# MVVM: CommandBinding

Contents

[MVVM: CommandBinding 1](#_Toc433136310)

[Duiding 2](#_Toc433136311)

[PluralSight module 2](#_Toc433136312)

[ICommand interface 2](#_Toc433136313)

[ButtonBase.Command 2](#_Toc433136314)

[ICommand interface 2](#_Toc433136315)

[Commanding infrastructuur gebruiken 3](#_Toc433136316)

[NuGet package MVVMLight 3](#_Toc433136317)

[MVVMLight meerwaarde 3](#_Toc433136318)

[NuGet Package MVVMLight: installatie 3](#_Toc433136319)

[MVVMLight Commandbinding 4](#_Toc433136320)

[MVVMLight doelstelling 4](#_Toc433136321)

[RelayCommand 4](#_Toc433136322)

[Toekennen van een RelayCommand 5](#_Toc433136323)

[Binden van een RelayCommand 6](#_Toc433136324)

[ObservableCollection in MVVM 7](#_Toc433136325)

[LandInfoRepository. RemoveLandInfo(LandInfo landTeDeleten) 7](#_Toc433136326)

[ObservableCollection<T> 8](#_Toc433136327)

[LandInfoVM 8](#_Toc433136328)

[ProbleemStelling die leidt tot LandInfoVM 8](#_Toc433136329)

[LandInfoVM klasse 9](#_Toc433136330)

[IOnPropertyChangedListener 10](#_Toc433136331)

[Extra informatie 10](#_Toc433136332)

[Blend en CommandBinding 10](#_Toc433136333)

[Event binding and ICommand 10](#_Toc433136334)

[RelayCommand.CanExecuteChanged event 11](#_Toc433136335)

[RelayCommand met parameter 11](#_Toc433136336)

## Duiding

In de voorbije lessen werd MVVM DataBinding geïntroduceerd. Hierbij worden properties uit een ViewModel gebind aan properties in de UI. Door gebruik te maken van TwoWay binding en INotifyPropertyChanged worden het ViewModel en de UI gesynchroniseerd. Wat momenteel in de oplossing ontbreekt is het starten van commando’s die activiteiten buiten de UI uitvoeren. Database acties (insert, update, delete), een printcommando, het refreshen van een lijst zijn hiervan voorbeelden.

In de loop van deze tekst worden ook ObservableCollection en RelayCommand geïntroduceerd.

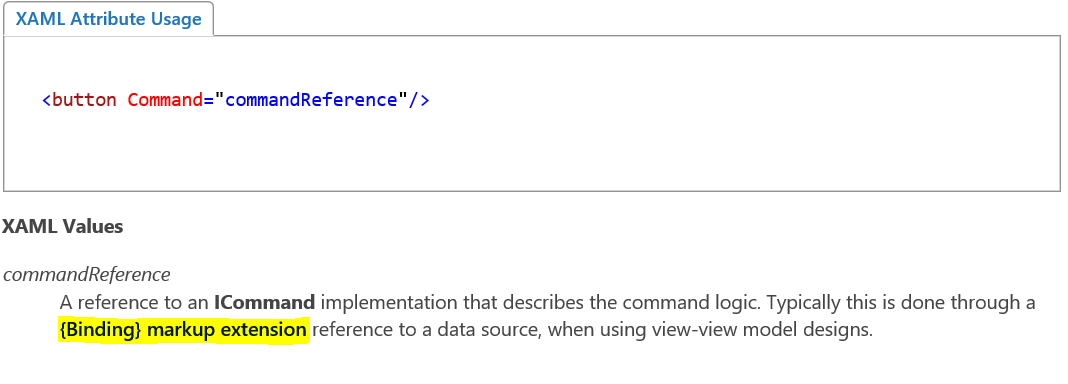
## PluralSight module

Voor wie meer wil weten omtrent MVVM (of een alternatieve invalshoek wenst door te nemen) verwijs ik graag naar (de Engelstalige video) [MVVM Light Toolkit Fundamentals](http://www.pluralsight.com/courses/mvvm-light-toolkit-fundamentals). Deze video gaat trouwens een stuk verder dan we in deze cursus wensen te bereiken.

## [ICommand interface](https://msdn.microsoft.com/en-us/library/windows/apps/system.windows.input.icommand(v=vs.110).aspx)

