# DataValidatie

Contents

[DataValidatie 1](#_Toc436832696)

[DataValidatie: duiding 2](#_Toc436832697)

[DataValidatie in WPF 2](#_Toc436832698)

[ErrorTemplate 2](#_Toc436832699)

[ValidationRule class 3](#_Toc436832700)

[IDataErrorInfo interface 4](#_Toc436832701)

[INotifyDataErrorInfo Interface 5](#_Toc436832702)

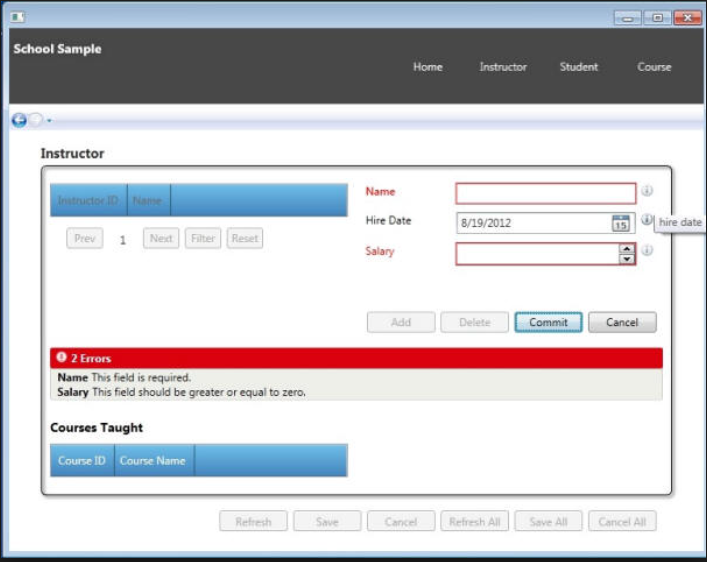
[Data annotations 5](#_Toc436832703)

[DataValidatie in UWP 7](#_Toc436832704)

[Aanbevolen literatuur 7](#_Toc436832705)

[Ter vervollediging: Indexers 7](#_Toc436832706)

[Adaptive Code 7](#_Toc436832707)



## DataValidatie: duiding

Een goede toepassing helpt de gebruiker bij het invullen van correcte data: een onmiddellijke (visuele) feedback geeft aan welke inputvelden foutief zijn, *en* waarom de ingegeven waarde foutief is. Het is behoorlijk ergerlijk indien de toepassing bij het *saven* de melding brengt dat iets verkeerd was, zonder dit meer in detail te specifiëren.

Een watermerk en een required- field aanduiding in de UI zijn ook zeer behulpzaam om de minder intuitieve velden correct te laten invullen, maar zijn op zich geen DataValidatie.

De mogelijkheden in WinRT (UWP) en WPF zijn sterk verschillend, reden waarom we ze hier elk apart behandelen. Ook zullen diverse frameworks extra mogelijkheden aanbieden, die in deze cursus echter niet uitgediept zullen worden.

