

C# Mobile

Week 6

- Data binding



**DE HOGESCHOOL
MET HET NETWERK**

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be



Doelstellingen

- De junior-collega kent de hoofdonderdelen van het binding-proces
- De junior-collega kan controls aan elkaar verbinden met behulp van binding
- De junior-collega kan op een binding een verschillende modus definiëren
- De junior-collega kan de ***INotifyPropertyChanged*** interface correct implementeren op een eigen klasse
- De junior-collega kan individuele objecten en lijsten zo instellen dat ze de nodige ***events*** versturen zodat *binding* gerealiseerd kan worden
- De junior-collega kan zelf ***converters*** opstellen die binnen een *binding* waarden converteren
- De junior-collega kan een eenvoudige ***Command*** via een *binding* koppelen
- De junior-collega kan een ***compiled binding*** opstellen



Data Binding

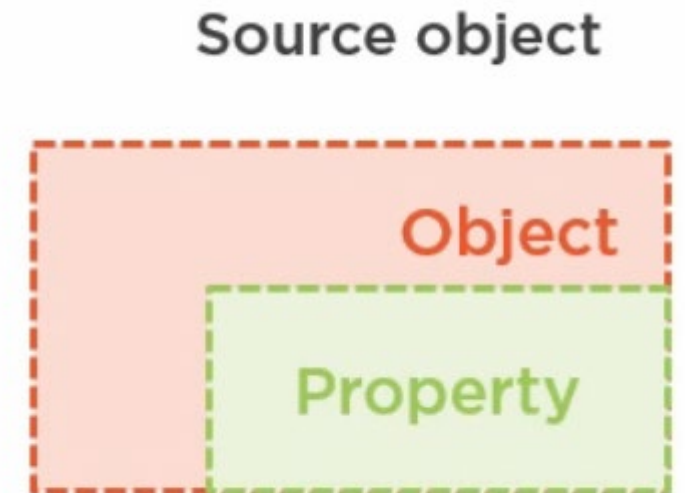
Binding

- Tot hier toe:
 - Er veranderde iets in de UI → Event
 - we passen iets aan in het achterliggende model of
 - we passen iets aan in de layout
 - Nog geen mogelijkheid om iets wat in het model veranderde te tonen in de UI

 **Binding**

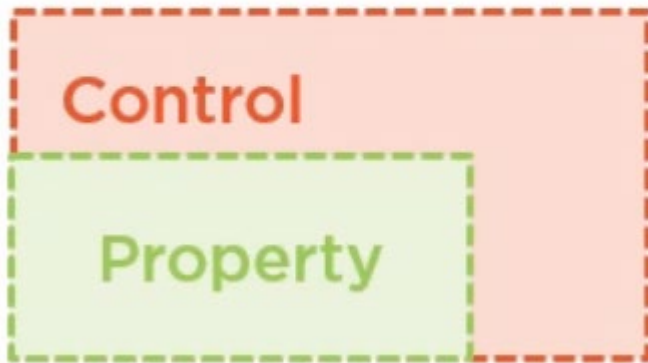
Binding: onderdelen

- **Source object**
 - Dit is het achterliggende object in C#
 - Kan bijvoorbeeld een eigen klasse zijn (bv Actor.cs)
 - Het object heeft één of meer **properties** waarvan we een representatie in de UI willen