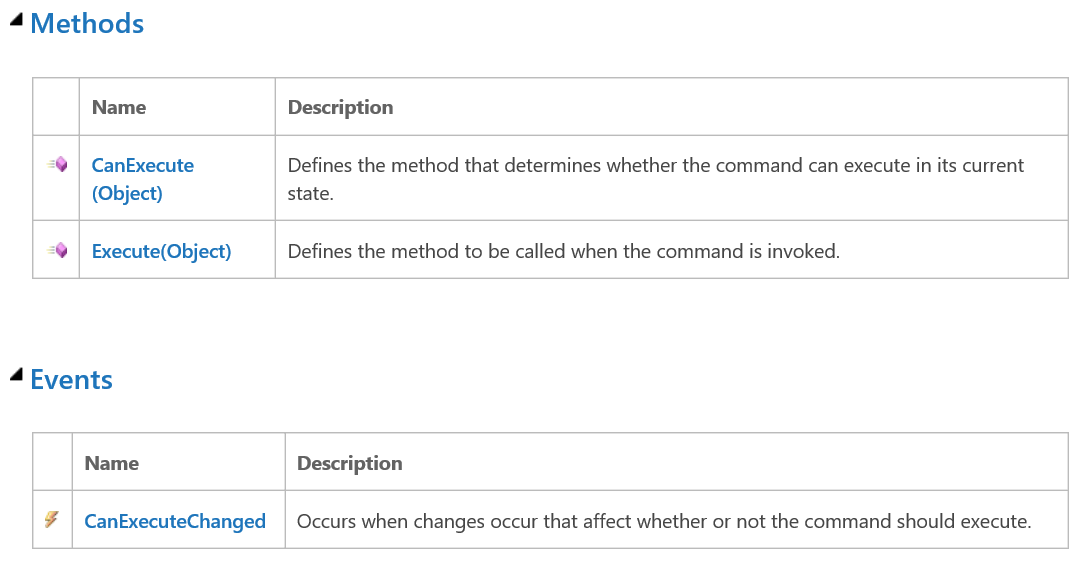
### ButtonBase.Command

.NET ondersteunt voor (een beperkt aantal) klasses de mogelijkheid om hun Command property te binden aan een ICommand object. Onderstaande screenshot komt uit [ButtonBase.Command property](https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.primitives.buttonbase.command.aspx))



### ICommand interface

De ICommand interface wordt beschreven op [ICommand Interface](https://msdn.microsoft.com/en-us/library/windows/apps/system.windows.input.icommand.aspx). Belangrijke elementen zijn:



### Commanding infrastructuur gebruiken

Om gebruik te maken van deze *Commanding* infrastructuur moeten we een aantal stappen doorlopen:

* Ontwikkel een klasse die de ICommand interface implementeert:
  + CanExecute: een functie die aangeeft of het commando uitgevoert kan worden;
  + Execute: het uit te voeren commando
  + CanExecuteChanged: een event dat aangeeft dat de mogelijkheid om het commando uit te voeren misschien gewijzigd is. In combinatie met CanExecute kan het Enabled zijn van de control die werd gebonden aan het commando aangestuurd worden.

Het uitvoeren van bovenstaande zaken voor elke commando dat we wensen te gebruiken is omslachtig en werkt ontradend ☺. Gelukkig werden bibliotheken ontwikkelt die het gebruik van de Command mogelijkheden enerzijds gemakkelijker maken en anderzijds de mogelijkheden sterk verbeteren

## NuGet package MVVMLight

### MVVMLight meerwaarde

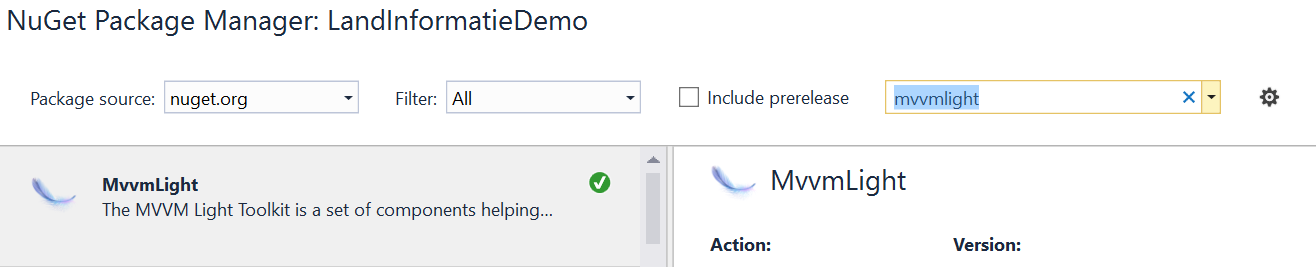
We maken gebruik van een MVVM framework zoals MVVMLight omwille van volgende redenen:

* Het opzetten van de uit te voeren commando’s wordt een stuk eenvoudiger door gebruik te maken van geprepareerde klasses zoals RelayCommand;
* In combinatie met Blend zullen we de RelayCommand’s van het ViewModel kunnen koppelen aan willekeurige events van willekeurige controls (de default .NET implementatie is beperkt tot ButtonBase classes en hun *Click* equivalenten).

### NuGet Package MVVMLight: installatie

MVVMLight is beschikbaar voor de verschillende .NET platformen en voor Xamarin (Xamarin is een .NET port voor iOS en Android: deze laat u toe om in Visual Studio (of Xamarin Studio) in C# te ontwikkelen voor iOS en Android)

Rechts klikken op uw project, kies **Manage NuGet packages ...** en installeer MVVMLight



## MVVMLight Commandbinding

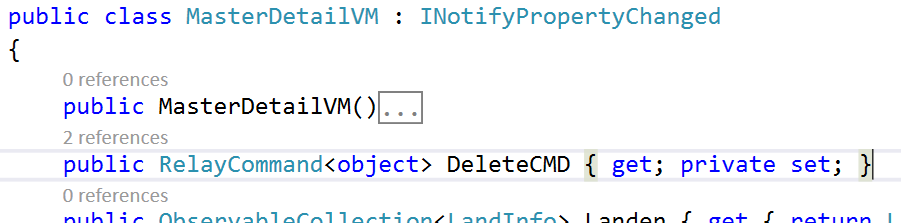
### MVVMLight doelstelling

Ten opzichte van de vorige les werd in de master/detail Page een button (Delete) toegevoegd. In een klassieke benadering zou het *Deleten* van een LandInfo object in de codebehind geprogrammeerd worden door de Click- eventhandler op te vangen en te verwerken.

