# Databinding en DataTemplates

Contents

[Databinding en DataTemplates 1](#_Toc431970444)

[Voorbereiding 2](#_Toc431970445)

[DataBinding 3](#_Toc431970446)

[DataBinding: inleiding 3](#_Toc431970447)

[DataBinding via XAML: de markup extension {Binding 3](#_Toc431970448)

[DataBinding via XAML: {Binding en DataContext 3](#_Toc431970449)

[DataBinding via XAML: {Binding en ElementName 4](#_Toc431970450)

[DataBinding via XAML: {Binding en Source 4](#_Toc431970451)

[DataBinding via de ambient property DataContext 5](#_Toc431970452)

[Ambient properties 5](#_Toc431970453)

[DataContext: een ambient property! 5](#_Toc431970454)

[DataContext en ItemsControl 6](#_Toc431970455)

[Visualisatie voor model klasses instanties 7](#_Toc431970456)

[Huidige oplossing: UserControls 7](#_Toc431970457)

[Binding oplossing: Datatemplates 8](#_Toc431970458)

[Wat is een DataTemplate? 8](#_Toc431970459)

[DataTemplate en Binding 8](#_Toc431970460)

[Hoe/waarom een datatemplate gebruiken? 8](#_Toc431970461)

[ItemsControl.ItemTemplate property 8](#_Toc431970462)

[DataTemplates In het algemeen: 10](#_Toc431970463)

[DataTemplate resource 10](#_Toc431970464)

[DataTemplateSelector (niet voor deze les) 11](#_Toc431970465)

[{Binding Mode=TwoWay 11](#_Toc431970466)

[INotifyPropertyChanged Interface 12](#_Toc431970467)

[Dependency properties 13](#_Toc431970468)

[{x:Bind markup extension 13](#_Toc431970469)

[Binding via C# code (niet te kennen voor het examen) 14](#_Toc431970470)

[Bindings en de IValueConverter interface 14](#_Toc431970471)

[Blend en DataTemplates 15](#_Toc431970472)

[Binding: stand van zaken na deze les 15](#_Toc431970473)

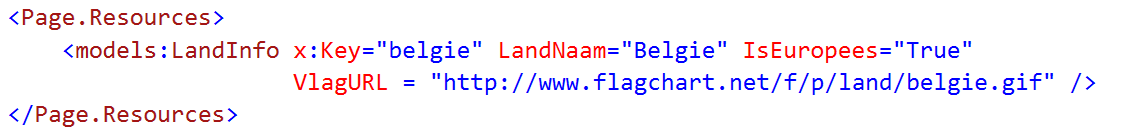
## Voorbereiding

De voorbeelden in deze cursus maken gebruik van de model klasse LandInfo en zijn bijhorende repository klasse:

* LandNaam: String
* VlagURL: String, verwijst naar een image van de vlag van het land
* IsEuropees: Boolean

De constructie van de klasse LandInfo is dusdanig dat er verschillende property types (String en Boolean) zijn waarbij één van de String waardes verwijst naar een image. Doorheen de voorbeelden zal hierdoor duidelijk worden dat Binding voor alle types mogelijk is en een beperkte set impliciete typeconversies gedefinieerd is.

Indien we in de voorbeelden gebruik maken van de StaticResource belgie dan is die gedefinieerd als volgt:



De StaticResource landrepos is gedefinieerd als volgt:

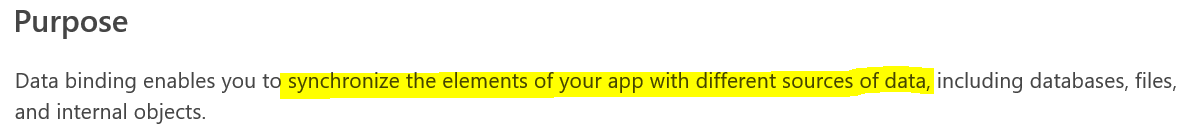


Via zijn Landen propertie wordt een lijst van LandInfo objecten aangeleverd

## DataBinding

### DataBinding: inleiding

Voor wie graag zelf de documentatie doorneemt: [Data Binding (XAML)](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh758318.aspx) is een goed startpunt, met daarin onder andere volgende beschrijving:



In deze les context: Data binding zal gebruikt worden om UI- properties te synchroniseren met model object properties. De synchronisatie zal in beide richtingen (TwoWay) gebeuren: UI- aanpassingen worden doorgegeven aan het achterliggende model object, en wijzigingen in de model state wordt gereflecteerd in de UI.

### DataBinding via XAML: de markup extension {Binding

#### DataBinding via XAML: {Binding en DataContext

Om met de deur in huis te vallen, onderstaande XAML code definieert een Binding:



* de Text property van de TextBlock wordt gebind aan de LandNaam property
* By default zal de LandNaam gezocht worden in de DataContext van de control, in dit geval de StaticResource belgie.

Een gelijkaardige binding, met een ander datatype:



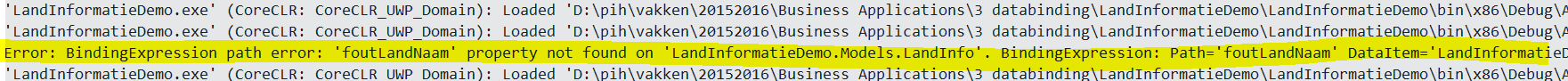
* De Source property van de Image wordt gebind aan de VlagURL property (case sensitive XML!)
* Er gebeurt een impliciete typeconversie van het type String (VlagURL) naar het verwachte type ImageSource
* By default zal de VlagURL gezocht worden in de DataContext van de control, in dit geval de StaticResource belgie.
* De binding source levert de waarde die getoond zal worden
* De binding target is de control wiens properties gebind worden aan de binding source

Te onthouden: de tekst **ctlprop=”{Binding Path=dcprop}”** copieert de waarde van de datacontext property **dcprop** naar de controlproperty **ctlprop**.