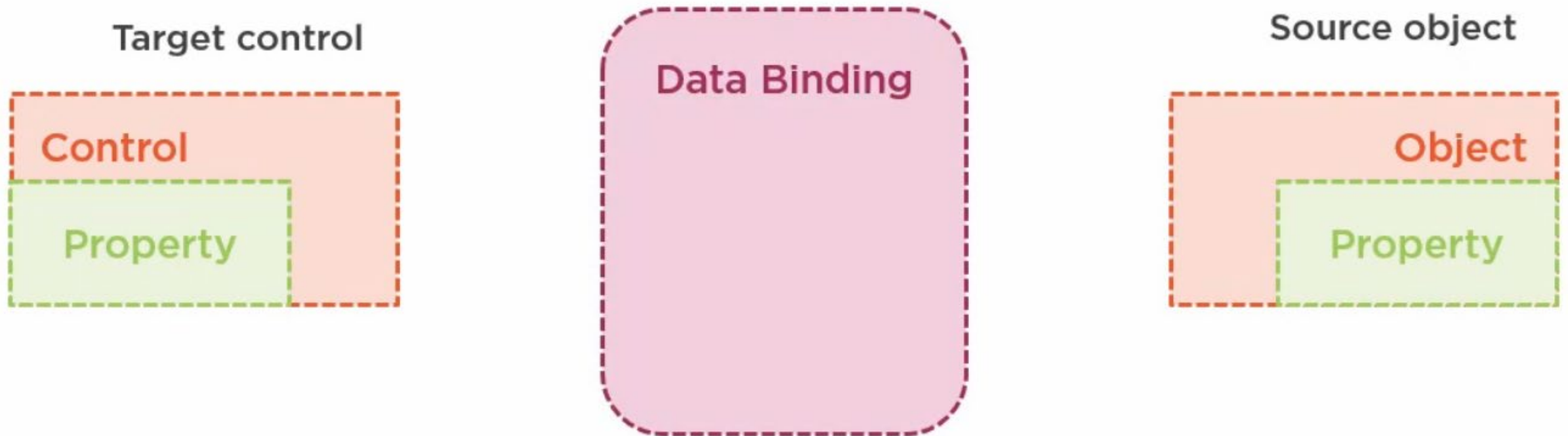
Binding: onderdelen

Target control

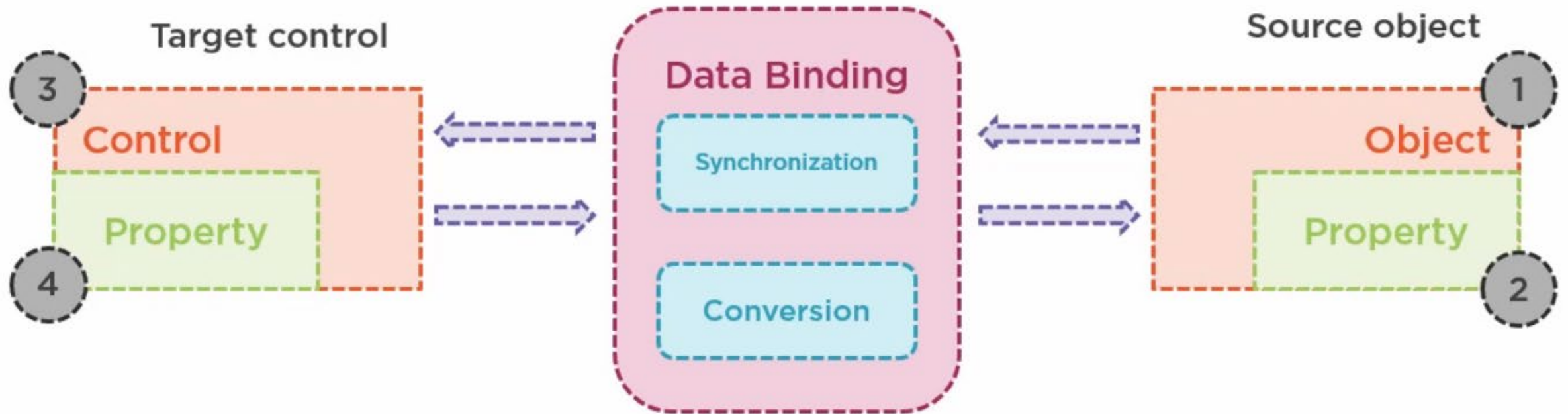


- **Target control**
 - Dit is een *control* element in de UI (meestal in XAML)
 - Bijvoorbeeld een *Label* of *Image*
 - Ook deze controls hebben ***properties*** waarin we info willen tonen

Binding: onderdelen



Binding: schema

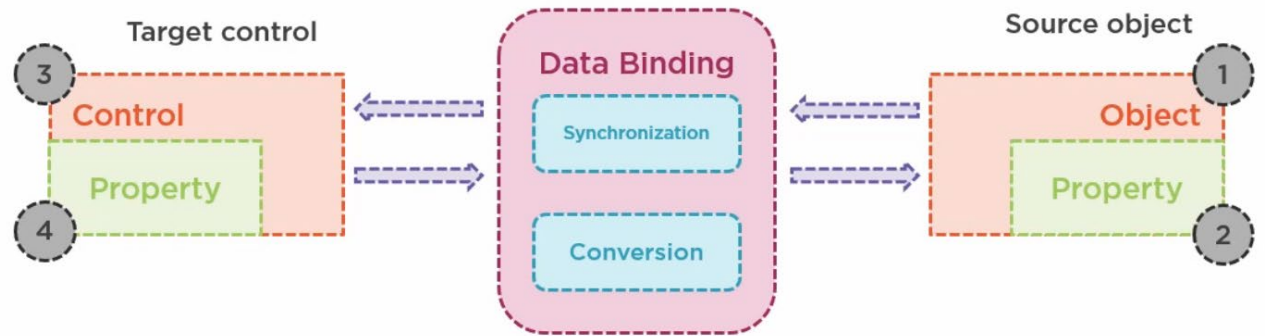


Let op: pijlen mogelijk in twee richtingen (*one- of two-way binding*)