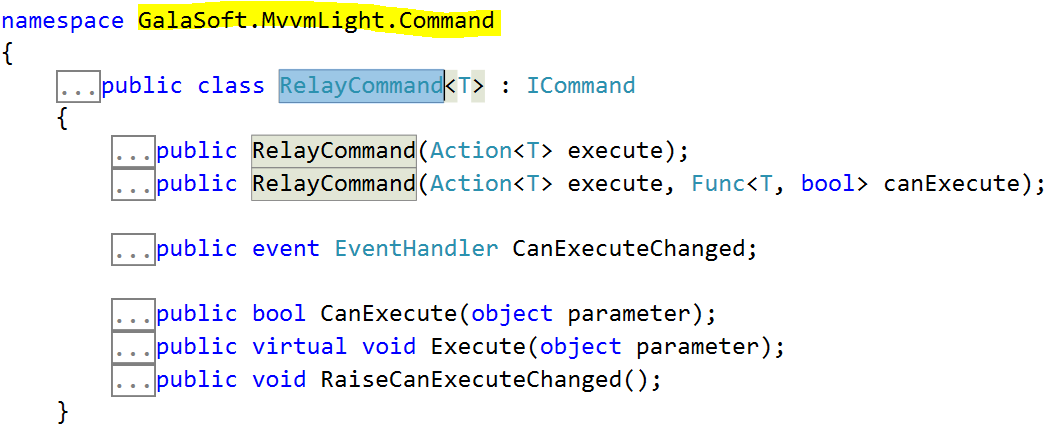
In een MVVM benadering wensen we het Click- event van de button te *Binden* aan een ICommand (RelayCommand) property van het ViewModel.

### RelayCommand

Zoals in de meerwaarde toelichting van MVVMLight aan bod kwam zullen we via MVVMLight gemakkelijk ICommand properties aan ons ViewModel kunnen toevoegen. Een voorbeeld als startpunt van de toelichting vergemakkelijkt de duiding:



* U merkt dat in het ViewModel een property DeleteCMD met return- type RelayCommand<Object> werd gedefinieerd
* Indien u F12 (= Goto Definition) duwt op de RelayCommand klasse merkt u dat deze klasse de ICommand interface implementeert:

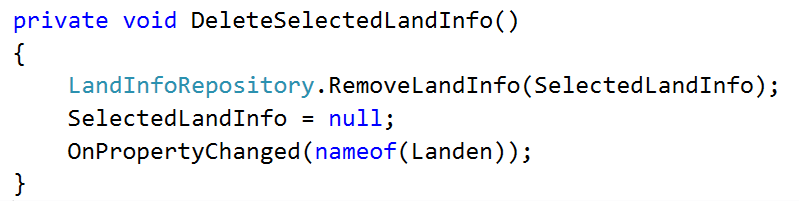


* Daarnaast is ook duidelijk dat deze klasse zich in de namespace GalaSoft.MvvmLight.Command bevindt, zodat het gepaste using- statement nodig is (te genereren via **ctrl ;**)
* U bemerkt dat RelayCommand een generische klasse is met twee constructors die Action en Func parameters hebben. Action en Func horen niet in deze cursus, ze worden meer in detail toegelicht in de cursus .NET Programmer Solutions.

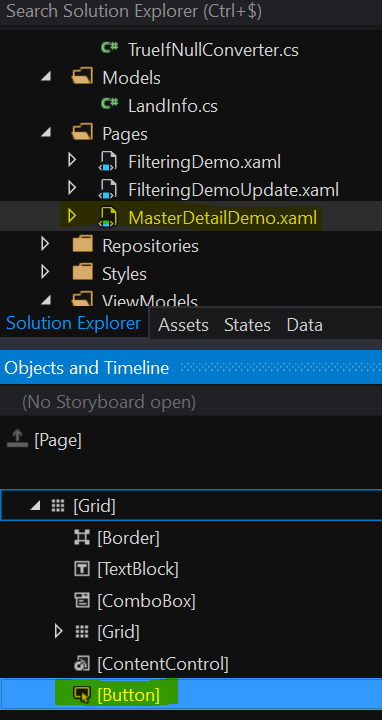
### Toekennen van een RelayCommand

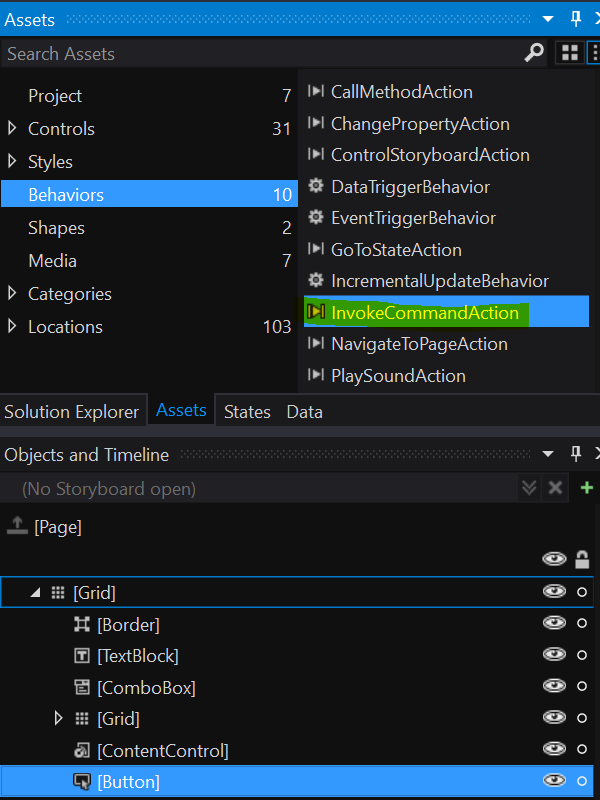
Via de constructor van de ViewModel klasse kennen we een nieuw RelayCommand toe aan de DeleteCMD property:

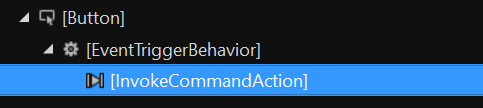


* Bemerk tussen de ronde haakjes van de RelayCommand constructor de lambda- expressie (lambda expressies worden meer in detail toegelicht in de cursus .NET Programmer Solutions ☺).
  + In de lambda expressie is o (links van =>) de naam van de parameter van type object (het parameter type wordt bepaalt door RelayCommand<Object> en hoeft niet Object te zijn). Deze **o** kan rechts van => gebruikt worden, wat hier niet het geval is)
  + Voor de eenvoud van onze oplossing raad ik aan om rechts van => een methode van het ViewModel op te roepen.
* Voorzie in het ViewModel de methode DeleteSelectedLandInfo die u in het RelayCommand aanroept. Deze methode vervult volgende taken:
  + roep de repository methode aan, en geeft als parameter de SelectedLandInfo property mee.
  + Verwittig de bindings dat Landen werd aangepast
  + Het aanpassen van de landenlijst gebeurt door de repository klasse. In een meer complexe situatie zou hier een ondersteunende database of webservice worden aangesproken.
    - Voorzie deze static methode in de repository (met een momenteel nog lege body) en plaats een breakpoint. Deze methode heeft één parameter: het te deleten landinfo object.
  + Zoek zelf de betekenis van NameOf op

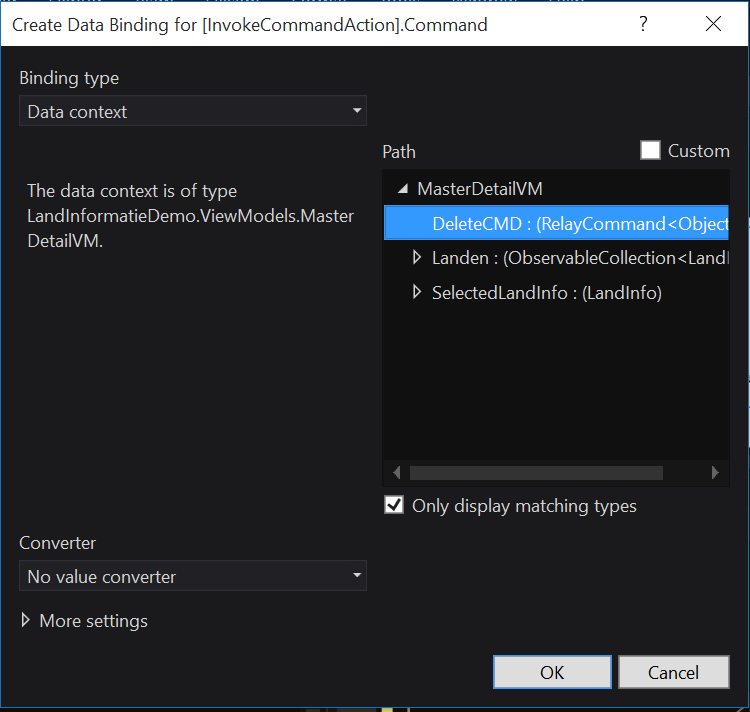
### Binden van een RelayCommand

Het binden van een RelayCommand aan een willekeurig event van een willekeurige control gebeurt met behulp van behaviors. Hoewel we de XAML code zelf kunnen schrijven is het bij de start wellicht eenvoudiger om gebruik te maken van **Blend**:

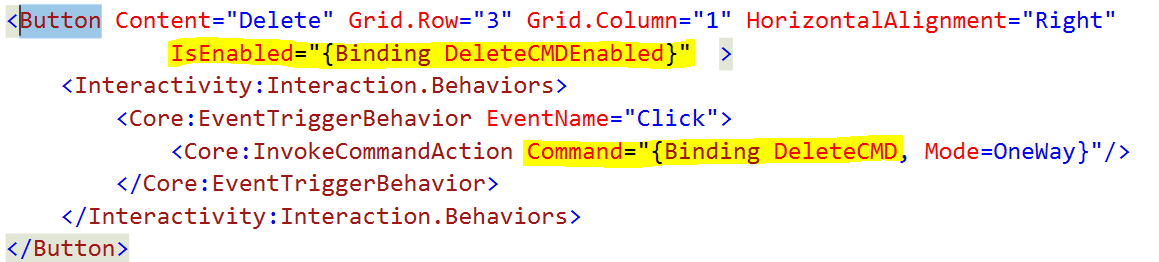
* Plaats een Delete- button op de Master/Detail page die we vorige week ontwikkelden. **Wanneer we klikken op deze knop moeten we het geselecteerde land verwijderen uit de landenlijst**
* Open het project in Blend (right click, **Design in Blend ...**)
* Open het Objects en timeline voor de Page waarop u de Button plaatste:
* Kies **Assets** en daarbinnen **Behaviors**
* Sleep een InvokeCommandAction behavior op de button
  + Hierdoor worden een EventTriggerBehavior en een InvokeCommandAction genest in de Button geplaatst
  + U kan beide apart toevoegen: sleep eerst een EventTriggerBehavior op de control, en vervolgens een InvokeCommandAction op de EventTriggerBehavior



Beide toegevoegde objecten kunnen nu ook via Blend geconfigureerd worden (via het properties window). Indien u dit een aantal keer uitvoerde kanu dit wellicht ook gedeeltelijk vanuit Visual Studio realiseren.