Opmerking: in de Binding is het Path attribuut de default. U kan de bovenstaande XAML code verkort noteren als:

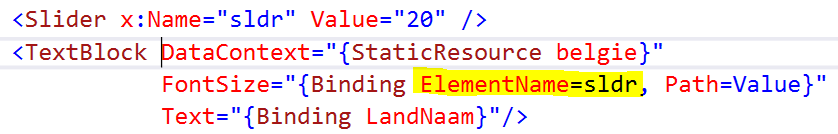


Opmerking: iedereen maakt wel eens een tikfout: Een foutieve property naam in de Binding levert at runtime volgende boodschap in het outputwindow:



#### DataBinding via XAML: {Binding en ElementName

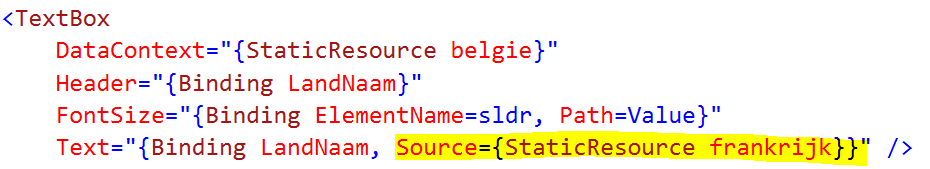
DataBinding kan ook properties van verschillende UI- elementen met elkaar verbinden:



* Indien de Binding een ElementName gebruikt moet deze de naam van een control zijn. Deze control wordt dan voor deze binding gebruikt in plaats van de DataContext
* In bovenstaande XAML code heeft de TextBlock twee bound properties:
  + Text is bound aan LandNaam, gevonden in de DataContext
  + FontSize is bound aan Value, gevonden in sldr

#### DataBinding via XAML: {Binding en Source

DataBinding kan ook properties van een ander object dan de DataContext raadplegen:

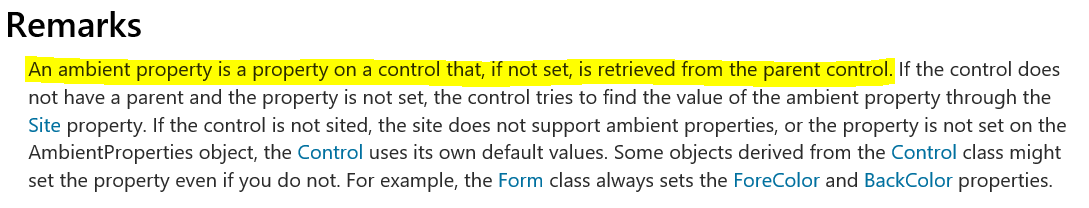


* Indien de Binding een Source gebruikt zal deze in deze les een StaticResource zijn. Deze StaticResource wordt dan voor deze binding gebruikt in plaats van de DataContext
* Bovenstaande TextBox definieert drie bindings die elk hun informatie uit een verschillend object ophalen:
  + De Header wordt bekomen via de LandNaam property van de DataContext;
  + De FontSize wordt bekomen via de Value property van het element sldr;
  + De Text wordt bekomen via de LandNaam property van de static resource frankrijk

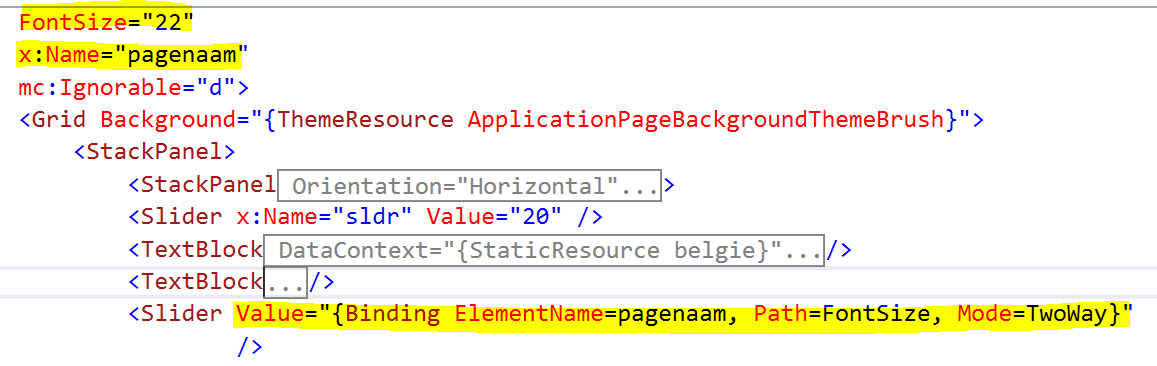
Opmerking: weet u wat de Header property van een TextBox is?