Binding: opstellen

Definiëren van de binding

- In C#
- In XAML
- Combinatie van de twee



Binding: opstellen

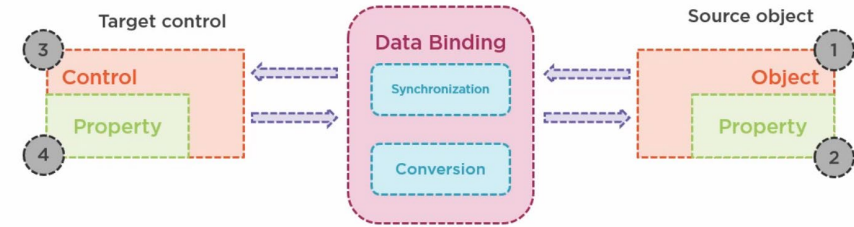
Definiëren van de binding

- In C#
- In XAML
- Combinatie van de twee

```
/* A World without data-binding: */  
//ActorImage.Source = selectedA.ProfilePicture;  
//ActorNameLabel.Text = selectedA.Name;  
  
this.BindingContext = selectedA;  
  
Label actorNameLabel = new Label();  
  
Binding actorNameBinding = new Binding();  
actorNameBinding.Source = selectedA;  
actorNameBinding.Path = "Name";  
  
actorNameLabel.SetBinding(Label.TextProperty, actorNameBinding);  
  
ActorDetailStack.Children.Add(actorNameLabel);
```

Probleem: heel omslachtig & wordt niet vaak gedaan

Binding: opstellen



Definiëren van de binding

- In C#
- In XAML
- Combinatie van de twee

```
3 <TargetControl 4
    TargetProperty="{Binding SourceProperty, Source = ...,
                        bindingProperties}" /> 2 1
```

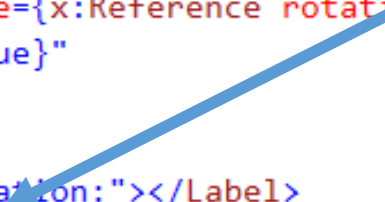
Probleem: ook het Source-object moet in XAML gedefinieerd zijn

Binding: opstellen in XAML

- Goede optie als zowel Source en Target in XAML beschikbaar zijn

```
<StackLayout Padding="10, 0">
    <Label Text="TEXT"
        FontSize="40"
        HorizontalOptions="Center"
        VerticalOptions="CenterAndExpand"
        Rotation="{Binding Source={x:Reference rotation_slider},
            Path=Value}"
    />


    <Label Text="Slider for rotation:"></Label>
    <Slider x:Name="rotation_slider"
        Value="0"
        Minimum="0"
        Maximum="360"
        VerticalOptions="CenterAndExpand" />
```



Binding: opstellen in XAML

- Goede optie als zowel Source en Target in XAML beschikbaar zijn
- Maakt gebruik van x:Reference

```
<StackLayout Padding="10, 0">  
3 <Label Text="TEXT"  
  FontSize="40"  
  HorizontalOptions="Center"  
  VerticalOptions="CenterAndExpand"  
  Rotation="{Binding Source={x:Reference rotation_slider},  
4    Path=Value}"  
/>  
  
1 <Label Text="Slider for rotation:"></Label>  
2 <Slider x:Name="rotation_slider"  
  Value="0"  
  Minimum="0"  
  Maximum="360"  
  VerticalOptions="CenterAndExpand" />
```



Binding: opstellen

- **Meestal: object in C#, control in XAML**
=> Combinatie voor definiëren binding