* Verifieer bij EventTriggerBehavior dat u ook ander events dan het Click event kan opvangen (er is een events keuzelijst in het property window (vergeet niet het juiste object aan te klikken in Objects en TimeLine)
* Bind de Command property van het InvokeCommandAction object aan de DeleteCMD property van het viewmodel. Ga hiervoor als volgt te werk:
  + Klik op het vierkantje naast Command in het property window (**Binding** tooltip) en kies **’create DataBinding’**
  + Kies de juiste property

Opmerking: niet enkel de XAML-code van de Button werd aangepast. Bovenaan de Page werden de Interactivity en Core namespaces als xmlns attribuut toegevoegd. De resulterende XAML code voor de Button ziet er als volgt uit:

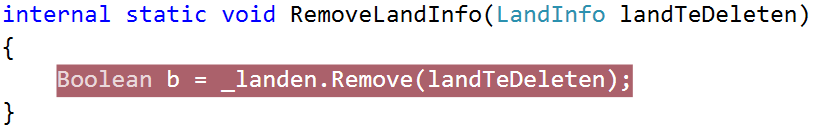


* Bemerk dat we de **IsEnabled** property binden aan een ViewModel property. Het RelayCommando ondersteunt het Enablen en Disablen van zijn gebinde control ook, maar dit zou ons voor deze les te ver leiden. Indien u dit wenst kan u de IsEnabled property van de Button ook via Blend binden, op dezelfde manier als de Command property van de InvokeCommandAction.

Wanneer u nu op de Delete knop klikt wordt de Action in DeleteCMD uitgevoerd en zal de uitvoering stoppen in de methode in de Repository klasse: CommandBinding illustrated. De Delete-knop is enkel beschikbaar indien er een LandInfo object werd gekozen.

## ObservableCollection in MVVM

### LandInfoRepository. RemoveLandInfo(LandInfo landTeDeleten)



Bovenstaande implementatie om een element uit een lijst te verwijderen ligt voor de hand. Gecombineerd met de OnPropertyChanged(“Landen”) verwachten we dat het verwijderen van het land België het gevraagde land verwijderd. Indien we dit testen blijkt echter dat telkens het laatste land uit de lijst verwijderd, wat ook het geselecteerde land was. Dit kunnen we verklaren op volgende manier:

* Hoewel we OnPropertyChanged(“Landen”) uitvoeren detecteert het systeem dat we met dezelfde Landen List<LandInfo> verzameling werken. De gegevens worden dan ook niet gerefreshed (we werken toch met dezelfde data), wel wordt de lengte van de lijst aangepast, waardoor het laatste element uit de lijst verdwijnt.

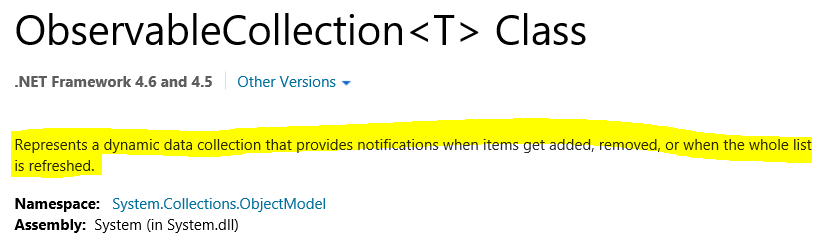
We hebben nood aan een verzameling type dat het wijzigen van zijn elementen aan een Binding kan doorgeven.

### ObservableCollection<T>

Dit generieke datatype lijkt sterk op List<T> maar implementeert ook InotifyCollectionChanged:



In de tekst [ObservableCollection<T>](https://msdn.microsoft.com/en-us/library/ms668604(v=vs.110).aspx) lezen we:



Indien een ItemsControl zijn ItemsSource bindt aan een ObservableCollection property zal de binding zijn items gepast synchroniseren met de verzameling.

Wijzig het property type van Landen en verander dit in ObservableCollection<LandInfo>. Dit vraagt ook elders in de code wat aanpassingen, de compiler helpt u met wat foutboodschappen bij het compileren. Indien u de toepassing nu terug opstart en uitvoert zal u merken dat het geselecteerde land verwijderd wordt.

Opmerking: doordat het geselecteerde LandInfo object uit de verzameling wordt verwijderd wordt het SelectedItem van de ComboBox null. Door de TwoWay binding wordt dit doorgegeven aan het ViewModel waardoor ook de detail van het zonet verwijderde land niet meer getoond wordt. We moeten in dit geval dus niet zelf een OnPropertyChanged(“SelectedLandInfo”) coderen.

## LandInfoVM

### ProbleemStelling die leidt tot LandInfoVM

In de rechtse figuur waarin een gefilterde lijst wordt getoond wensen we een Delete knop per geselecteerde item. Omdat deze knop een actie voor dit specifieke object zal starten (het is belangrijk dat u het juiste land verwijderd) zullen we een ViewModel klasse introduceren voor LandInfo objecten.

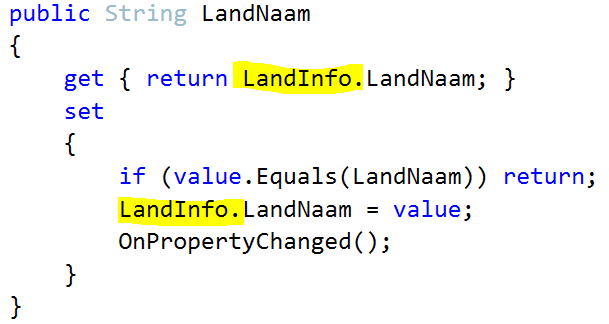
LandInfoVM toont de informatie van een LandInfo object (dit zou ook kunnen met een DataTemplate voor LandInfo) maar heeft daarnaast een aantal extra’s:

* LandInfoVM instanties bieden Delete- ondersteuning
* LandInfoVM instanties bieden de mogelijkheid om, bij het wijzigen van de IsEuropees checkbox het element onmiddellijk uit de lijst te verwijderen.