## DataValidatie in WPF

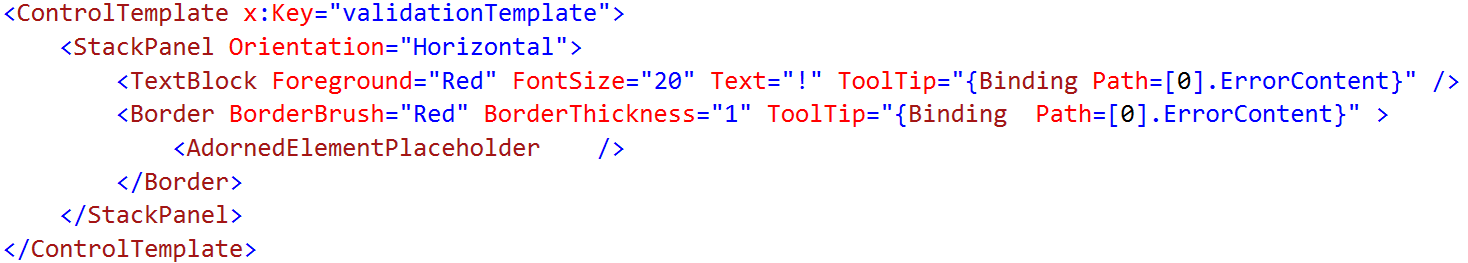
### ErrorTemplate

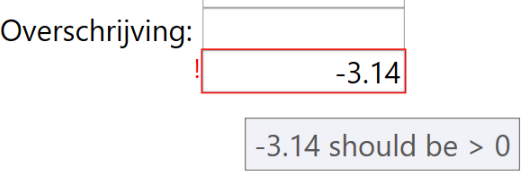
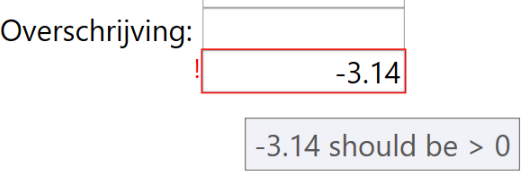
Wanneer we foutieve gebruikers input detecteren zullen we dit in de UI visualiseren. Om dit te bewerkstelligen zullen we niet de bestaande controls met foutieve informatie aanpassen, maar zullen we deze controls in een extra template opnemen. Deze extra template laag zal via kleuren en extra tekst de fout duiden. In deze context spreken we van het versieren (to adorn, to decorate) van de oorspronkelijke control:

De default ErrorTemplate plaatst een control met foutieve input in een rode box:



Een voorbeeld van dergelijke Error- template kan zijn:



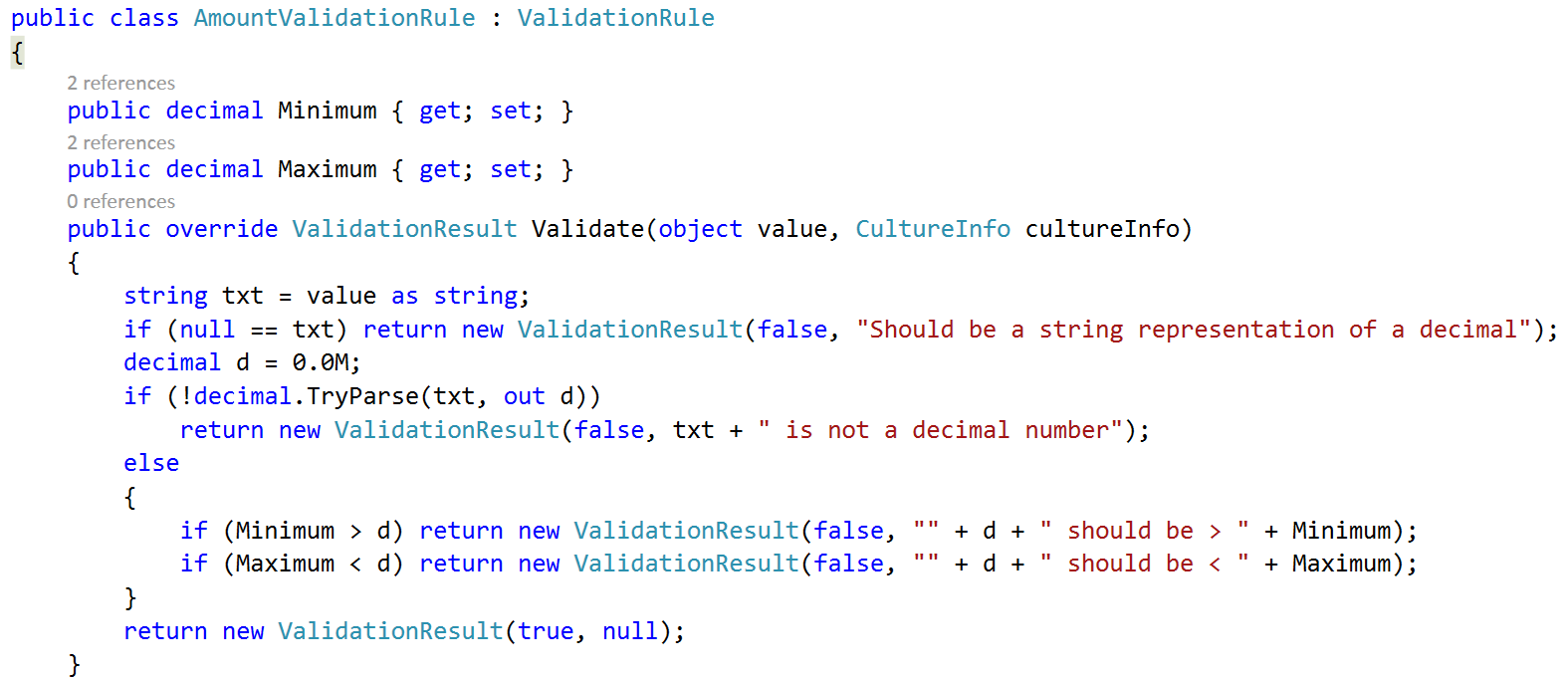
* Bemerk dat dit een ControlTemplate is (en niet een DataTemplate zoals we momenteel gewoon waren te gebruiken).
* De AdornedElementPlaceHolder zal bij gebruik van de template vervangen worden door de control waarom deze template wordt toegepast. Dit kan een TextBox, een Button of gelijk welke andere control zijn.
* De validationTemplate zal de te decoreren control in een stackpanel plaatsen zodat deze control kan worden voorafgegaan door een rode !, en waarna de control zelf in een rode border wordt geplaatst. Uiteindelijk wordt er een ToolTip tekst ingesteld op de eerste foutboodschap. Waar deze foutboodschappen vandaan komen komt verder in deze tekst aan bod.