Binding: opstellen

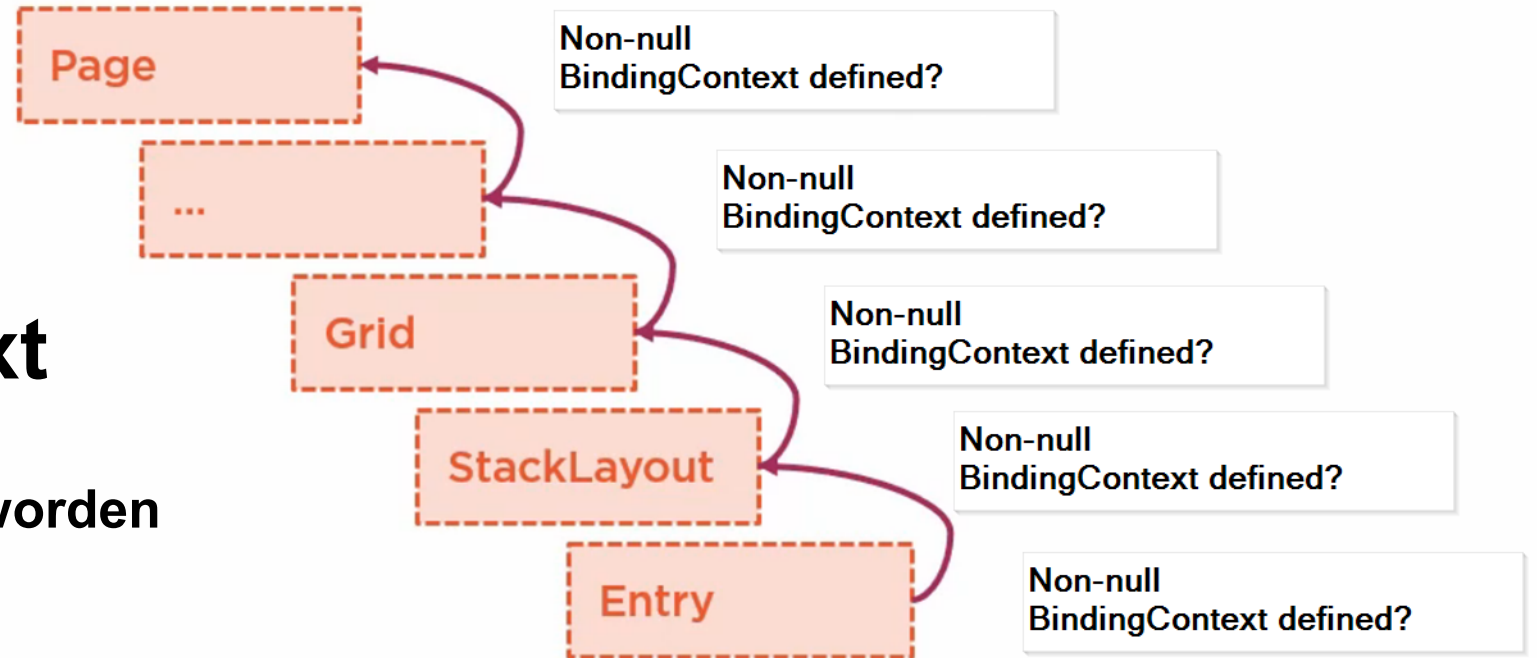
Definiëren van de binding

- In C#
- In XAML
- Combinatie

⇒ **BindingContext**

Kan op elk XAML element worden ingesteld

Gebruikt *Tree Walking* om op zoek te gaan naar het *Source object*



Voorbeeld: ActorsApplication

In het voorbeeld hebben we een eigen klasse Actor:

```
public class Actor
{
    3 references
    public string Name { get; set; }
    3 references
    public string FirstName { get; set; }
    2 references
    public int BirthYear { get; set; }
    2 references
    public string ProfilePicture { get; set; }
}
```

Binding: opstellen

Definiëren van de binding

- In C#
- In XAML
- Combinatie
⇒ **BindingContext**

In voorbeeld:
ProfilePicture wordt gezocht op
een object dat *gebund* is
In dit geval: op de **StackLayout**

