cancel button. Deze zet Handled to true zodat dit event niet meer naar boven wordt doorgegeven (prevents 'bubble up')

WPF command model

- **Geven de mogelijkheid om dezelfde actie te lanceren op verschillende manieren**
  Bv Print menu item, Print button, Ctrl+P
- **Er zijn verschillende event handlers nodig**
- **Bv indien we op bepaalde momenten printing willen 'disablen'**
  Wanneer we controls disabled moeten zetten en de shortcut-keys op bepaalde moment moeten negeren
  Is moeilijk voor maintenance en bebugging
- **WPF command model maakt het beheren van events gemakkelijker**
  Delegateert events naar de juiste commands
  Kunnen hetzelde command aan meerdere controls hangen
  Houdt de enabled state van controls gesynchroniseerd met de command state

Command model concepten

- **Command** - implementeert ICommand
  Bevat **Execute** en **CanExecute** properties, **CanExecuteChanged** event
  Representeert een command, maar bevat geen code dat de taak zelf zal uitvoeren
  De Command Library is een built-in library van common commands, zoals New, Save, Print, Copy, Paste,…
  Laat toe om custom commands aan te maken

- **Command Source – implementeert ICommandSource**
  Command, CommandTarget en CommandParameter properties
  Button, MenuItem, bv zijn command sources

Command model concepten

- **CommandBinding**
  - Command property, Executed en CanExecute events
  - Linkt een command aan de applicatie- logica (event handler)
- **Command target**
  - Element waarop de command wordt uitgevoerd
  - bv. een Paste command kan bv text in a TextBox invoeren
  - De command source kan explicitly de command target instellen
  - Indien de command target niet definieerd is, wordt standaard het element
  - the element met keyboard focus als command target genomen
  - Sommige controls kunnen command events zelf afhandelen
    - bv TextBox handelt zelf de Cut, Paste and Copy
    - Het is dan niet nodig om event handlers voor deze controls te voorzien

Command voorbeeld

- **Het zetten van de Command property van command sources, door gebruik te maken van library commands**

```
<Menu Grid.Row="0">
    <MenuItem Header="File">
        <MenuItem Command="New"></MenuItem>
        <MenuItem Command="Open"></MenuItem>
        <MenuItem Command="Save"></MenuItem>
        <MenuItem Command="SaveAs"></MenuItem>
        <Separator></Separator>
        <MenuItem Command="Close"></MenuItem>
    </MenuItem>
</Menu>

<ToolBarTray Grid.Row="1">
    <ToolBar>
        <Button Command="New">New</Button>
        <Button Command="Open">Open</Button>
        <Button Command="Save">Save</Button>
    </ToolBar>
    <ToolBar>
        <Button Command="Cut">Cut</Button>
        <Button Command="Copy">Copy</Button>
        <Button Command="Paste">Paste</Button>

    </ToolBar>
</ToolBarTray>
<TextBox Name="txt" Margin="5" Grid.Row="2"
   TextWrapping="Wrap" AcceptsReturn="True"
   TextChanged="txt_TextChanged"></TextBox>
```

TextBox zal de command target zijn als de focus erop staat

Command voorbeeld

```
<Window.CommandBindings>
    <CommandBinding Command="New"
                    Executed="NewCommand" />
    <CommandBinding Command="Open"
                    Executed="OpenCommand" />
    <CommandBinding Command="Save"
                    Executed="SaveCommand_Executed"
                    CanExecute="SaveCommand_CanExecute"/>
</Window.CommandBindings>
```

- **Het is niet nodig om event handlers voor Cut, Paste, Copy te voorzien aangezien deze voor TextBox built-in zijn**
- **Cut en Copy worden enabled wanneer de text is 'gehighlighted'**

Command voorbeeld

```
private void NewCommand(object sender, ExecutedRoutedEventArgs e)
{
    MessageBox.Show("New command triggered with " + e.Source.ToString());
    isDirty = false;
}

private void OpenCommand(object sender, ExecutedRoutedEventArgs e)
{
    isDirty = false;
}

private void SaveCommand_Executed(object sender, ExecutedRoutedEventArgs e)
{
    MessageBox.Show("Save command triggered with " + e.Source.ToString());
    isDirty = false;
}

private bool isDirty = false;

private void txt_TextChanged(object sender, RoutedEventArgs e)
{
    isDirty = true;
}

private void SaveCommand_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = isDirty;
}
```

vlag dat aanduidt of de text box text bevat dat nog niet verwerkt is (saved)

TextChanged event op target (Text Box) zal een **CanExecute** event veroorzaken op command binding, deze wordt hier gebruikt om de **CanExecute** property van Command aan te passen

Referenties

- **Drag & Drop Databinding with the Entity Framework and WPF**
  Julie Lerman
  http://msdn.microsoft.com/en-us/data/gg610409
- **Chapter 9: Programming Entity Framework**
  Julie Lerman (again!)
- **Pro WPF in C#**
  Matthew MacDonald
  Apress