Wat rest is de validatie-logica ontwikkelen om aan te geven dat de userinput al dan niet verkeerd is (invalid input = ongeldige input)

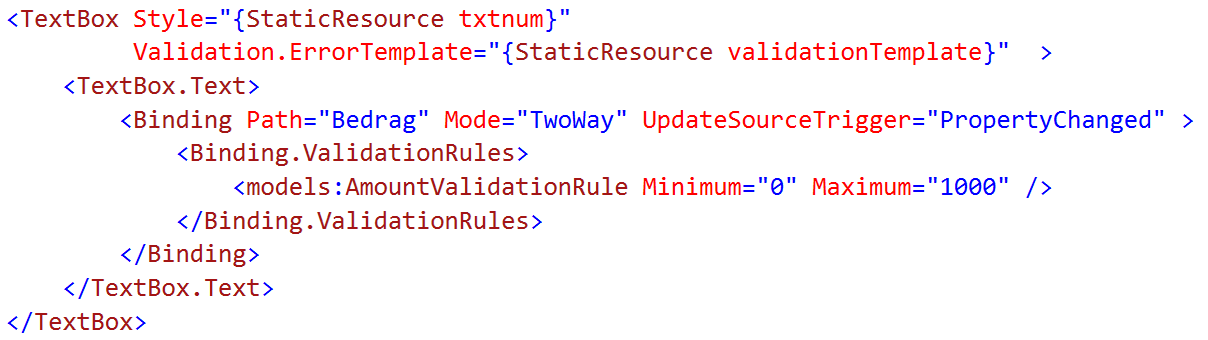
### ValidationRule class

De [ValidationRule class](https://msdn.microsoft.com/en-us/library/system.windows.controls.validationrule(v=vs.110).aspx) levert de basis klasse waarvan we erven om eigen validatie regels te coderen. De validatieregel wordt dan gekoppeld aan een binding via XAML code (die iets minder eenvoudig dan gewenst is).

Veronderstel onderstaande ValidationRule klasse AmountValidationRule:



* Bemerk dat we de *Validate* methode overschrijven. Deze levert een ValidationResult waarin het al dan niet slagen van de test met de bijhorende errorboodschap(String) wordt verzameld.
* Het is perfect mogelijk om in de ervende klasse extra properties (in dit geval Minimum en Maximum) te voorzien die informatie verzamelen, nodig om Validate correct uit te voeren. Deze properties zullen via XAML code een waarde krijgen.
* Onderstaande XAML code bindt een Text- property en voorziet een validationrule:



Bovenstaande code blinkt uit door zijn tekstlengte (niet positief) en zou ik liever niet schrijven.