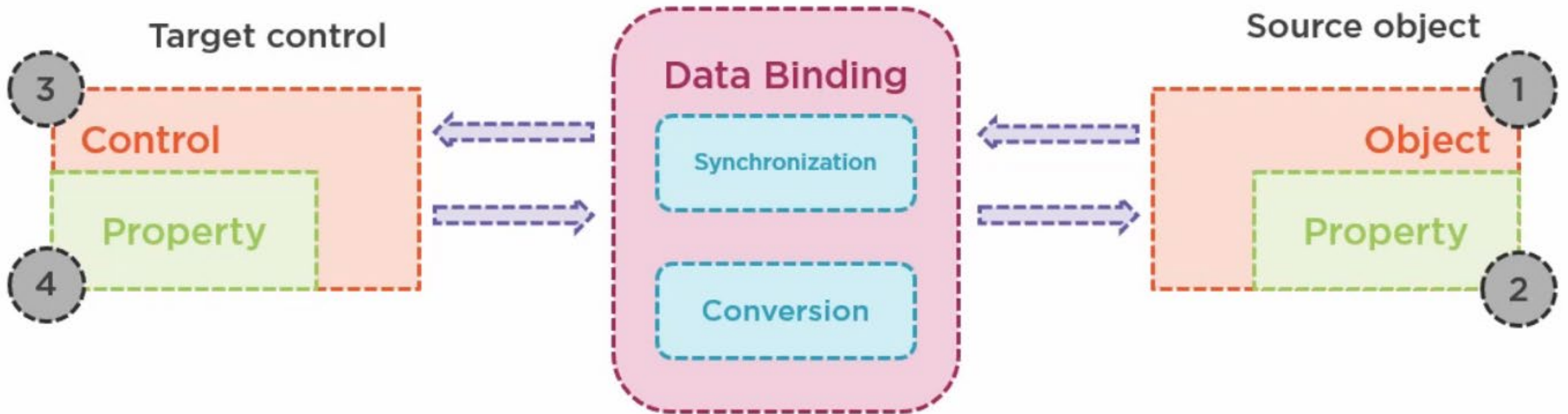
```
ActorDetail.xaml.cs  X
BindingSamples (net6.0-android)  BindingSamples.ActorDetail

25 public ActorDetail()
26 {
27     InitializeComponent();
28
29     Actor selectedActor = new Actor()
30     {
31         BirthYear = 1930,
32         Name = "Connery",
33         FirstName = "Sean",
34         ProfilePicture = "https://upload.wikimedia.org
35     };
36
37     ActorDetailStack.BindingContext = selectedActor;
38 }
```

```
ActorDetail.xaml  X
ContentPage  ContentPage

5 Title="ActorDetail">
6 <VerticalStackLayout VerticalOptions="Center" x:Name="ActorDetailStack">
7
8     <Image WidthRequest="250" HeightRequest="250" Aspect="AspectFill"
9         Source="{Binding ProfilePicture}"
10     />
11     <Label VerticalOptions="Center" HorizontalOptions="Center"
12         Text="{Binding Name}"
13     />
14     <Label VerticalOptions="Center" HorizontalOptions="Center"
15         Text="{Binding FirstName}"
16     />
17     <Label VerticalOptions="Center" HorizontalOptions="Center"
18         Text="{Binding BirthYear}"
19     />
20
21 </VerticalStackLayout>
22
```

Binding: modes



Binding: modes

- **Default**
- **OneTime**
Eenmalig van source naar target, maar alleen als de BindingContext wijzigt
- **OneWay**
Veranderingen in source worden doorgevoerd naar target
- **TwoWay**
Veranderingen worden in twee richtingen doorgevoerd
- **OneWayToSource**
Verandering in target (control) worden doorgegeven naar source (default bij SelectedItem in ListView)

Binding: modes

- **Default =>**
- OneTime
- OneWay
- TwoWay
- OneWayToSource

Most bindable properties have a default binding mode of `OneWay` but the following properties have a default binding mode of `TwoWay` :

- `Date` property of `DatePicker`
- `Text` property of `Editor`, `Entry`, `SearchBar`, and `EntryCell`
- `IsRefreshing` property of `ListView`
- `SelectedItem` property of `MultiPage`
- `SelectedIndex` and `SelectedItem` properties of `Picker`
- `Value` property of `Slider` and `Stepper`
- `IsToggled` property of `Switch`
- `On` property of `SwitchCell`
- `Time` property of `TimePicker`

Zie documentatie:

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/binding-mode>

Binding: INotifyPropertyChanged

Bij binding van Target naar Source:

Target moet weten indien er aanpassing is in de Source

Luistert naar PropertyChanged Event

Eigen objecten moeten dit Event uitsturen

=> INotifyPropertyChanged-interface

```
//  
// Summary:  
//     Notifies clients that a property value has changed.  
public interface INotifyPropertyChanged  
{  
    //  
    // Summary:  
    //     Occurs when a property value changes.  
    event PropertyChangedEventHandler PropertyChanged;  
}
```

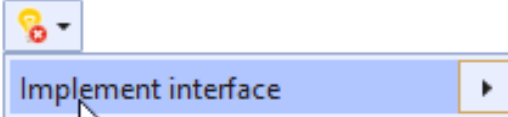
INotifyPropertyChanged: stappen

Stap 1: Klasse implementeert de interface

```
public class Actor: INotifyPropertyChanged  
{  
    .  
    .  
    .  
}
```

Stap 2: Klasse implementeert de interface

```
public class Actor: INotifyPropertyChanged  
{  
    .  
    .  
    .  
}
```