### DataBinding via de ambient property DataContext

#### Ambient properties



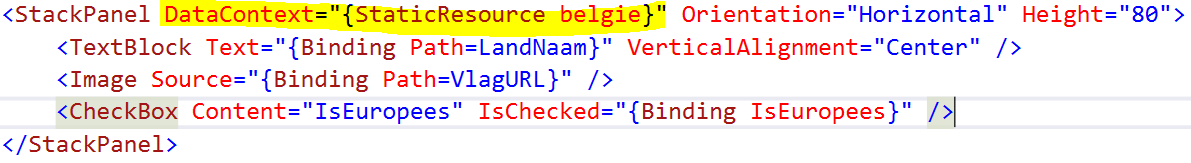
Een voorbeeld kan veel verhelderen:



* De ElementName in de binding geeft aan dat de bron van de binding de pagina zelf is
* De startwaarde van de slider wordt overgenomen van de FontSize van de pagina en is dus 22
* Door de Mode=TwoWay declaratie zal een UI wijziging naar de binding source worden doorgegeven. We kunnen dus de page fontsize aanpassen
* Omdat FontSize een ambient property is zullen de TextBlock controls de FontSize waarde erven van hun parent tenzij ze zelf de FontSize instellen. Het verschuiven van de slider zal dan ook de lettergrootte van die TextBlock controls aanpassen (demo)

#### DataContext: een ambient property!

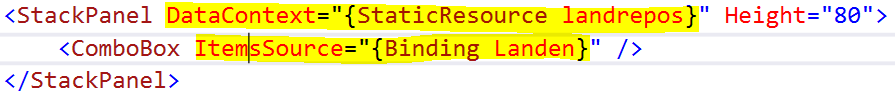
Het is geen toeval dat de DataContext property ook een ambient property is! Deze featur zal het mogelijk maken om complexe UI met weinig inspanning de juiste informatie te laten tonen!



* De TextBlock, Image en CheckBox control definiëren geen DataContext en bekomen dan ook hun DataContext van hun parent (het StackPanel, met StaticResource belgie);
* Omdat de Bindings geen Source of ElementName opgeven wordt de Source van de Bindings het DataContext object
* De drie controls tonen dan ook de informatie die wordt opgehaald uit de StaticResource belgie

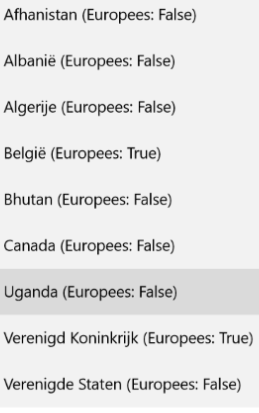
#### DataContext en ItemsControl

De [ItemsControl class (Represents a control that can be used to present a collection of items.)](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.ui.xaml.controls.itemscontrol.aspx) definieert de base class waarvan lijst- controls erven. Deze klasse introduceert de ItemsSource property die gebind kan worden aan een lijst van objecten. Indien dit het geval is zal de datacontext, gebruikt bij de visualisatie van elk van deze objecten het te visualiseren object zelf zijn:



* De DataContext van het StackPanel wordt doorgegeven aan de ComboBox;
* De Landen property geeft een lijst van LandInfo objecten terug
* Elk landinfo object in de ComboBox zal, wanneer het getoond wordt, zichzelf als DataContext op zijn visualisatie instellen:

De combobox weet niet hoe hij een LandInfo object goed moet voorstellen, en toont de ToString waarde in een TextBlock. Het zal de taak van de ontwikkelaar zijn om voor de klasse LandInfo een goede visualisatie te voorzien voor de ComboBox.

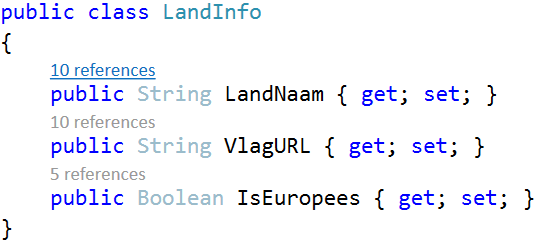


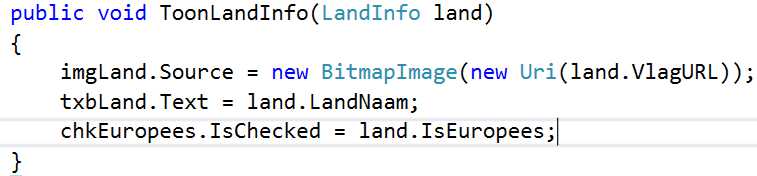
Een aangepast ToString methode levert volgende output:

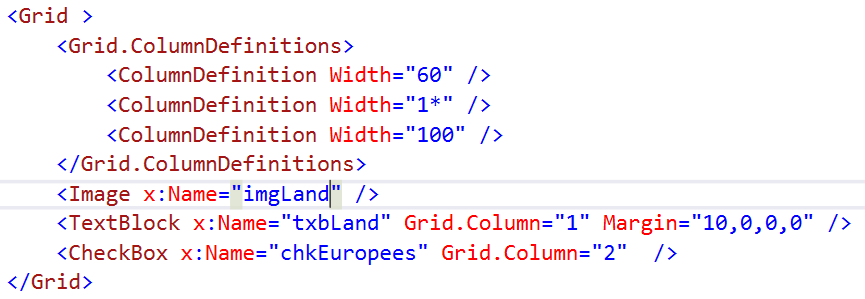
## Visualisatie voor model klasses instanties

### Huidige oplossing: UserControls

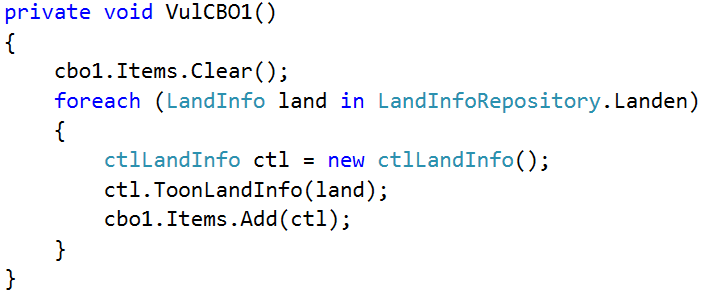
* Informatie wordt via de repositories in model instanties geplaatst. In onderstaand voorbeeld is LandInfo een model klasse:



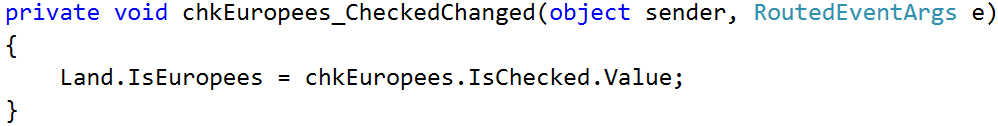
* Voor elke model klasse waarvan de instanties gevisualiseerd moeten worden voorzien we een usercontrol klasse
  + Deze usercontrol klasse bestaat uit twee componenten:
    - .xaml- file die de visuele elementen bevat en *styled*;
    - .cs file die de model property waardes copieert naar de xaml properties



* De koppeling tussen de visuele elementen en de model- properties is [Boilerplate code](https://en.wikipedia.org/wiki/Boilerplate_code). Het hoeft dan ook niet te verbazen dat het definiëren van deze koppelingen via een .NET infrastructuur (DataTemplates) ondersteund wordt (topic van deze les: Binding)
* Het tonen van een lijst Model- instanties gebeurt door voor elk te tonen object een control aan te maken die dit object toont:



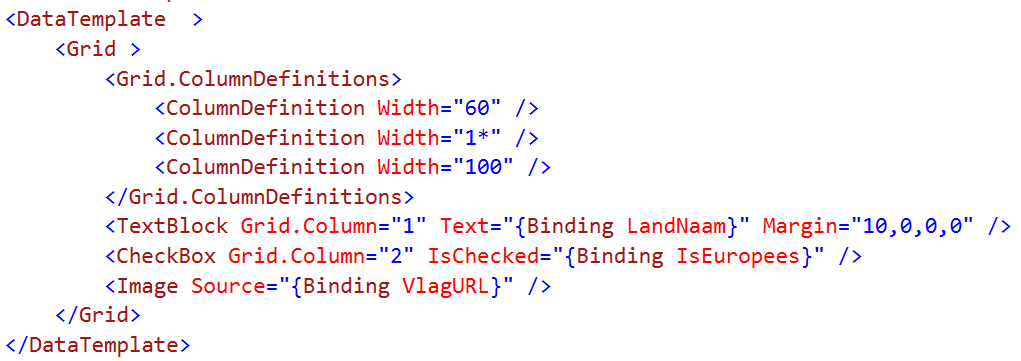
* Indien we de UI aanpassingen van de gebruiker willen terugkoppelen naar het model coderen we eventhandlers:



* Indien properties van het model wijzigen (wellicht door C# code) hebben we in de beperkte usercontrol benadering geen weet van de wijziging en zullen we dus de visualisatie ook niet aanpassen.

### Binding oplossing: Datatemplates

#### Wat is een DataTemplate?



Bovenstaande DataTemplate definieert een visuele structuur die bruikbaar is om een LandInfo object te visualiseren:

### DataTemplate en Binding

Essentieel is het *binden* van de XAML properties aan de model properties. Deze bindings zullen er voor zorgen dat de model properties gebruikt worden om de XAML properties in te stellen

* Bovenstaande werkt enkel indien de bron van de informatie een LandInfo object zal zijn (dit staat **NIET** in de template). Dit zal gerealiseerd worden door de DataContext in te stellen!
* De koppeling tussen de XAML source (type **ImageSource**) en de model VlagURL (type **String**) toont aan dat het Binding mechanisme typeconversies ondersteunt
* De koppelingen beperken zich niet tot String types: zolang het toegekende type toekenbaar is aan het XAML- property type (eventueel via (impliciete of expliciete) dataconversie) kan een binding geschreven worden.

### Hoe/waarom een datatemplate gebruiken?

Het is de bedoeling om met de datatemplates (en zijn bindings) het gebruik van een usercontrol overbodig te maken : De datatemplate bevat de XAML-code die in de .xaml file van de usercontrol wordt gevonden, en de Binding legt de verbanden die in de .cs file worden geprogrammeerd.

#### [ItemsControl.ItemTemplate property](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.ui.xaml.controls.itemscontrol.itemtemplate.aspx)

De [ItemsControl class (Represents a control that can be used to present a collection of items.)](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.ui.xaml.controls.itemscontrol.aspx) definieert de ItemTemplate property waarmee de layout van zijn items beschreven wordt. Elke klasse die erft van ItemsControl erft deze property, en dus de mogelijkheid om de opmaak van zijn items via de ItemTemplate te beschrijven. Een (beperkte) lijst controls die erven van ItemsControl is: ListView, GridView, ListBox, ComboBox, FlipView, MenuFlyoutPresenter, ..). Onderstaande XAML-code zal op de ComboBox de ItemTemplate property instellen die toegepast zal worden op al zijn items:

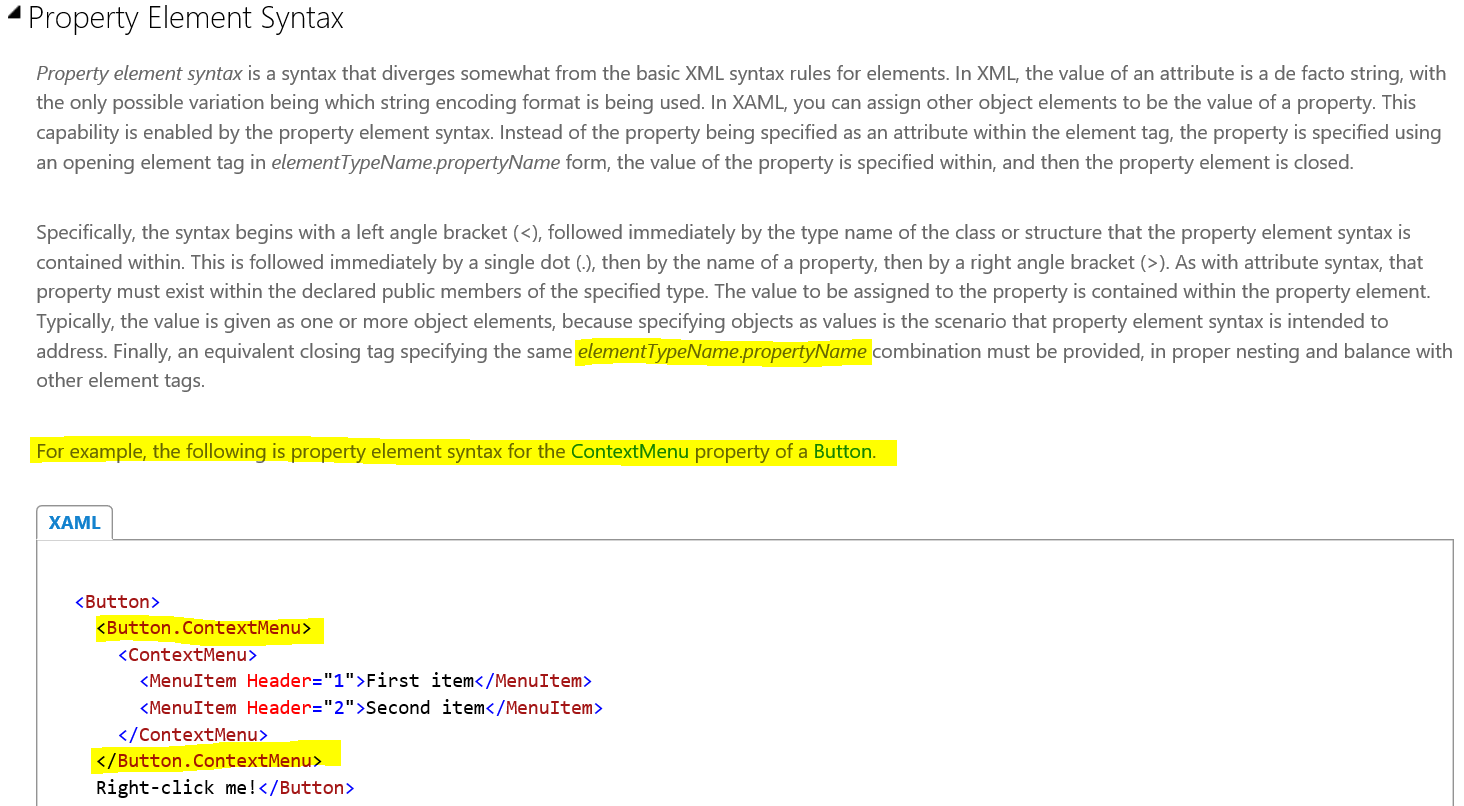


Bovenstaande combobox kent een aantal Bindings:

* De ItemsSource property is gebound aan de Landen property van de StaticResource Landrepos;
* In de ItemTemplate property wordt een DataTemplate ontwikkelt die het visualiseren van de objecten in de ItemsSource zal verzorgen. Belangrijk voor de Bindings in de DataTemplate is het feit dat de DataContext van de DataTemplate voor elk item wordt ingesteld op het item zelf;
* Het instellen van de ItemsSource property kan ook via C# code:



Opmerking: de XAML syntax, waarbij een attribuut (in dit geval: de ComboBox zijn ItemTemplate) wordt gedefinieerd door middel van een geneste tag heet [Property element syntax](https://msdn.microsoft.com/en-us/library/vstudio/ms788723(v=vs.100).aspx#object_element_syntax) (u moet wat scrollen in de webpage):



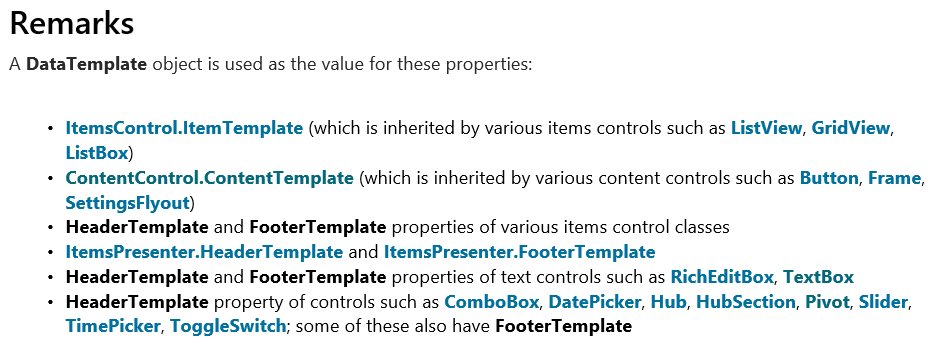
[ContentControl.ContentTemplate property](https://msdn.microsoft.com/en-us/library/windows.ui.xaml.controls.contentcontrol.contenttemplate.aspx)

Idem als voorgaande, maar dan voor de content (1 element) van de ContentControl. ContentControls zijn onder andere: Button, CheckBox, Frame(!), SettingsFlyout, .. . Het instellen van de content van de ContentControl gebeurt momenteel door middel van C# code:



Een button:

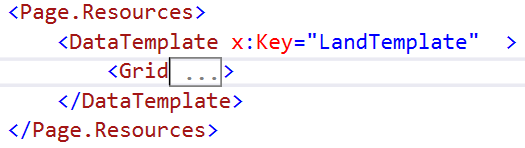
#### DataTemplates In het algemeen:



Bovenstaande screenshot komt uit [DataTemplate class](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/BR242348(v=win.10).aspx?appId=Dev14IDEF1&l=EN-US&k=k(Windows.UI.Xaml.DataTemplate)%3bk(VS.XamlEditor)%3bk(TargetFrameworkMoniker-.NETCore,Version%3dv5.0)&rd=true)

### DataTemplate resource

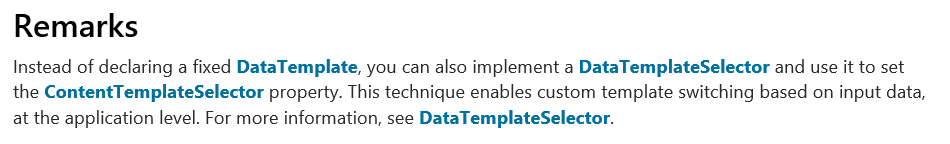
In voorgaande voorbeelden werd dezelfde templatetekst hergebruikt op verschillende plaatsen. Het is natuurlijk beter om die template als resource te definiëren en als resource in te stellen op de gewenste properties:



Indien u de DataTemplate ook op andere paginas gebruikt is het nog beter om deze op applicatie niveau te definiëren (via de merged dictionaries in App.xaml, gelijkaardig aan de style definities).

### DataTemplateSelector (niet voor deze les)

<https://msdn.microsoft.com/en-us/library/windows.ui.xaml.controls.contentcontrol.contenttemplate.aspx>



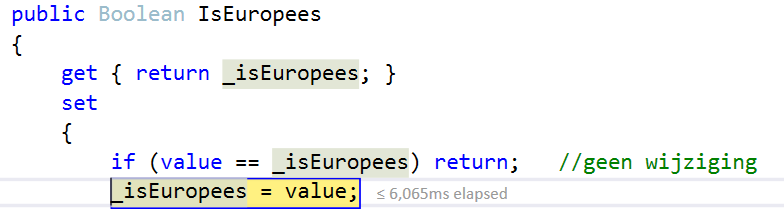
## {Binding Mode=TwoWay

De default waarde voor de Mode property van een Binding is **OneWay**: de inhoud van het model- object wordt doorgegeven aan de XAML property van de control.

Indien de gebruiker in de UI de property wijzigt (een bound Text in een Tekstvak wijzigt, of de bound IsChecked property van een CheckBox aanpast) wordt de nieuwe waarde default niet terug gestuurd naar het model object.

Indien de Binding mode als **TwoWay** wordt ingesteld worden UI wijzigingen doorgegeven naar de gevonden model property. Dit is het gemakkelijkst te testen door een breakpoint in de setter van de model property te plaatsen:

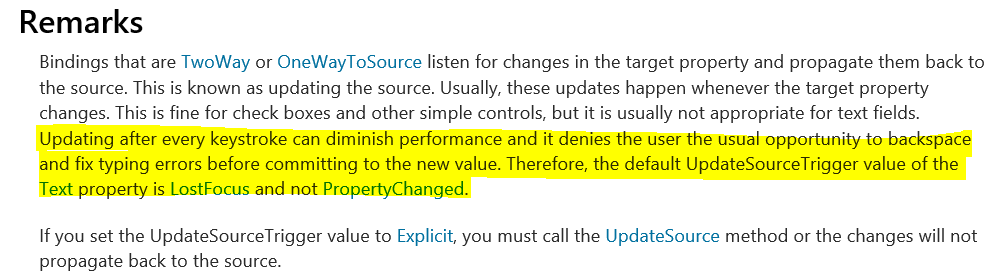




Wanneer de gebruiker op de CheckBox klikt wordt de gewijzigde IsChecked waarde doorgegeven aan het achterliggende model object.

TwoWay binding is belangrijk wanneer we de gebruiker willen toelaten om via de UI data te wijzigen in de database: de gewijzigde model informatie kan uiteindelijk naar de database geschreven worden.

Wanneer de UI de wijziging doorgeeft aan het model wordt gedefinieerd door [Binding.UpdateSourceTrigger Property](https://msdn.microsoft.com/en-us/library/system.windows.data.binding.updatesourcetrigger(v=vs.110).aspx). Voor de meeste controls is dit onmiddellijk bij wijziging van de waarde, voor de Text property van een TextBox is dit op het LostFocus event. Op deze manier kan de gebruiker zijn volledige aanpassing intikken en wordt de gewijzigde tekst enkel bij het verlaten van het tekstveld doorgegeven. Elke wijziging na elke toetsaanslag doorgeven is weinig performant:

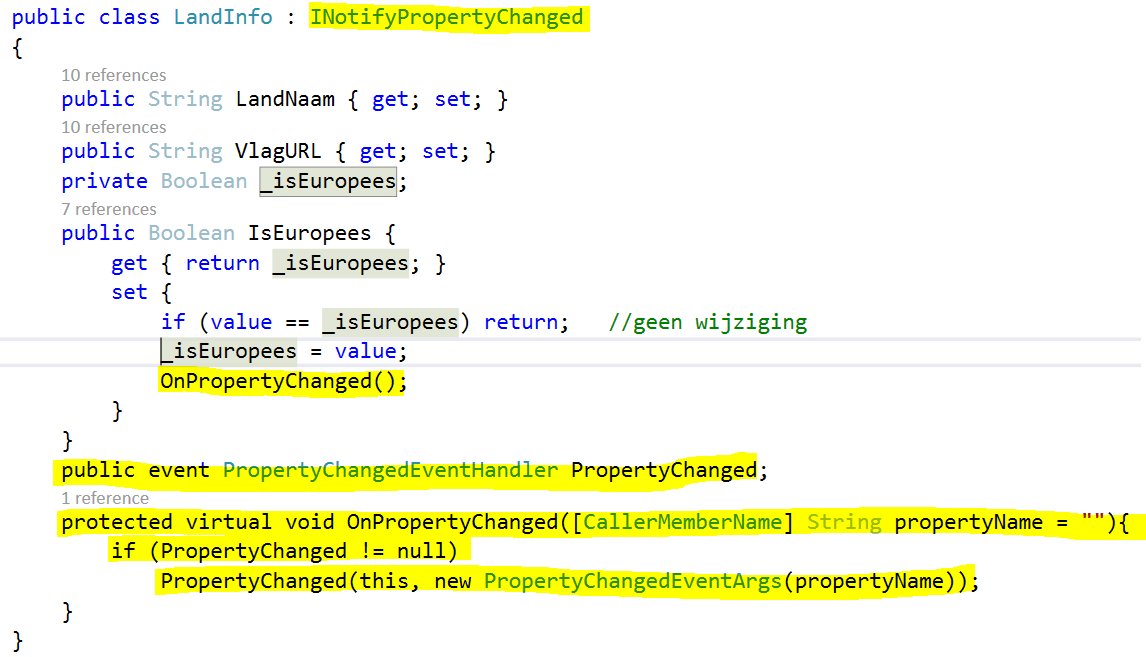


Het instellen van de UpdateSourceTrigger op een tekstveld en het testen van het resultaat wordt een onderdeel van de oefening.

## [INotifyPropertyChanged Interface](https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx)

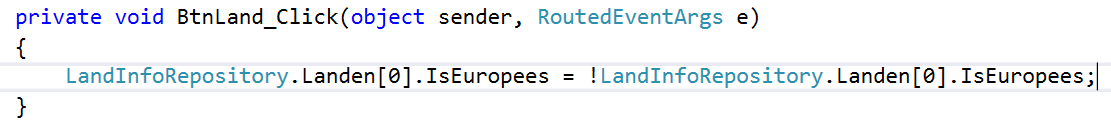
Een Binding koppelt een model property met een UI property. Indien de waarde van de model property wijzigt moet de Binding hiervan verwittigt worden om de UI aan te passen (het is niet zo dat de Binding het achterliggende model object elke seconde zal ondervragen, het model object moet de Binding via een oproep op de hoogte brengen).

Opdat een model instantie een Binding kan verwittigen omtrent het wijzigen van een propery moet de INotifyPropertyChanged interface geïmplementeerd worden:



* De interface definieert enkel het event PropertyChanged
* Indien er een propertywijziging moet doorgegeven worden zullen we dit event triggeren. Gebruik hiervoor de OnPropertyChanged methode zoals hierboven beschreven.
  + [CallerMemberName]: attribuut dat de oproepende methodenaam in de bijhorende parameternaam plaatst
  + Indien er abonnees voor dit event zijn wordt het event gestart. Indien de object in een binding wordt gebruikt is de binding een abonnee
  + De binding moet verwittigd worden welke property werd aangepast, zodat de bound property van het gerelateerde UI element kan aangepast worden.

Demo: bovenstaande implementatie werd gecombineerd met onderstaande C# code:

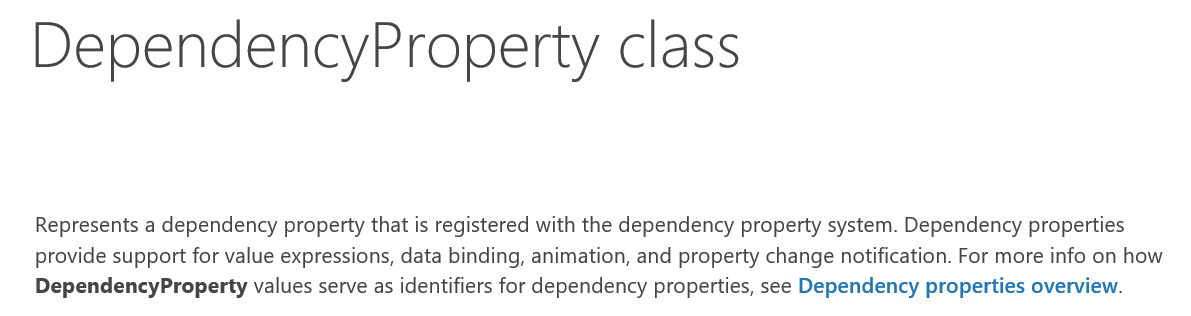


Indien het LandInfoRepository.Landen[0] object reeds werd gevisualiseerd zal de IsEuropees CheckBox gewijzigd worden!

## Dependency properties

Zonder er in deze cursus veel dieper op in te gaan (zie .NET programmer solutions voor een technische uitdieping): het .NET framework gebruikt veel Dependency Properties in plaats van de properties zoals wij deze definiëren. Een Dependency property biedt een binding de mogelijkheid om de wijziging van de depencency property waarde te detecteren, zonder dat we nood hebben aan de InotifyPropertyChanged interface (via een callback method, zie .NET programmer solutions ☺).

Naar XAML- code toe is het definiëren van een Binding op basis van een DependencyProperty of een INotifyPropertyChanged ondersteunde property identiek. In deze module zullen we, indien we zelf een property definiëren, gebruik maken van de INotifyPropertyChanged interface om de binding te verwittigen.



## {x:Bind markup extension

Met de introductie van UWP werd de markup extensie x:Bind geïntroduceerd. Deze is sterk gelijkaardig aan de {Binding extension, met volgende verschillen:

* Ze is efficienter geïmplementeerd, met compile time checks;
* Ze heeft minder mogelijkheden

Bottom line: gebruik ze waar mogelijk ☺

Meer informatie kan u vinden op [{x:Bind} markup extension](-%09https:/msdn.microsoft.com/en-us/library/windows/apps/mt204783.aspx)

## Binding via C# code (niet te kennen voor het examen)

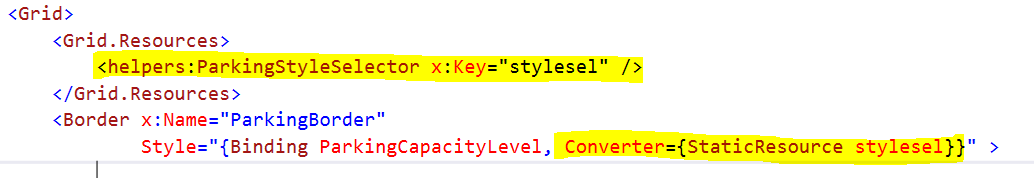
De C# die hier wordt gepresenteerd bindt de FontSize property van this (de Page) aand de Value property van de slider sldrFontSize

## Bindings en de IValueConverter interface

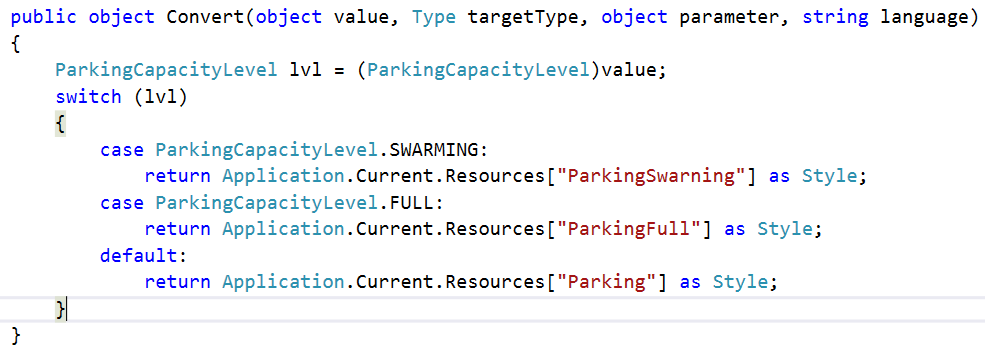
Bij het definiëren van een Binding zijn er twee redenen waarom een ValueConverter nodig kan zijn:

* Het type van de source property kan niet door de infrastructuur omgevormd worden tot het type van de target property;
* De waarde van de source property is niet de waarde die we wensen toe te kennen, maar de toe te kennen waarde kan berekend worden door middel van een converter op basis van de toegekende waarde

Een voorbeeld kan dit verduidelijken (style van de parkings op basis van bezettingsgraad):



* In deze XAML code wordt de Style property van een border gebind aan een ParkingCapacityLevel. Dit is duidelijk een niet voor de hand liggende toekenning: de target verwacht een style en we leveren een enumerated datatype;
* De converter (we ontwikkelen de klasse ParkingStyleSelector zelf) is in staat om op basis van een ParkingCapacityLevel waarde een Style te bepalen en deze als resultaatwaarde voor de Binding terug te geven.
* De converter klasses definieer ik meestal in een **helpers** folder
* Om de converter te kunnen gebruiken moeten we hem als resource definiëren



## Blend en DataTemplates

Niet van toepassing (het werkt wel, maar ik vind het momenteel geen meerwaarde hebben)

## Binding: stand van zaken na deze les

* We kunnen DataTemplates gebruiken om de usercontrols uit de OOP benadering overbodig te maken
* DataTemplates maken intensief gebruik van Bindings
* De source van een Binding kan bepaald worden door de DataContext, de Source en de ElementName
* De DataContext is een ambient property, wat op termijn zal leiden tot het MVVM pattern
* De INotifyPropertyChanged interface is nodig om wijzigingen in de model instanties door te geven aan de UI
* De IValueConverter interface is nodig om in een aantal situaties een gebinde waarde om te vormen naar een betere waarde