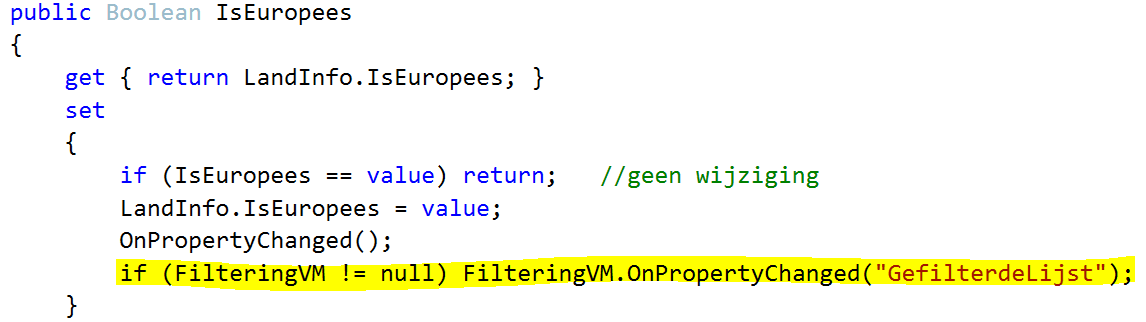
Voor beide bovenstaande features moet het LandInfoVM object kunnen communiceren met het ViewModel van de Page waarbinnen deze LandInfoVM items worden getoond. Op die manier kan een LandInfoVM object bijvoorbeeld het ViewModel van zijn Page duidelijk maken dat de gefilterde lijst moet herladen worden.

### LandInfoVM klasse

* Bemerk dat we elke relevante property van de LandInfo model klasse doorgeven via de LandInfoVM klasse
  + Dit is dom apewerk waarvoor ik jammer genoeg nog geen automatische oplossing vond
  + Voor de LandNaam property vindt u de voorbeeld code hieronder vermeld:

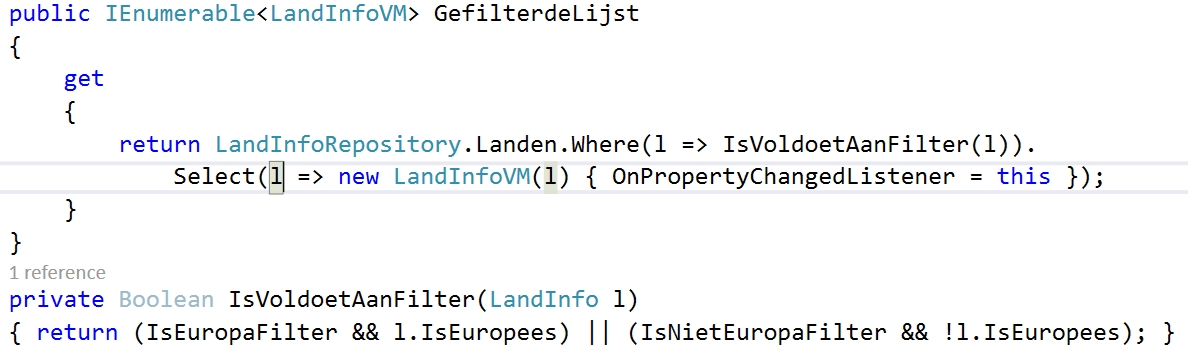


* De property FileringVM verwijst naar het ViewModel van de page waarbinnen deze LandInfoVM objecten gebruikt zullen worden
  + Indien u de toepassing volledig volgens het boekje maakt zou u een interface moeten definiëren die de mogelijkheden van een eventuele listener beschrijft. Dit Page ViewModel zou deze interface moeten implementeren en zichzelf registereren. Het voordeel van deze benadering is dat LandInfoVM dan ook in andere Pages (met andere ViewModels gebruikt kan worden)
* Maak op het ViewModel van de page de methode OnPropertyChanged public (dit zou de enige methode van de interface zijn)
* Indien het LandInfoVM object een actie registreert die een property van het Page ViewModel wijzigt is het voldoende om OnPropertyChanged(“gewijzigdepropertynaam”) op te roepen:



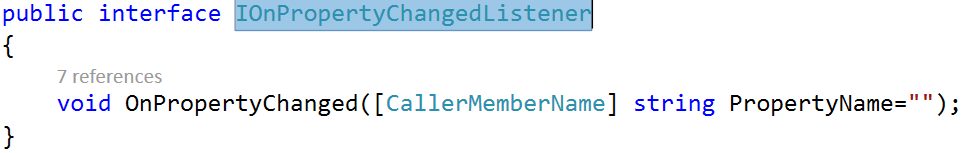
Bovenstaande code maakt dat het afvinken van de Europees checkbox het item onmiddellijk uit de lijst verwijderd. Indien we een LandInfoVM object vragen zijn LandInfo object te verwijderen roepen we dezelfde OnPropertyChanged op en wordt de GefilterdeLijst property terug aangepast.

