# UWP Resources

## UWP Resources: duiding

Bij het programmeren in C# worden variabelen gebruikt om waardes in op te slaan zodat deze later kunnen hergebruikt worden. In XAML wordt iets gelijkaardigs gerealiseerd met behulp van Resources. Resources zijn objecten, aangemaakt in XAML en aanspreekbaar op basis van hun key. Resources worden veel gebruikt in Bindingsituaties die pas volgende les aan bod komen.



Screenshot uit tekst: [ResourceDictionary and XAML resource references](https://msdn.microsoft.com/en-us/library/windows/apps/mt187273.aspx)

## UWP Resources: inleiding

Lees de tekst [ResourceDictionary and XAML resource references](https://msdn.microsoft.com/en-us/library/windows/apps/mt187273.aspx). Wat u uit deze tekst zeker moet oppikken zijn volgende elementen (in volgorde van de tekst):

1. Resources worden gedefinieerd in een ResourceDictionary. Die kan ofwel in een .Resources tag of in een aparte file gedefinieerd worden
   1. FrameworkElement definieert de Resources property. Alle ervende klasses (onder andere alle controls) erven dus deze property.
   2. Ook niet FrameworkElement objecten kunnen resources **gebruiken**, ze kunnen ze wel niet definiëren
2. In XAML gedefinieerde resources zijn opvraagbaar in C# code via de .Resources property van het element dat de resource definieert:  
     
   Screenshot uit tekst: [ResourceDictionary and XAML resource references](https://msdn.microsoft.com/en-us/library/windows/apps/mt187273.aspx)
3. Merge een dictionary- file (zelf te maken) met een page dictionary. Test uw oplossing!
   1. Om te testen gebruikt u een resource die gedefinieerd werd in de andere file
4. Lees de topic **Lookup behavior for XAML resource references** grondig! door. Herlees hem en probeer test- scenario’s uit te denken (op papier?) waarmee u kan nagaan of u de Lookup behavior begrijpt. Samengevat, en voldoende voor deze cursus komt de lookup hierop neer:
   1. Lookup behavior wordt enkel toegepast voor een resource dat op basis van zijn key wordt gezocht
   2. De gezochte resource wordt eerst gezocht in de resources van het zoekende element.
   3. Bij het zoeken in de resources van een element gaan we als volgt te werk:
      1. Eerst worden de keys van de niet merged dictionary resources gechecked (de keys die niet in een andere file werden gedefinieerd). Indien de key wordt gevonden stopt het opzoeken.
      2. Indien hier niet gevonden worden de keys in de merged dictionary files doorzocht. Indien er meerdere files gemerged werden worden deze files in omgekeerde volgorde van toevoegen aan de merged dictionary doorzocht. Indien de key in één van de files wordt gevonden stopt het opzoeken.
         1. Een key is uniek in zijn resourcefile, maar kan meerdere keren voorkomen in verschillende resourcefiles die worden gemerged.
   4. Indien de key niet wordt gevonden in dit element wordt het zoeken hernomen in de resources van het parent element
   5. Indien de resource key niet werd gevonden in de resources van het element of één van zijn parent elementen wordt de resource gezocht in de Application.Resources dictionary (gedefieerd in App.xaml)
   6. Forward referencing wordt niet ondersteund: u kan enkel resources opzoeken die tekstureel reeds eerder werden gedefinieerd.