INotifyPropertyChanged: stappen

Stap 3: Maak method die het event kan afvuren

```
public void RaiseEventPropChanged(string propertyname)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyname));
}
```

Stap 4: Properties aanpassen zodat ze het event uitsturen

```
public string Name { get; set; }
```

```
private string _name;
public string Name { get => _name;
    set {
        _name = value;
        RaiseEventPropChanged(nameof(Name));
    }
}
```


INotifyPropertyChanged: stappen

Stap 4: Properties aanpassen zodat ze het event uitsturen

```
private string _name;  
public string Name { get => _name;  
    set {  
        _name = value;  
        RaiseEventPropChanged(nameof(Name));  
    }  
}
```

4.1 Private veld toegevoegd

4.2 get geeft de waarde terug

4.3 set stelt de waarde in en vuurt het event af!

(Stap 5: Herhalen voor alle properties die 'afgeluisterd' moeten worden)



Binding to Single Object

Binding to Single Object

- **Via BindingContext**
- **Up-to-date binding nodig?**
 - `INotifyPropertyChanged` implementeren op de eigen klasse

```
private string _name;  
[REDACTED]  
public string Name { get => _name;  
    set {  
        _name = value;  
        RaiseEventPropChanged(nameof(Name));  
    }  
}
```



Binding to a Collection

Binding to a Collection

- Hebben we eigenlijk al gedaan!
 - ⇒ Collectionview: ItemsSource is een binding
 - ⇒ Ook in de DataTemplate hebben we al binding gebruikt:
`<Image Grid.Row="0" Grid.Column="0" Source="{Binding ProfilePictureURL}"></Image>`

Binding to a Collection

- Toevoegen: we kunnen een nieuw item toevoegen met behulp van de Add-operatie
- Voorbeeld:

```
private void AddActor(object sender, EventArgs e)
{
    Actor newActor = new Actor { Name = "Test Actor" };
    _listActors.Add(newActor);
}
```

Binding to a Collection

```
private void AddActor(object sender, EventArgs e)
{
    Actor newActor = new Actor { Name = "Test Actor" };
    _listActors.Add(newActor);
}
```

- Wat gebeurt er als we iets toevoegen aan de lijst? Niets.
 - Ook de lijst moet een event sturen, zoals bij de properties van onze eigen klasse
=> ObservableCollection!

```
List<Acteur> actorList = new List<Acteur> {  
  
ObservableCollection<Acteur> actorList = new ObservableCollection<Acteur> {
```

Binding to a Collection

- Verwijderen: als CommandParameter kan je het hele object meegeven als “.”
- Voorbeeld toevoegen en verwijderen uit lijst:

```
0 references
private void AddAnActor(object sender, EventArgs e)
{
    actorList.Add(new Actor() { Name = "Test" });
}
```

```
0 references
private void DeleteActor(object sender, EventArgs e)
{
    ImageButton clickedButton = (sender as ImageButton);
    actorList.Remove(clickedButton.CommandParameter as Actor);
}
```

```
<Label VerticalOptions="Center" Margin="15,0" Text="{Binding Name}"></Label>
<ImageButton Source="trashicon.png"
    Clicked="DeleteActor" CommandParameter="{Binding .}" />
</HorizontalStackLayout>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
<Button Text="Add Actor" Clicked="AddAnActor"></Button>
</VerticalStackLayout>
```

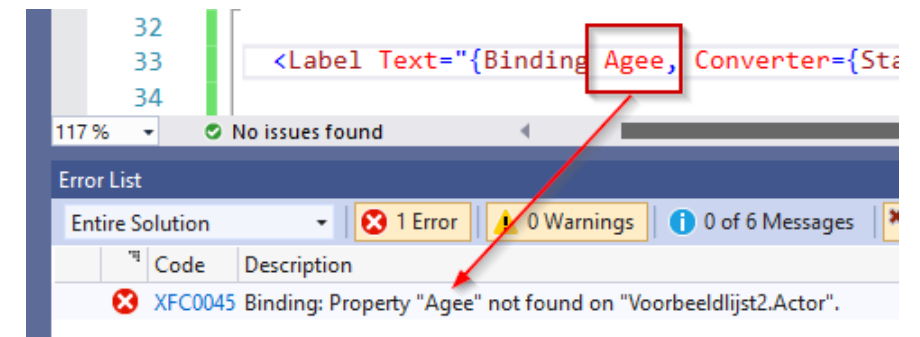


Advanced Binding Concepts

Compiled Data Binding

- We zien fouten in *bindings* soms pas at run-time
 - De applicatie zoekt de bindings via *reflection*
 - *Traag*
 - *Foutgevoelig*

=> *Compiled Data Binding*
- Aangeven in XAML met welk object we gaan binden:
 - Via x:DataType `<StackLayout x:DataType="local:Actor">`
 - Bij het *compilen* wordt nu al een error getoond!



Converters

- Een converter 'haakt' in tijdens de binding
- Bij het setten&veranderen van een value uit de binding wordt converter opgeroepen
- Gewone C# klasse die de **IValueConverter** implementeert

```
0 references
public class AgeConverter : IValueConverter
{
    0 references
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    0 references
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

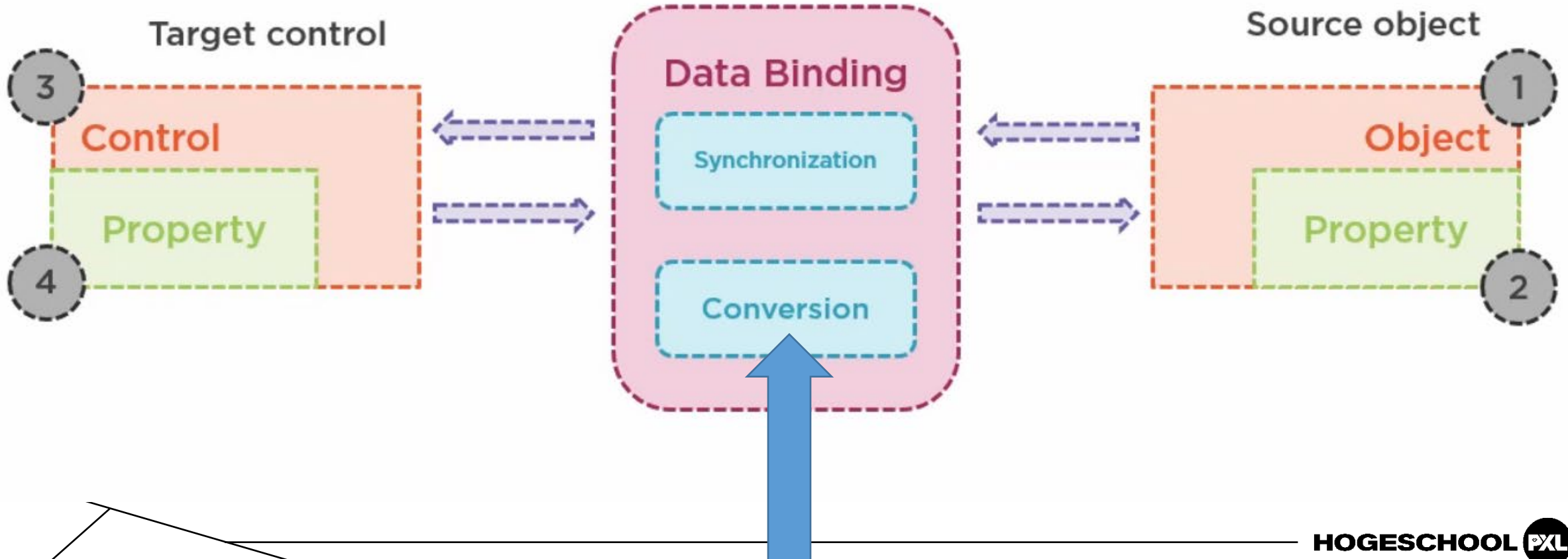
Converters

- Interface met twee methods:
 - **Convert**: opgeroepen bij binding van source naar target
 - **ConvertBack**: opgeroepen als er in de target iets verandert en dit ook in de source moet aanpassen (bijvoorbeeld bij *TwoWay*)

```
0 references
public class AgeConverter : IValueConverter
{
    0 references
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    0 references
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Converters: schema



Converters: in de praktijk

- Toevoegen aan de Resources (van App):

```
<Application.Resources>
  <ResourceDictionary>
    <converter:AgeConverter x:Key="AgeConverter"></converter:AgeConverter>
  </ResourceDictionary>
</Application.Resources>
```

- Toevoegen aan de Binding:

```
<Label Text="{Binding Age, Converter={StaticResource AgeConverter}}" ></Label>
```

- Kan ook gebruikt worden om naar een totaal ander object om te zetten!
 - Bijvoorbeeld een nummer naar een kleur
 - Via *ConverterParameter* kan bovendien een extra parameter meegeven worden

Converters: built-in

- **StringFormat**
 - Heel snel een String formateren
 - Single quotes rond de format
 - Voorbeelden: zie [link](#)
- FallbackValue
- TargetNullValue

```
<Slider x:Name="slider" />
<Label Text="{Binding Source={x:Reference slider},
                    Path=Value,
                    StringFormat='The slider value is {0:F2}}'" />

<BoxView />

<TimePicker x:Name="timePicker" />
<Label Text="{Binding Source={x:Reference timePicker},
                    Path=Time,
                    StringFormat='The TimeSpan is {0:c}}'" />

<BoxView />

<Entry x:Name="entry" />
<Label Text="{Binding Source={x:Reference entry},
                    Path=Text,
                    StringFormat='The Entry text is &quot;{0}&quot;}'" />

<BoxView />

<StackLayout BindingContext="{x:Static sys:DateTime.Now}">
    <Label Text="{Binding}" />
    <Label Text="{Binding Path=Ticks,
                        StringFormat='{0:N0} ticks since 1/1/1}'" />
    <Label Text="{Binding StringFormat='The {{0:MMMM}} specifier produces {0:MMMM}}'" />
    <Label Text="{Binding StringFormat='The long date is {0:D}}'" />
</StackLayout>

<BoxView />

<StackLayout BindingContext="{x:Static sys:Math.PI}">
    <Label Text="{Binding}" />
    <Label Text="{Binding StringFormat='PI to 4 decimal points = {0:F4}}'" />
    <Label Text="{Binding StringFormat='PI in scientific notation = {0:E7}}'" />
</StackLayout>
```

Converters: built-in

- StringFormat
- **FallbackValue**
 - Binding niet gevonden -> Default waarde meegeven

```
<Label Text="{Binding Population, FallbackValue='Population size unknown'}"  
... />
```

- TargetNullValue

Converters: built-in

- StringFormat
- FallbackValue
- **TargetNullValue**
 - Path wel gevonden MAAR **null** waarde op de binding?
 - > Verander dit naar iets anders
 - Handig om bijvoorbeeld een default image te tonen:

```
<Image Source="{Binding ImageUrl, TargetNullValue={StaticResource fallbackImageUrl}}"
... />
<Label Text="{Binding Location, TargetNullValue={StaticResource locationUnknown}}"
... />
```

```
<!-- BindingSamplesApp -->
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
      <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
    </ResourceDictionary.MergedDictionaries>
    <x:String x:Key="fallbackImageUrl">https://upload.wikimedia.org/wik:
    <x:String x:Key="locationUnknown">Location unknown</x:String>
  </ResourceDictionary>
</Application.Resources>
```



Baboon

Africa & Asia



Seated Monkey

Location unknown



Capuchin Monkey

Central & South America



Blue Monkey

Central and East Africa



Squirrel Monkey

Central & South America



Face-Palm Monkey

Location unknown



Golden Lion Tamarin

Brazil

Multibinding

- Meerdere bindings combineren (zie link)
- Handige ingebouwde multibinding: Format Strings

```
<Label>
  <Label.Text>
    <MultiBinding StringFormat="{0} {1} {2}">
      <Binding Path="Employee1.Forename" />
      <Binding Path="Employee1.MiddleName" />
      <Binding Path="Employee1.Surname" />
    </MultiBinding>
  </Label.Text>
</Label>
```

Commands

- Via binding kunnen we niet enkel waarden tonen/aanpassen
- Ook methodes die uitgevoerd kunnen worden!
=> Commands
- De property met de command moet de ICommand-interface implementeren

```
public interface ICommand
{
    public void Execute (Object parameter);
    public bool CanExecute (Object parameter);
    public event EventHandler CanExecuteChanged;
}
```

Commands

- Property toevoegen aan het object dat aan de BindingContext gebonden is:

```
0 references  
public ICommand AddActorCommand { get; private set; }
```

- Command instellen (bijvoorbeeld in de constructor):
 - Twee functies kunnen toegevoegd worden: execute en canExecute (optioneel)


```
AddActorCommand = new Command(  
    ▲ 3 of 4 ▼ Command(Action execute, Func<bool> canExecute)  
    Initializes a new instance of the Command class.  
    execute: An Action to execute when the Command is executed.
```

- Bijvoorbeeld:

```
AddActorCommand = new Command(() => { actorList.Add(new Actor() { Name = "Test" }); });
```

```
AddActorCommand = new Command(AddActor);
```

```
0 references  
private void AddActor()  
{  
    actorList.Add(new Actor() { Name = "Test" });  
}
```



Extra bronnen

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/>

<https://app.pluralsight.com/library/courses/wpf-6-fundamentals/table-of-contents>