We bekomen een lijst van LandInfoVM objecten door een lijst van LandInfo objecten om te vormen. Met LINQ kunnen we dit eenvoudig op de volgende manier formuleren:



### IOnPropertyChangedListener

Deze interface kan gedefinieerd worden om ViewModels als listeners in deel viewmodels te registereren. Deze interface heeft één interfacelement:



Omdat de ViewModels reeds INotifyPropertyChanged implementeren, en u veronderstelt wordt hiervoor de OnPropertyChanged methode te gebruiken voldoen alle ViewModels reeds aan deze interface. U moet dit wel nog opleggen via de class definitie van deze ViewModels:



U dient dan de listener te registereren als property in de sub ViewModel elementen:

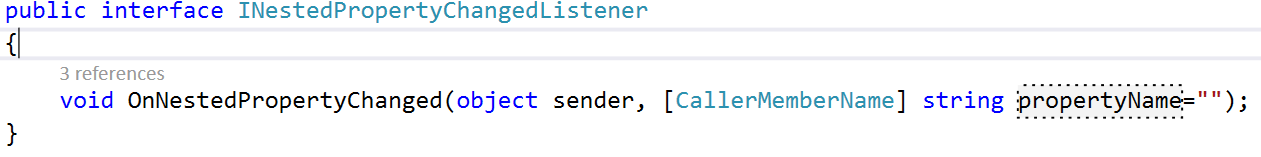


Wie vorig jaar OOP volgde herkent ongetwijfeld de Listener benadering!

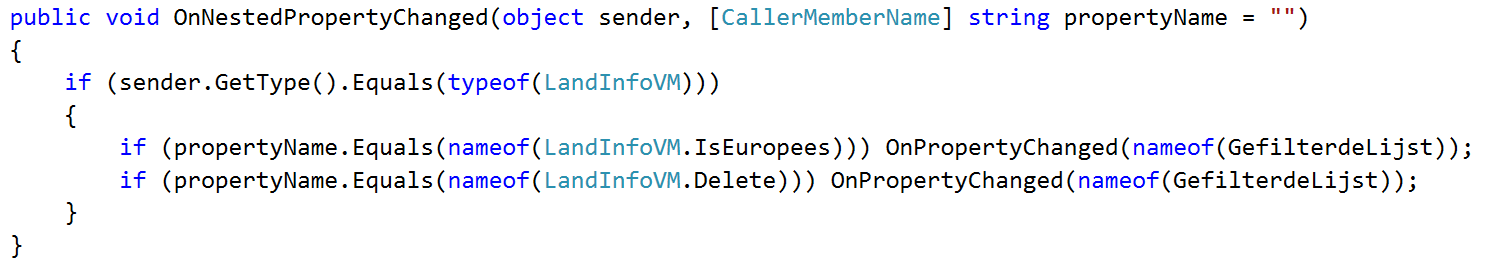
### INestedPropertyChangedListener

De IOnPropertyChangedListener wordt gebruikt om de listener duidelijk te maken welke van zijn properties gewijzigd werden, zodat de OnPropertyChanged methode kon opgeroepen worden. Dit is een minder correcte (verkeerde) uitwerking: LandInfoVM weet welke van zijn eigen properties gewijzigd zijn, maar kan niet beslissen welke properties in zijn listener hierdoor gewijzigd moeten worden: verschillende listeners kunnen trouwens verschillende properties wijzigen indien bijvoorbeeld IsEuropees veranderd voor een LandInfoVM object.

Wijzig de interface, zowel qua naam als methode zoals beschreven in onderstaande code:



De listener code (in de het Page ViewModel) kan als volgt gecodeerd worden:



In de OnNestedPropertyChanged methode ontvangen we twee parameters:

* sender: het object dat de listener contacteert
* propertyName: geeft aan welke property in sender werd gewijzigd

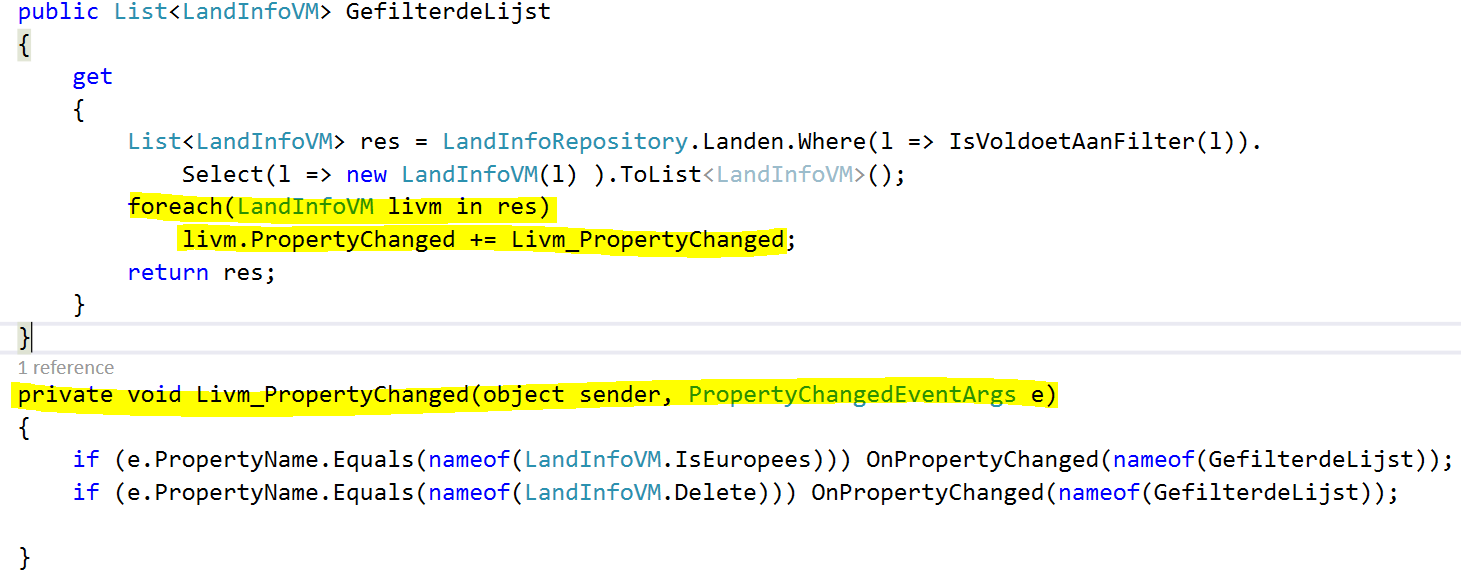
Omdat het viewmodel in principe voor meerdere model viewmodels een listener kan zijn testen we eerst op het type van de sender. Op basis van dit type en de gewijzigde property beslissen we welke eigen properties aangepast moeten worden. In bovenstaand geval beperken we ons tot het refreshen van de GefilterdeLijst binding, maar uiteraard is het ook mogelijk om flink wat meer logica (berekeningen) uit te voeren.