## UWP Resources: praktisch gebruik

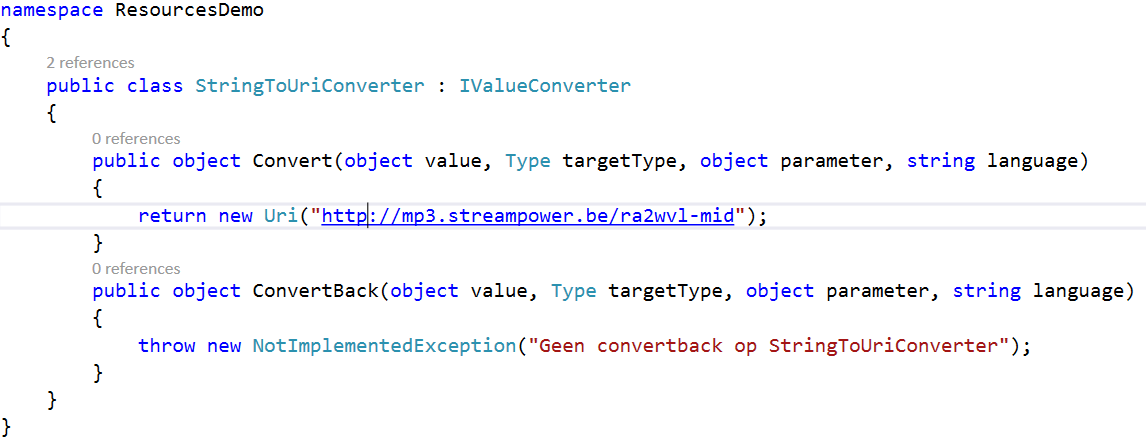
XAML resources worden aangemaakt om een aantal cases te ondersteunen:

* Algemene waardes die doorheen de ganse applicatie gebruikt kunnen worden
  + Styles
  + Templates
  + ViewModel, Locator, ...
  + Instanties van primitieve(re) klasses die (vooral) in styles en templates gebruikt kunnen worden
* App.xaml is de aangewezen plaats om resources, nodig in de ganse toepassing, te definiëren. Om dit te structureren zal u in de App.xaml wellicht een merged dictionary aanmaken die verschillende dictionary files zal bundelen.
* Styles en Templates worden gemakkelijk aangemaakt via Blend en of Visual Studio (right click de te stylen of templaten control en kies het juiste popup menu). Indien u wat ervaring hebt in het XAML code schrijven kan dit eventueel zonder de design tools te gebruiken.

## XAML resource start voorbeelden



* De resource **mijnblauw** is een SolidColorBrush met als toegekende kleur RoyalBlue. Deze kan gebruikt worden overal waar een Brush wordt ingesteld, in het voorbeeld de Background property van de Grid
* Op dezelfde manier werd een fontsize gedefinieerd die aan een fontsize property gekoppeld kan worden (niet gebeurd in dit voorbeeld omdat dit beter via styles kan ingesteld worden)
* De resource **logobm** is een BitmapImage die als source van een Image control kan ingesteld worden. De UriSource van deze BitmapImage wordt ingesteld met de static resource **koeimg**.
  + Er gebeurt een automatische conversie van String (het type van de resource **koeimg**) naar Uri (het type van de property UriSource). Welke conversies automatisch gebeuren is in UWP een niet gedocumenteerd gegeven ☺
* De static resource **bella** is een instantie van de klasse Beest die we zelf ontwikkelden (vandaar de prefix local, zie verder omtrent namespaces)  
  
* De contentcontrol’s content wordt ingesteld op de static resource **bella**. Indien geen contenttemplate wordt opgegeven wordt het ToString resultaat van de content getoond.
* Om de naam en de foto van bella te tonen wordt gebruik gemaakt van een datatemplate. Deze is op zijn beurt een static resource (die door middel van binding de UI van een Beest definieert).
* De source van de webview wordt via een binding ingesteld op de static resource **radio2**. Deze is een **String**, terwijl de source een **Uri** verwacht. Omdat static resources niet impliciet worden omgevormd in een binding (wat wel het geval was met het **String**- attribuut in **logobm**) hebben we nood aan een TypeConverter object. Hiertoe gebruiken we een instantie van de (eigen) klasse **StringToUri** die de vereiste conversie kan uitvoeren.

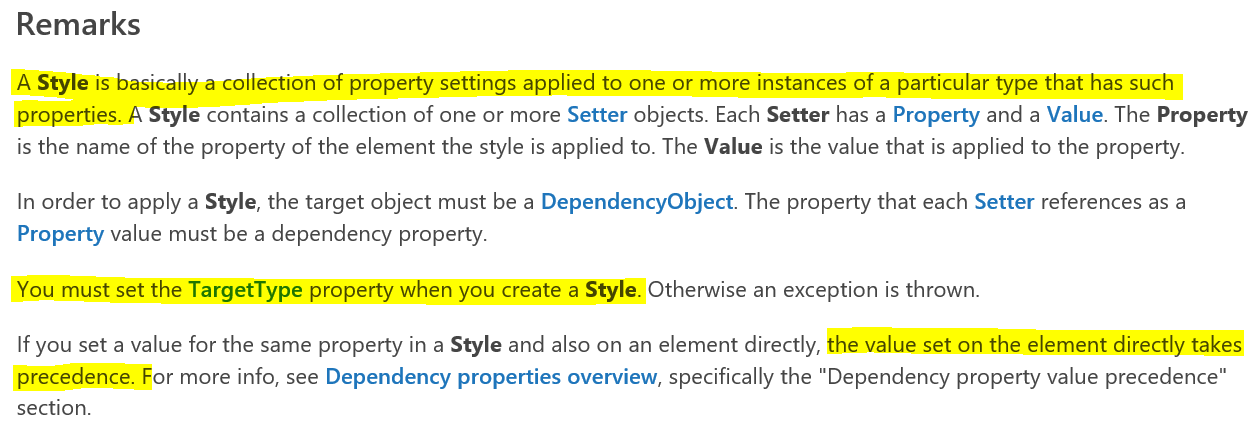
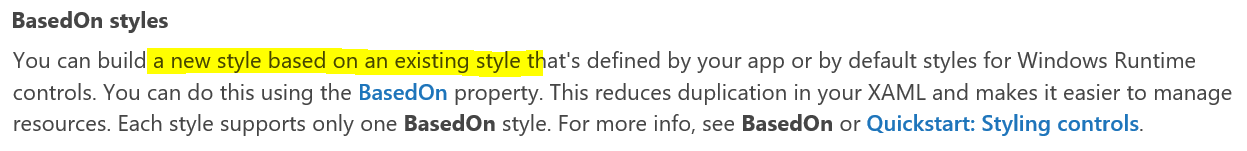
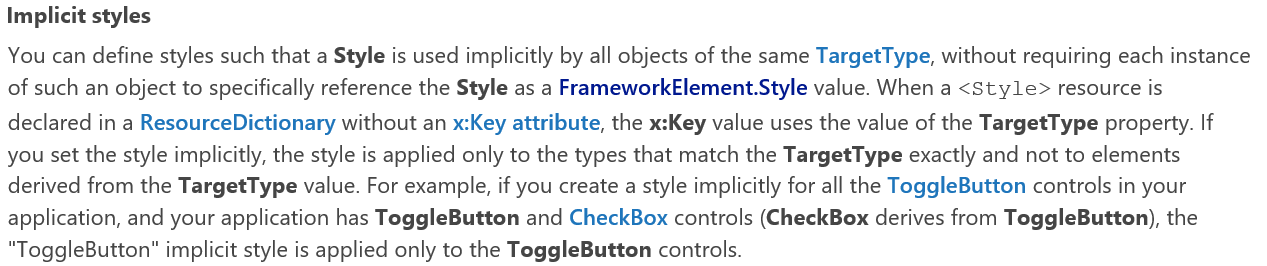


## XAML Style resources

Net zoals in HTML zijn styles de manier bij uitstek om doorheen een ganse applicatie visuele elementen op een gelijkaardige manier op te maken. Als voorbeeld zullen we via Blend de Style van een image aanpassen:

* Right click op een image (in de **designer** of de **objects en timeline**), kies ‘edit style’ en kies de gepaste optie (current, copy, empty)
  + Dit maakt een static resource (Style) aan die aan de image wordt gekoppeld
* Wijzig de properties zodat die de gewenste stijlkenmerken aannemen. De nodige XAML- tags worden aan de static resource toegevoegd.

De tekst [Style class](https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.style.aspx) beschrijft in detail de mogelijkheden en het gebruik van Styles. Lees deze tekst en leer onder andere volgende elementen:

* 
* 
* 

## XAML theme resources

Theme resources worden in detail beschreven in [XAML theme resources](https://msdn.microsoft.com/en-us/library/windows/apps/mt187274.aspx).

Wat je vooral moet onthouden:

* Er zijn drie themes: **Light**, **Dark** en **High Contrast**
* Indien de gebruiker zijn theme instelling wijzigt worden de theme waardes binnen de toepassing terug gelezen en toegepast. Dit is een belangrijk verschil met gewone Static Resources.

## Dynamic resources

In UWP toepassingen gebruiken we bijna altijd Static Resources. Deze worden éénmalig geladen en wijzigen niet voor de duur van de toepassing.

Dynamic resources daarentegen (vooral in WPF toepassingen) worden bij elk gebruik opnieuw opgehaald. Indien het resource object in de dictionary wordt vervangen zal het nieuwe object gebruikt worden (zie WPF demo)

## UWP Resources: nMCT invalshoek

### Resources: instanties van klasses

De tag, gebruikt om een resource aan te maken, geeft aan van welke klasse de resource een instantie zal zijn. De tagnaam bestaat uit een namespace alias en een klasse:

<x:String

* x: de namespace alias die refereert naar de namespace waarbinnen de klasse String gezocht zal worden. De intellisense helpt u door na de : enkel die klasses op te lijsten die in die namespace zitten en in aanmerking komen om Resources te maken
  + er moet onder andere een default constructor zijn (waarom?)
* String is de klasse binnen deze namespace (een klassenaam is uniek binnen een namespace).

### Namespaces

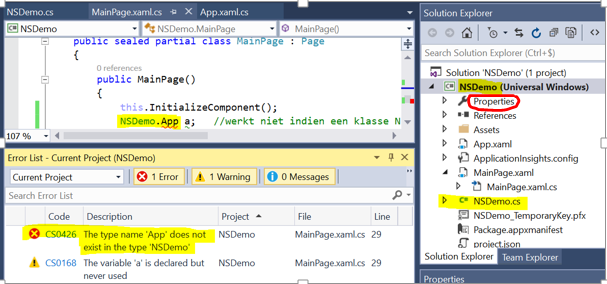
#### Namespaces in C#

Lees [Using Namespaces (C# Programming Guide)](https://msdn.microsoft.com/en-us/library/dfb3cx8s.aspx) zodat u weet wat namespace zijn. Na het lezen van deze tekst zou u moeten begrijpen dat de klasse **Button** niet bestaat maar dit een verwijzing is naar de klasse **Windows.UI.Xaml.Controls.Button**.

* De fully qualified name identificeert de klasse op unieke wijze
* Een klassenaam bestaat uit twee stukken
* Namespaces kunnen genest zijn door gebruik te maken van een ‘.’ tussen verschillende stukken van de namespace naam.

**Reflectie**: indien u na het lezen van deze tekst niet weet wat een namespace alias is zou u hem best herlezen.

**Reflectie**: op basis van deze tekst en het gegeven dat de default namespace van een project de projectnaam is begrijpt u dat het **geen** goed idee is om in een project een klasse te coderen met dezelfde naam als de projectnaam! (u kan de default namespace terugvinden in de properties van het project)



#### Namespaces in XAML

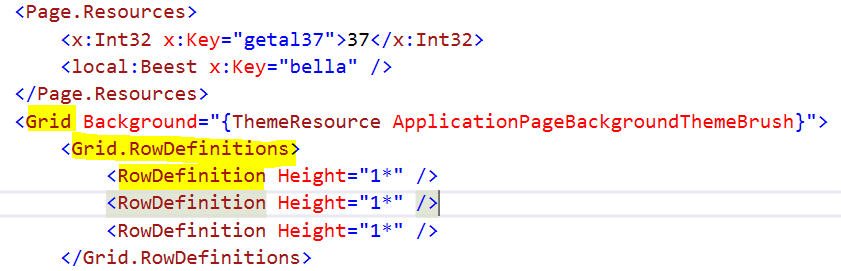
Ook in XAML wordt gebruik gemaakt van namespaces:

* Eén namespace is de default namespace
* Alle andere namespaces definiëren namespace aliases



In bovenstaande XAML code worden vijf namespaces gedefinieerd. De eerste heeft geen alias en is de default, de tweede heeft als alias x.

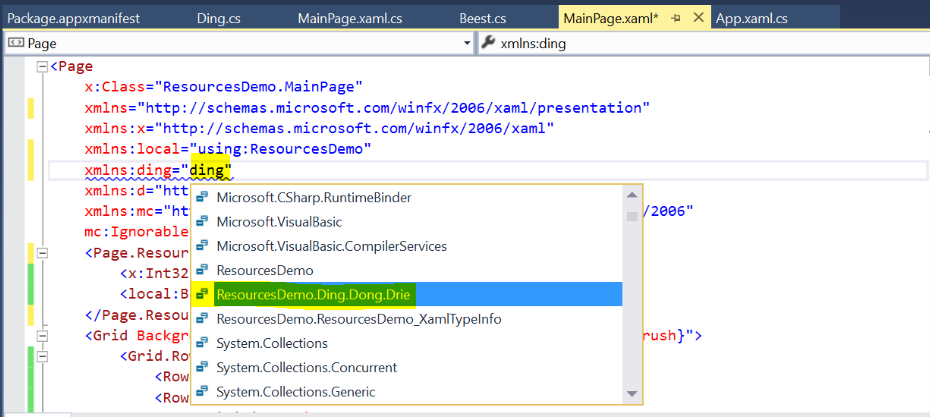
* Indien u in XAML-code een tag niet prefixed met een alias wordt de default namespace gebruikt. In onderstaande code zijn de resource tags geprefixed met respectievelijk de namespace aliasses x en local, de Grid tag is niet geprefixed en de Grid klasse wordt dan ook gezocht in de default namespace.



**Reflectie**: een interessant artikel dat de oorsprong van xmlns=<http://schemas.microsoft.com/winfx/2006/xaml/presentation> toelicht is te vinden op [XmlnsDefinition for a Cool Namespace Mapping](http://blog.pixelingene.com/2006/10/xmlnsdefinition-for-a-cool-namespace-mapping/)

**Reflectie**: Visual Studio voegt een namespace alias toe met de alias local die verwijst naar de default namespace van het project waarin u nu aan het werken bent.

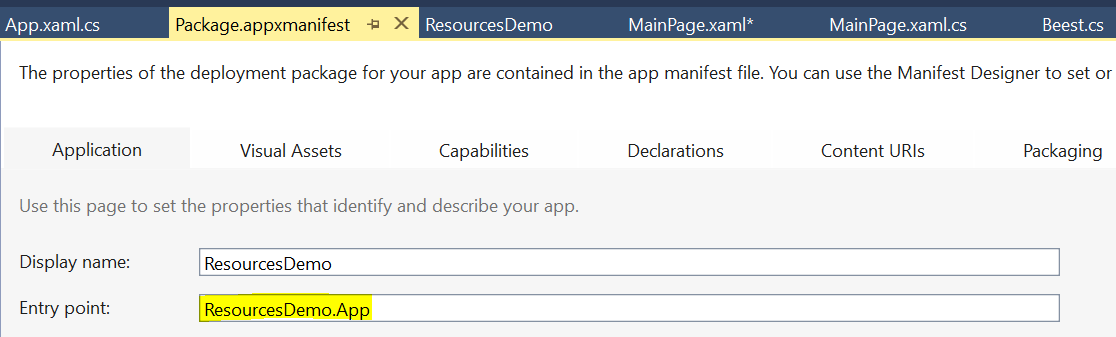
**Hint**: indien u zelf een namespace alias toevoegt raadt ik u sterk aan de intellisense te gebruiken!



Bemerk dat u zich niet hoeft te beperken tot een prefix bij het zoeken van de namespace (in het voorbeeld wordt gepositioneerd op basis van de tekst ‘ding’).

## App.xaml

De klasse App is default het startpunt van uw toepassing, zoals wordt aangegeven in het manifest van de app:



De klasse wordt beschreven door middel van twee files:

* App.xaml
  + File waarin onder andere de App- resources worden beschreven
* App.xaml.cs
  + File waarin onder andere het OnLaunched event wordt beschreven waarin onder andere de startpagina wordt aangegeven

**WistjeDatje**: Partial classes zijn classes wiens tekst beschreven wordt in verschillende class- definities (niet noodzakelijk maar bijna altijd worden deze class definities in verschillende files geschreven). Dit is uitermate zinvol bij gegenereerde code:

* Plaats de gegenereerde code in een aparte file waar ze bij aanpassingen hergegenereerd kan worden
* Plaats uw eigen custom code in een aparte file die niet overschreven wordt bij hergeneratie