Bemerk dat de TextBox via de Validation.ErrorTemplate dependency property een extra control template instelt om in geval van error de informatie te tonen.

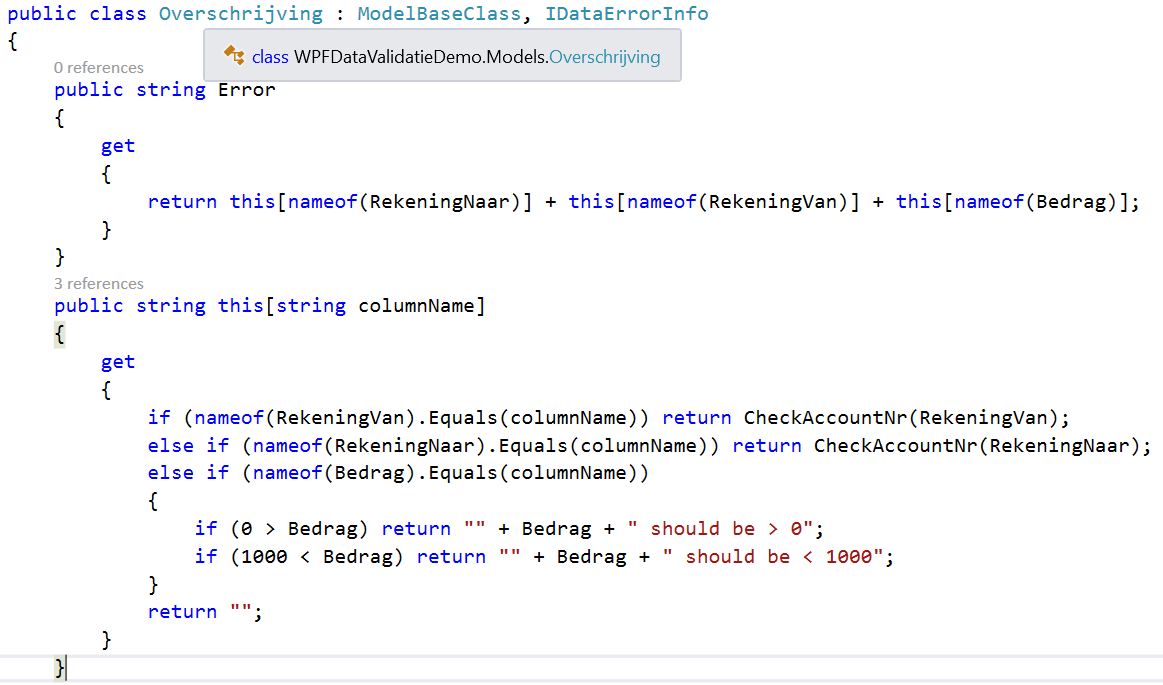
### IDataErrorInfo interface

Deze [interface](https://msdn.microsoft.com/en-us/library/743swcz7(v=vs.110).aspx) biedt een alternatief voor de verschillende validation rules die we anders moeten coderen.

Een bijkomend voordeel is dat we ze op de gebinde klasse (het model of viewmodel) kunnen definiëren. Het is logischer om de voorwaardes op het (view)model te definiëren en deze in de visualisatie te hergebruiken dan om de voorwaardes zowel in de view (ter ondersteuning van de gebruiker) als in het (view)model (ter garantie van de correcte informatie) te voorzien.

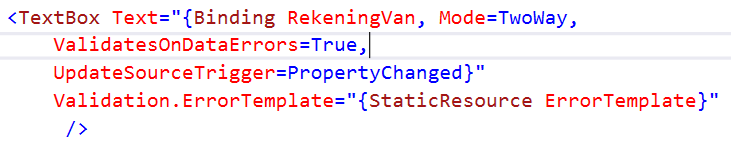
De interface definieert slechts twee properties:



* Beide properties hebben een String resultaat type: indien een fout gedetecteerd wordt kunnen we een foutboodschap teruggeven;
* Veelal zal de Error property een concatenatie van de verschillende property msg’s bevatten, hoewel dit helemaal geen vereiste is.
* De speciale array notatie Item[String] duidt op een indexer. Onderstaande code zal dit wellicht intuitief duidelijk maken, onderaan de tekst komt een beperkte toelichting van het indexer concept.

De klasse overschrijving implementeert de interface IdataErrorInfo. We kunnen deze interface implementatie niet naar de ModelBaseClass verschuiven omdat de uitwerking voor elke ervende klasse verschillend zal zijn: elke klasse heeft zijn eigen properties met hun eigen logica om het valid zijn te testen.

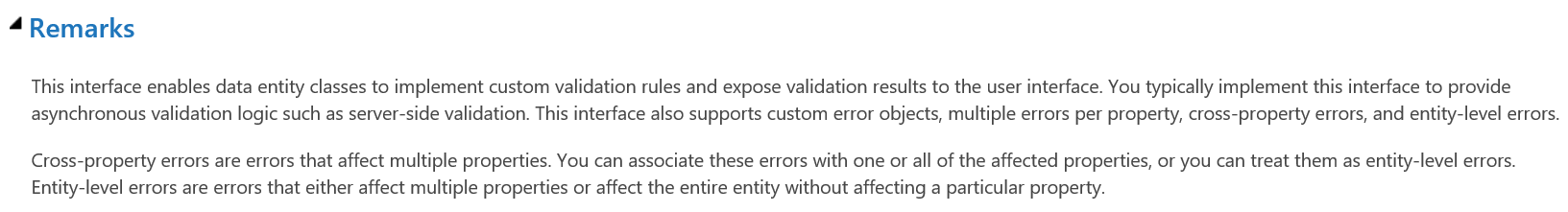
* Bemerk dat de Error property een concatenatie van de verschillende kolom errors verzamelt.
* De lege string of null geeft aan dat er geen fout is.

Wanneer het gebinde object de IDataErrorInfo interface implementeert moeten geen validation rules in de binding worden opgenomen: Telkens een binding waarde wijzigt zal de infrastructuur deze interface gebruiken om na te gaan of er een fout optrad:

* Belangrijk hierbij op te merken is dat de achterliggende machinerie slechts gestart wordt indien de ValidatesOnDataErrors property van de Binding op True wordt geplaatst.

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

The new kid on the block voor wat betreft data validatie:



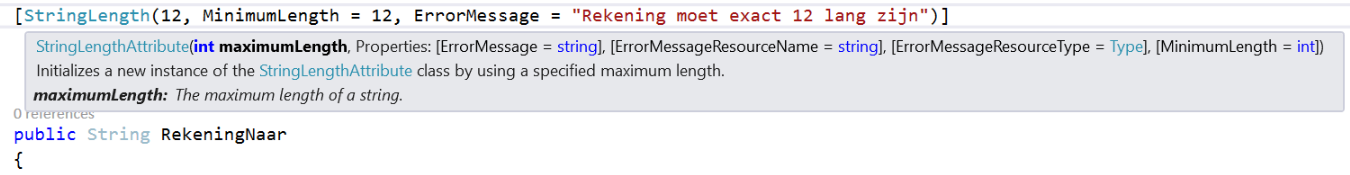
* Asynchrone validatie: indien userinput op de server (en niet lokaal) gevalideerd moet worden
* Mulitple errors per property
* Cross property errors: indien de correctheid van één property afhangt van de waarde van één of meerdere andere properties
* Entity level errors: indien de state van het object zelf een fouttoestand kan zijn, zonder dat de de properties verkeerd zijn (bijvoorbeeld: een uitgevallen netwerkverbinding)

Niet te kennen voor het labo van deze cursus ☺.

### Data annotations

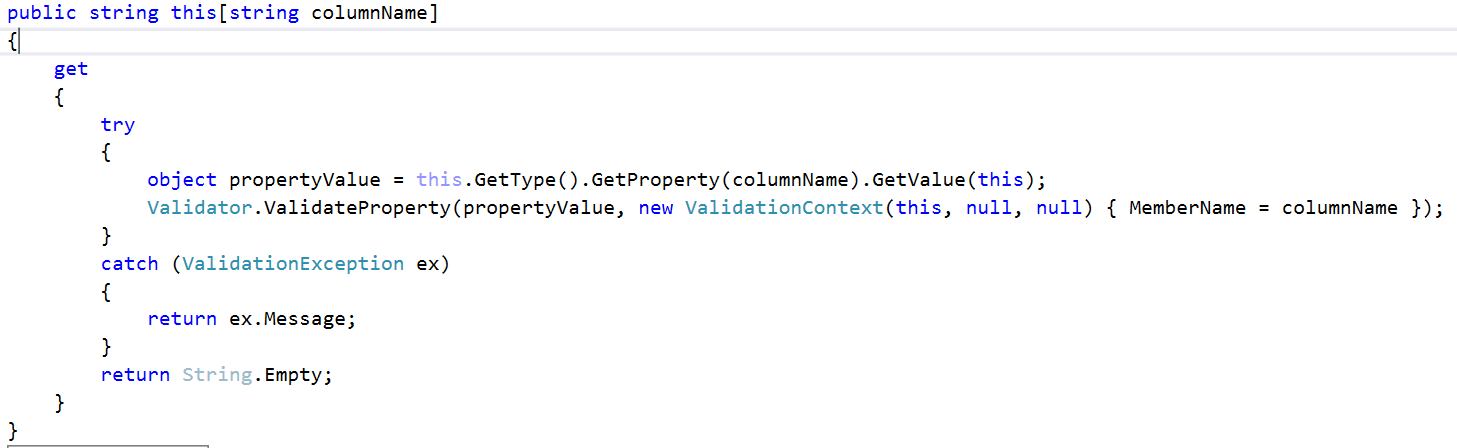
De IDataErrorInfo en INotifyDataErrorInfo interfaces zoals hierboven gedefinieerd hebben als nadeel dat het coderen van de validatie regels nogal saai, foutgevoelig en repetitief werk is.

Een declaratieve manier om voorwaardes op properties te definiëren is te verkiezen. Hiertoe zullen we gebruik maken van DataAnnotation attributen die op property niveau gedefinieerd worden:



Bovenstaande code geeft aan dat een RekeningNaar waarde exact 12 tekens lang moet zijn, en definieert de foutboodschap die getoond moet worden indien dit niet het geval is.

Deze DataAnnotations worden gebruikt tezamen met een herwerkte indexer implementatie die gebruik maakt van reflection om het valid zijn van een property te bepalen:

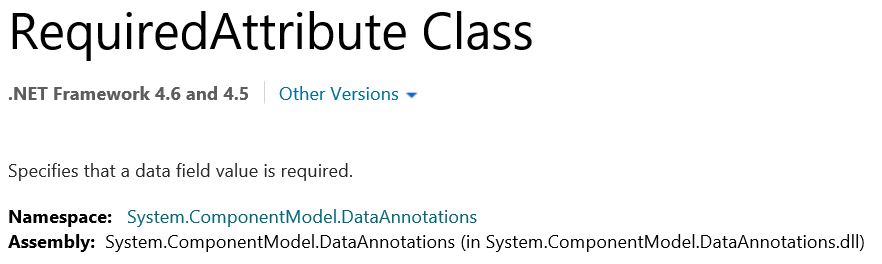


Deze code is voor elke property dezelfde:

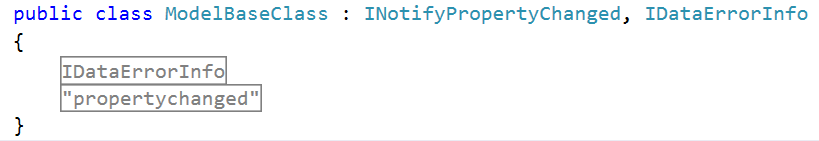
* propertyValue bevat de property waarde voor de gespecifieerde columnName (propertyName);
* de Validator.ValidateProperty oproep gaat na of deze waarde voldoet aan de opgelegde dataannotations. Indien de waarde niet voldoet wordt een exception veroorzaakt waarvan de Message de in de dataannotatie gedefinieerde foutboodschap is.

Opmerkingen:

* de DataAnnotation attributen zitten in een library die niet default gereferenced wordt. De bibliotheek die moet toegevoegd worden wordt vermeld in de MSDN uitleg van de klasses:



* Omdat deze code voor elke property gelijk is kunnen we de implementatie naar de ModelBaseClass verschuiven zodat we ze slechts éénmalig moeten uitwerken. Ze moet dan uiteraard uit elk van de ervende modelklasses verwijderd worden.



* Bemerk dat deze data annotaties dezelfde zijn als deze die in ASP.NET gebruikt worden.
* Voor een volledige opsomming van bruikbare dataannotations verwijs ik graag naar de msdn documentatie: [System.ComponentModel.DataAnnotations Namespace](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations(v=vs.110).aspx)

De beperkte set DataAnnotaties alleen is niet voldoende. Indien we validaties willen uitvoeren die niet met een DataAnnotatie kunnen beschreven worden zullen we moeten terugvallen op de eerdere mogelijkheden.

## DataValidatie in UWP

UWP projecten hebben weinig tot geen data validatie ondersteuning:

* De controls hebben geen Validation.ErrorTemplate dependency property. Dit maakt het om te starten al moeilijk om de visualisatie aan te passen ingeval van invalid input;
* Er is geen validation infrastructuur in de Binding klasses
* De DataAnnotations zijn wel aanwezig: het is mogelijk om requirements op property niveau te definiëren, maar er ontbreekt infrastructuur om er achteraf iets mee te doen.

Er zijn een aantal pogingen om dit tekort op te vullen gedaan:

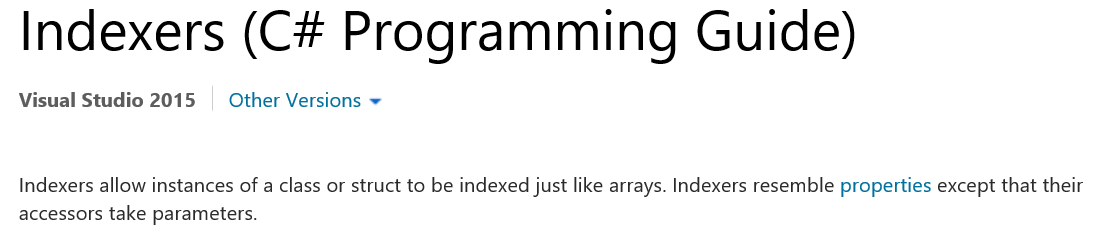
* Prism, zie [User input validation with Prism and data annotations on the UWP](http://blogs.u2u.be/diederik/post/2015/09/27/User-input-validation-with-Prism-and-data-annotations-on-the-UWP.aspx)
* [Let’s Code! Handling validation in your Windows Store app (WinRT-XAML).](http://blog.jerrynixon.com/2014/07/lets-code-handling-validation-in-your.html) is een artikel met een zeer interessante insteek (onder andere undo gedrag wordt geautomatiseerd) maar de techniciteit ervan overstijgt het beoogde niveau van deze cursus.

## Aanbevolen literatuur

* [Input Validation in Windows 8 Store apps](http://blogs.u2u.be/diederik/post/2012/10/09/Input-Validation-in-Windows-8-Store-apps.aspx)

## Ter vervollediging: Indexers

De code die IDataErrorInfo interface illustreert maakt gebruik van indexers. Zoals in de MSDN help [Indexers (C# Programming Guide)](https://msdn.microsoft.com/en-us/library/6x16t2tx.aspx) beschreven wordt:



* Het verschil met array indexing bestaat erin dat de gebruikte index (parameter) niet noodzakelijk een int- waarde moet definiëren.
  + In het geval van de IdataErrorInfo definieert de indexer een String parameter!
* Wanneer we een indexer definiëren gebruiken we het keyword *this*. Ditzelfde keywordt wordt trouwens ook gebruikt bij de definitie van extension methods of bij het duiden van het uitvoerende object.

## Adaptive Code (niet te kennen voor het examen)

Vorige week ontwikkelden we een adaptive UI voor de wielerzesdaagse: afhankelijk van de device grootte en het al dan niet beschikbaar zijn van het network werd de UI aangepast.

De video [adaptive code](https://channel9.msdn.com/Events/Windows/Developers-Guide-to-Windows-10-RTM/Adaptive-Code) is een goede intro.

* Platform extension SDK’s