### Opmerking: INestedPropertyChangedListener en events (niet te kennen voor het examen)

Wie wat vertrouwd is met event handling in .NET herkent in deze listener interface wellicht een event- like oplossing. Het is inderdaad zo dat de listeners in het vak OOP werden geïntroduceerd als event vervangers, en bij de definitie van de interface routine dezelfde problemen worden opgelost als bij een event definitie:

* wie stuurt het event? (sender, van type object)
* welke informatie omtrent het event kunnen we meesturen: e van type eventargs of een afgeleide klasse.

Daarbovenop is het reeds zo dat elke ViewModel klasse die we opzetten reeds de INotifyPropertyChanged interface implementeert, om Bindings te informeren omtrent wijzigingen. Deze Interface (en zijn event) kunnen we ook gebruiken om het Page ViewModel te informeren omtrent wijzigingen in het geneste ViewModel. Dit kan met volgende code in het Page ViewModel:

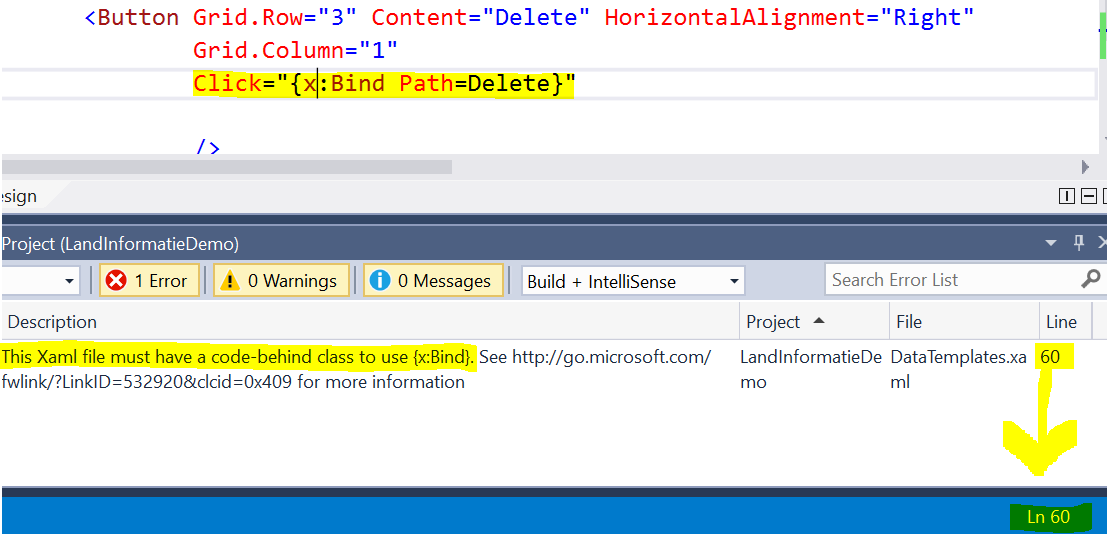


## Extra informatie

### Blend en CommandBinding

Indien we in een DataTemplate in een resourcedictionary willen CommandBinden is Blend minder bruikbaar omdat er geen DataContext gekend is. Ik zou dan ook aanraden om DataBindings eerst in de Page te definiëren en de XAML code daarna over te nemen in de DataTemplate.

### Event binding and ICommand

Windows 10 introduceerde de {x:Bind markup extension als alternatief voor {Binding. In de tekst [Event binding and ICommand](https://msdn.microsoft.com/en-us/library/windows/apps/xaml/mt210946.aspx#event_binding_and_icommand) wordt beschreven hoe {x:Bind een binding tussen een event en een ICommand kan definiëren. Jammer genoeg wordt deze binding niet gerealiseerd met de DataContext, maar wel met de CodeBehind file. Omdat DataTemplates in een resourcefile (zonder codebehind) worden geschreven biedt deze benadering geen technische oplossing voor MVVM command binding:

In een niet MVVM benadering kan deze binding syntax uiteraard wel interessant zijn.

### RelayCommand.CanExecuteChanged event

De originele ICommand interface oplossing definieert het CanExecuteChanged event. Dit wordt door de command- binding gebruikt om het Enabled zijn van de gebonden control aan te sturen. Een implementatie zoals RelayCommand (gevonden in MVVMLight) ondersteunt de methode RaiseCanExecuteChanged waardoor dit event wordt getriggered.

Opmerking: de werking van deze methode lijkt verstoord te zijn (bij mij werkt het niet ☺) bind ik het Enabled zijn van de betrokken control aan een extra property op het ViewModel. Deze benadering lijkt me trouwens ook éénduidiger te zijn.

### RelayCommand met parameter

Een RelayCommand kan met een parameter opgeroepen worden. Omdat we geen oproep syntax hebben zoals in C# met parameters wordt de parameter ingegeven via het commandparameter attribuut:

