

howest
hogeschool

Inleiding

Programming Basics

INHOUD

1	INLEIDING	4
1.1	Over deze syllabus	4
2	OVER DE PROGRAMMEERTAAL	5
2.1	C#	5
2.2	.NET	5
2.2.1	.NET Land, population: 3	5
2.2.2	.NET Core	6
2.2.3	Hoe het werkt	7
2.3	Programmeren is analyseren	7
2.4	Belangrijke begrippen	8
2.4.1	Assembly	8
2.4.2	.NET	8
2.4.3	Compiler	8
2.4.4	Design-time	8
2.4.5	Compile-time	8
2.4.6	Run-time	8
3	VISUAL STUDIO	9
3.1	Installatie	9
3.2	Rondleiding Visual Studio	10
3.2.1	Menu en toolbars	11
3.2.2	Windows	11
3.2.3	Extensions and updates	12
3.2.4	Projecten en solutions	13
4	.NET PROJECTEN	14
4.1	Console applicatie	14
4.1.1	Een nieuw project maken	14
4.1.2	Code, Build, Test, Repeat	17
1.1.1.1	Code	17
1.1.1.2	Build	20
1.1.1.3	Test + Debug	20
4.1.3	Build output	21
4.2	Windows Presentations Forms (WPF)	22
4.2.1	Een project toevoegen aan een solution	23
4.2.2	Control eigenschappen wijzigen	26
4.3	Webapplicatie	27
4.3.1	Een webapplicatie project maken	27

5	EEN EERSTE WPF APPLICATION	30
5.1	Project aanmaken	30
5.2	GUI elementen toevoegen en bewerken	31
5.3	Events afhandelen	34
6	WPF REFERENTIEMATERIAAL	37
6.1	Solutions en Projecten	37
6.2	Referentiemateriaal	37
7	DOCUMENTATIE OVER C#	38
7.1	In Visual Studio	38
7.2	Op het internet	39

1 INLEIDING

1.1 OVER DEZE SYLLABUS

Deze syllabus leert je programmeren met het .NET Core aan de hand van de programmeertaal C# en WPF.

Elk hoofdstuk behandelt een sleutelaspect van het programmeren aan de hand van voorbeelden en oefeningen. Tegelijkertijd leer je de basis voor het maken van een eenvoudige desktopapplicatie voor Windows met behulp van WPF.

Beperk je niet enkel tot het bestuderen van de codevoorbeelden die in deze syllabus staan, maar **experimenteer zelf** met eigen mini-projecten om zo tot een beter begrip van het geheel te komen.



Tijdens de lessen worden belangrijke aspecten uitgediept met deze syllabus als leidraad. Er wordt verwacht dat je deze syllabus zelfstandig doorneemt en de contactmomenten met de lector gebruikt om specifieke vragen te stellen.

2 OVER DE PROGRAMMEERTAAL

2.1 C#

Microsoft Visual C Sharp (zeg maar C#) is een moderne object-georiënteerde programmeertaal. Het is een zorgvuldig ontworpen taal met heel wat ingebouwde mogelijkheden en is zeer uitgebreid gedocumenteerd. Zoals bij elke hogere programmeertaal is C# zeer goed leesbaar voor een mens.



De programmeertaal werd oorspronkelijk ontworpen om te werken met Microsoft's .NET platform, en is een rechtstreeks concurrent met Java. Inmiddels heeft C# het klassieke .NET platform oversteegen en kan het ook gebruikt worden in het nieuwere, open source .NET Core en Mono, zodat talloze platforms (Windows, Linux, Mac, iOS, Android, spelconsole, enz.) ondersteund worden.

2.2 .NET

2.2.1 .NET LAND, POPULATION: 3

.NET (uitspraak: *dot net*) kent verschillende frameworks, die los van elkaar beschouwd kunnen worden. Als C# ontwikkelaar heb je **drie** smaken om uit te kiezen. Wanneer je informatie zoekt op het internet is het belangrijk om goed te weten voor welk Framework de informatie van toepassing is.

- **.NET Framework**

Dit is het oorspronkelijke, traditionele framework dat geleverd wordt met Windows. Het wordt voornamelijk gebruikt voor de ontwikkeling van Windows applicaties of standaard ASP.NET 4.x webapplicaties voor IIS, de webserver voor Windows systemen. Dit framework wordt standaard meegeleverd met je Windows installatie.

- **.NET Core**

Een cross platform versie van het oorspronkelijke framework waarmee applicaties gebouwd kunnen worden die op verschillende systemen kunnen uitgevoerd worden: Windows, Linux, Mac en Docker. **Deze syllabus behandelt voornamelijk dit framework. Merk wel op dat de GUI enkel op Windows gebruikt kan worden.**

- **Xamarin (Mono.Net)**

Dit framework wordt voornamelijk gebruikt voor de ontwikkeling van mobiele apps. Het is gebaseerd op Mono.NET, de oorspronkelijke open source versie dat het .NET Framework volledig tracht na te bouwen.

.NET FRAMEWORK	.NET CORE	XAMARIN
Platform for .NET applications on Windows	Cross-platform and open source framework optimized for modern app needs and developer workflows	Cross-platform and open source Mono-based runtime for iOS, OS X, and Android devices
Distributed with Windows	Distributed with app	Distributed with app

Voor applicaties gebouwd voor het .NET Framework is het vereist om dat Framework vooraf te hebben geïnstalleerd, dat is vergelijkbaar met Java. Voor .NET Core en Xamarin is dat niet zo. De benodigde bibliotheken kunnen met de applicatie verscheept worden naar de eindgebruiker.

2.2.2 .NET CORE

In deze syllabus zal je voornamelijk applicaties programmeren voor het .NET 5.0 Framework. Dit Framework is aan te bevelen bij het maken van nieuwe applicaties, omdat het niet enkel bruikbaar is voor Windows platformen. Ook al kan men .NET Core gebruiken voor Linux, macOS, ... systemen, toch zijn we in deze syllabus gebonden aan de Grafische User Interface (GUI) omdat we gebruik zullen maken van WPF-applicaties (Windows Presentation Forms).

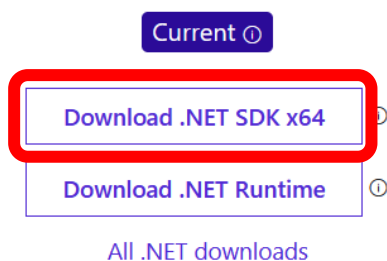
Het .NET Core dien je zelf nog te installeren. Voor deze module zullen we gebruiken maken van .NET 5.0. Aangezien we zelf applicaties zullen programmeren, dienen we de .NET Core SDK te kiezen en niet de runtime. SDK staat voor Software Developer Kit.

Je kan deze hier downloaden: <https://dotnet.microsoft.com/download>

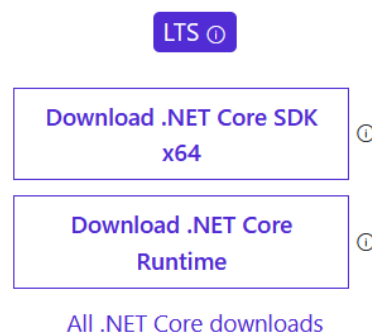


.NET is a free, cross-platform, open-source developer platform for building many different types of applications.

.NET 5.0 (recommended)



.NET Core 3.1

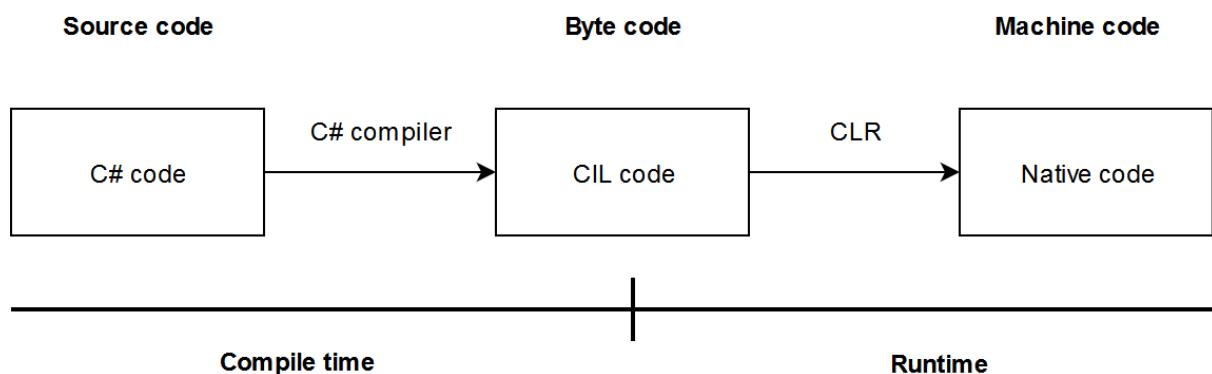


De eindgebruiker van jouw applicatie zal echter de .NET Framework versie waarvoor jouw applicatie geschreven werd geïnstalleerd moeten hebben. Elke nieuwe versie van Windows bevat de meest recente versie van het .NET Framework op dat moment.

2.2.3 HOE HET WERKT

Wanneer je een programma schrijft, is het uiteindelijke de bedoeling om dit programma uit te voeren op een machine. In C# wordt de broncode **gecompileerd** naar een tussentaal, de Common Intermediate Language (CIL). Het is deze tussentaal die terug te vinden is in de resulterende **.exe** en **.dll** bestanden. Na het maken van deze **binaries** is de compilatiefase, ook wel de **compile time** genoemd, afgerond.

Elke machine waarop je programma wordt uitgevoerd, krijgt dezelfde assembly (.exe bestand) en dus dezelfde code. Machines en CPU's verschillen echter qua capaciteiten en dus wordt de **Common Language Runtime** (CLR) ingeschakeld. Bij het uitvoeren van het programma zal de CLR de CIL code vertalen naar verstaanbare instructies voor die specifieke machine. De uitvoering van het programma noemen we **run time**.



2.3 PROGRAMMEREN IS ANALYSEREN

Wanneer je een applicatie programmeert dan is er een duidelijk doel voor die applicatie. Er zijn een aantal vereisten waaraan de applicatie moet voldoen, bijvoorbeeld: beheren van digitale facturen, afdrukken van facturen.

Voor je kan beginnen met het programmeerwerk moet je deze vereisten goed begrijpen en analyseren. Denk daarbij aan het volgende:

- Vereisten
 - Zijn de vereisten duidelijk?
 - Kunnen de vereisten opgesplitst worden in kleinere deelproblemen?
- Functioneren van het programma:
 - Wat zijn de gegevens waarmee gewerkt moet worden?
 - Wat moet er met die gegevens gebeuren?
 - Wat moet er met het resultaat gebeuren?
 - Welke gebruikersinteractie moet er plaatsvinden?

2.4 BELANGRIJKE BEGRIPPEN

2.4.1 ASSEMBLY

Een binair bestand dat CIL bytecode bevat die door de .NET CLR uitgevoerd wordt. Het bevat de vertaling van jouw broncode. Assemblies komen meestal in twee vormen:

- **Executable:**
een uitvoerbaar bestand dat eindigt op de .exe bestandsextensie.
- **Library:**
een bestand dat gemeenschappelijke functionaliteiten bevat die door andere assemblies gebruikt kunnen worden. De bestandsnaamextensie is meestal .dll

2.4.2 .NET

.NET is een verzameling assemblies (meestal libraries) die de programmeur toegang geven tot zeer uiteenlopende functionaliteiten. Er zijn drie versies van .NET: het klassieke .NET Framework, .NET Core en Mono.

2.4.3 COMPILER

Het programma dat jouw broncode omzet naar bytecode. In C# is dat de C Sharp Compiler, csc.exe dat te vinden is de installatiemap van het .NET Framework.

2.4.4 DESIGN-TIME

De ontwikkelingsfase waar de programmeur broncode schrijft en bewerkt.

2.4.5 COMPILE-TIME

De ontwikkelingsfase waar de compiler de broncode omzet naar een assembly.

2.4.6 RUN-TIME

Het uitvoeren van een programma door een eindgebruiker wordt ook wel run-time genoemd. Voor .NET applicaties wordt deze uitvoering geregeld door de Common Language Runtime, waarvoor een installatie van het .NET Framework nodig is op de machine.



Meer informatie

- [MS: Download .NET](#)
- [MS: .NET Programming languages](#)
- [MS: C# Guide](#)

3 VISUAL STUDIO

Microsoft Visual Studio is de meest gebruikte, meest complete Integrated Development Environment (IDE) voor het ontwikkelen van applicaties voor het .NET Framework. Het .NET Framework, samen met heel wat tools voor het ontwikkelen, wordt gratis ter beschikking gesteld. Visual Studio komt in verschillende uitvoeringen: Visual Studio Professional, Enterprise en Ultimate zijn commerciële versies waarvoor een licentiekost moet worden betaald.

Microsoft stelt echter een Community-editie van Visual Studio ter beschikking. Deze versie heeft dezelfde mogelijkheden als de Professional Edition en wordt aangeboden voor educatief, persoonlijk en zelfs commercieel gebruik voor bedrijven met max. 5 werknemers.

In deze cursus ga je aan de slag met de **Visual Studio 2019 Community Edition**.



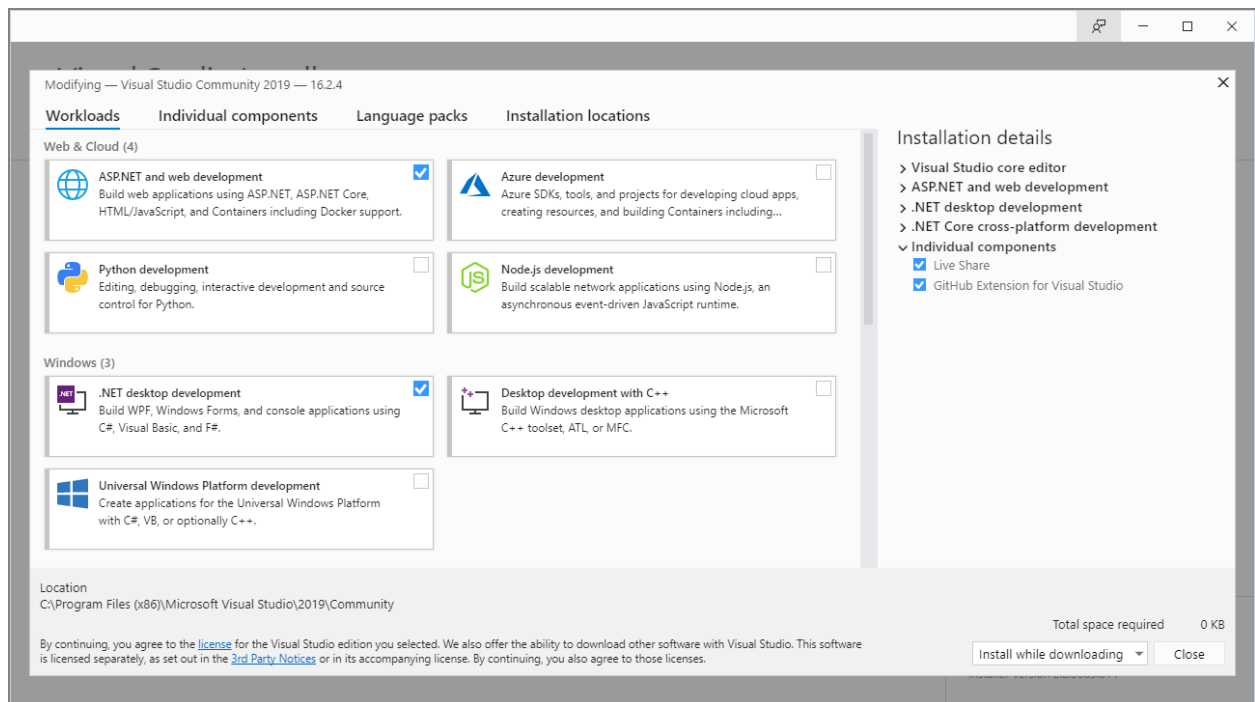
Visual Studio is slechts een **hulpmiddel** om makkelijk een .NET applicatie te programmeren.

Je kan ook .NET applicaties ontwikkelen met tal van andere gratis editoren; kladblok, notepad++, MonoDevelop en Visual Studio Code (niet te verwarren met Visual Studio zelf) zijn daar voorbeelden van.

3.1 INSTALLATIE

Je installeert Visual Studio als volgt:

1. Download Visual Studio Community Edition via <https://www.visualstudio.com/vs/community/>
2. Voer het installatiebestand uit.
3. Je krijgt een vraag om akkoord te gaan met de Licentieovereenkomst voor je kan doorgaan.
4. Eens het installatieprogramma is opgestart, krijg je het volgende scherm te zien:



Een ontwikkelaar hoeft zelden over alle ontwikkeltools te beschikken die Visual Studio te bieden heeft. Daarom worden alle beschikbare functionaliteiten van Visual Studio gemakshalve gebundeld in zogenaamde **Workloads**. Hiermee krijg je alle tools en functionaliteiten gekoppeld aan een bepaalde ontwikkelingstak; bijvoorbeeld webapplicaties, desktopapplicaties, mobile apps, enzovoort.

5. Installeer de volgende Workloads, zodat je alle voorbeelden in deze syllabus kan uittesten:

- **.NET desktop development**
- **ASP.NET and web development**
- **.NET Core cross-platform development**

Ben je nieuwsgierig van aard en wil je op verkenning? Goed zo! Installeer dan ook eens de volgende Workloads:

- Mobile development with .NET
- Game development with Unity

Controleer of je genoeg schijfruimte over hebt voor de gekozen Workloads.

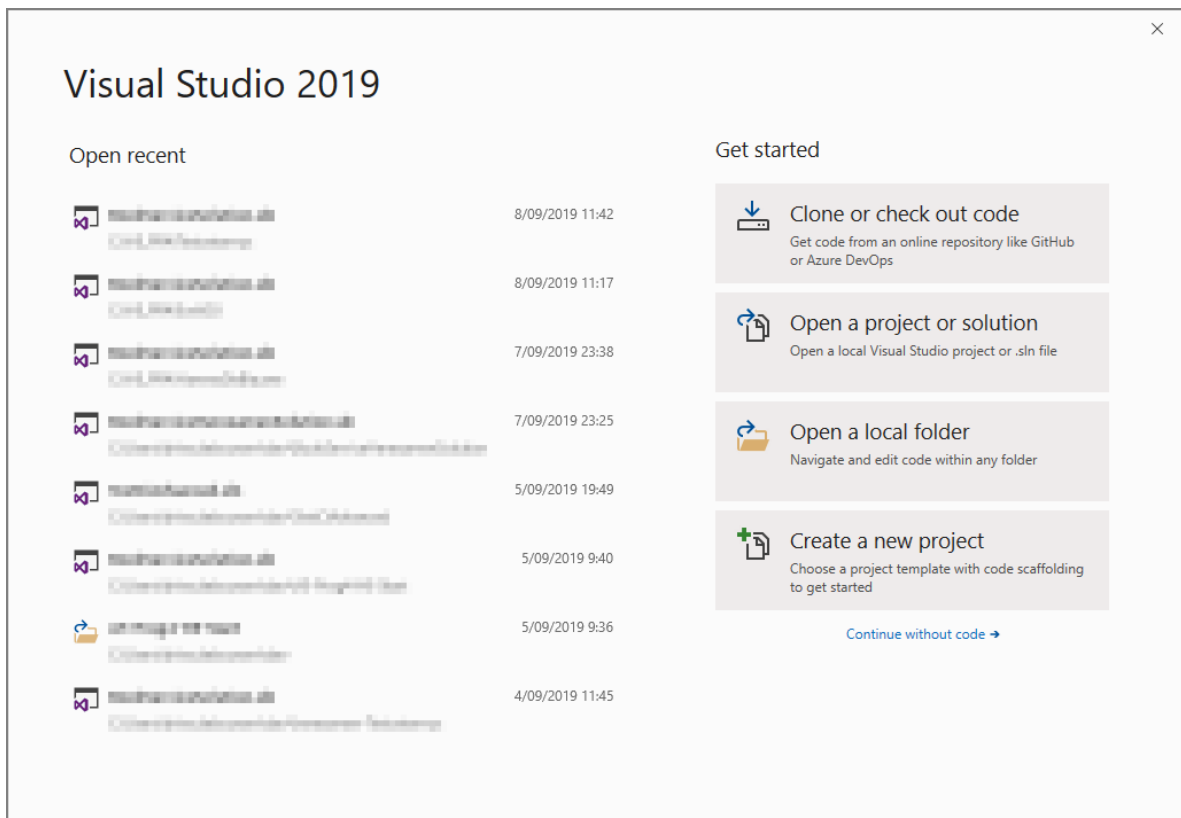
6. Als je klaar bent met kiezen, klik je op **Install**.

7. Wacht tot de installatie voltooid is. Daarna kan je Visual Studio starten via de **Launch** knop, of gewoon vanuit het Startmenu.

3.2 RONDLEIDING VISUAL STUDIO

Als je Visual Studio opstart dan komt je in het Start scherm terecht. Dit toont je de meest recente projecten waaraan je hebt gewerkt en enkele andere opties die we later nog zullen bespreken.

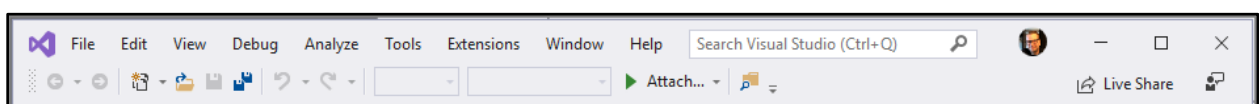
Klik voor nu op **Continue without code** om op verkenning te gaan in Visual Studio.



3.2.1 MENU EN TOOLBARS

Bovenaan het hoofdvenster van Visual Studio bevinden zich het menu en de toolbars. Het menu biedt toegang tot alle functionaliteiten die de IDE te bieden heeft en is te veelomvattend om hier te bespreken. In deze syllabus leer je gaandeweg het menu beter kennen.

De toolbars bevatten de contextuele taken die beschikbaar zijn voor de huidige situatie. Dat betekent dat de knoppen en toolbars zullen veranderen naargelang het bestand dat je bewerkt. Ook deze zaken leer je stap voor stap kennen.

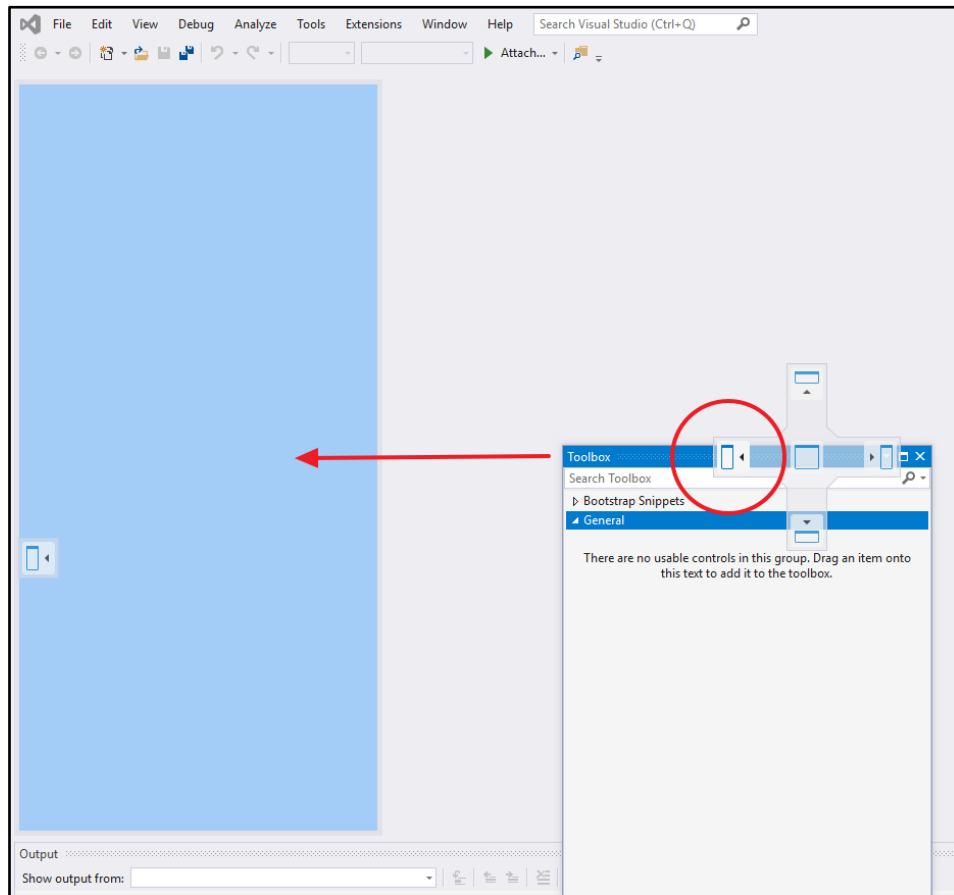


3.2.2 WINDOWS

Visual Studio is een zeer uitgebreide IDE die je volledig zelf kan inrichten. Het bestaat uit heel wat vensters die je kan tonen, verbergen en wijzigen van positie en grootte. Je hoeft ze heus niet allemaal te kennen, zolang je maar weet waar ze te vinden zijn. Maak het alvast een beetje gezellig, want dit wordt je nieuwe programmeeromgeving!

- Het actieve vensters is steeds gekleurd in een accent. In de beginsituatie is dat de Start Page.
- Om een vensters te sluiten, klik op het kruisje.
- Om een vensters te verslepen, versleep je de titelbalk ervan.
- Je kan deze uit het Visual Studio hoofdvenster loskoppelen (handig als je twee monitoren hebt)

- Je kan deze **docken** op een andere locatie in het hoofdvenster. Daartoe sleep je het venster naar het gewenste dock-icoontje. Onderstaande screenshot toont je hoe het toolbox venster aan de linkerkant geplaatst kan worden:



- Vensters die niet zichtbaar zijn kunnen meestal gevonden worden via het menu **View**. Andere, specifiekere Windows zijn te vinden onder het submenu Windows van de volgende menu's:
 - Debug
 - Test
 - Analyse
- Experimenteer eens grondig met deze vensters. Open, sluit en verplaats ze naar hartenlust! Eens je er een grondig zootje van hebt gemaakt, kan je de standaard lay-out terug oproepen via de volgende optie: **Window → Reset Window Layout**.

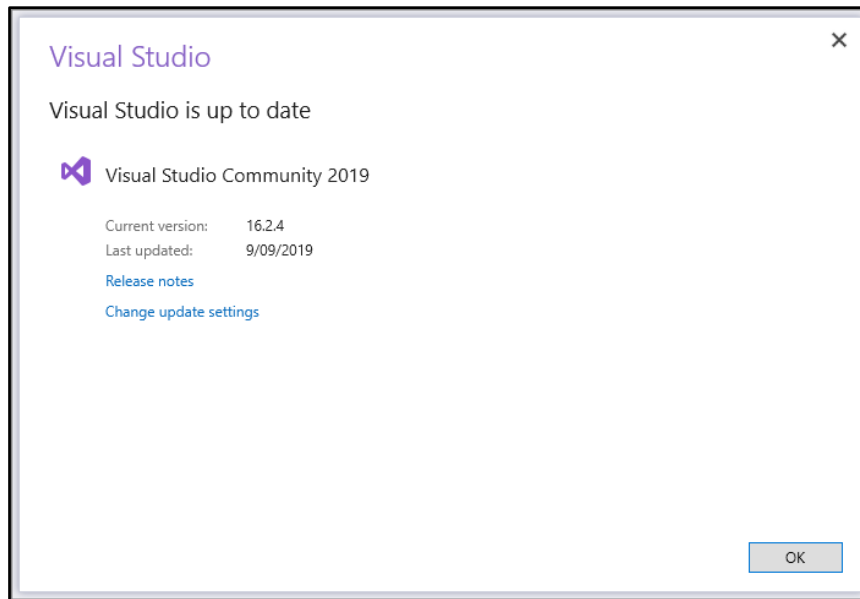
3.2.3 EXTENSIONS AND UPDATES

Behalve de standaard ingebouwde functionaliteiten zijn er heel wat handige tools en uitbreidingen beschikbaar voor Visual Studio. Deze kan je vinden via **Extensions → Manage Extensions**

Je ziet standaard welke tools en extensies je hebt geïnstalleerd via **Installed**. Om te zoeken naar nog niet geïnstalleerde uitbreidingen klik je op **Online**.

Tenslotte kan je de extensies up-to-date houden via **Updates**.

Indien je **Visual Studio** al op voorhand had geïnstalleerd, neem je best eens een kijkje of jouw versie de meest recente is. Dit doe je via **Help → Check for Updates**:



3.2.4 PROJECTEN EN SOLUTIONS

De manier waarop Visual Studio broncode samenhoudt, gebeurt in de vorm van projecten. Een project is een verzameling bestanden die bij elkaar horen en die bij compilatie een assembly opleveren (een .dll of .exe bestand).

Een project heeft meestal de extensie **.csproj** en is een wezen een oplist van alle bestanden die tot dat project behoren. De bestanden zelf worden uiteraard op schijf bewaard en zijn al dan niet in (sub)mappen ondergebracht.

Sommige applicaties bestaan uit meerdere assemblies, en dus meerdere projecten. In dat geval kunnen deze projecten gebundeld worden met behulp van een Solution.



Een **project** bevat een verzameling bestanden die onderdeel uitmaken van een programma of library.

Een **solution** bevat een verzameling projecten die met elkaar verwant zijn of elkaar nodig hebben om te functioneren.

4 .NET PROJECTEN

De eerste toepassing die je maakt met een nieuwe programmeertaal heeft vaak niet veel om het lijf: het is een test of er 'leven in de brouwerij' is. Vaak wordt in het eerste programma enkel een zin op het scherm gezet: "Hello World".

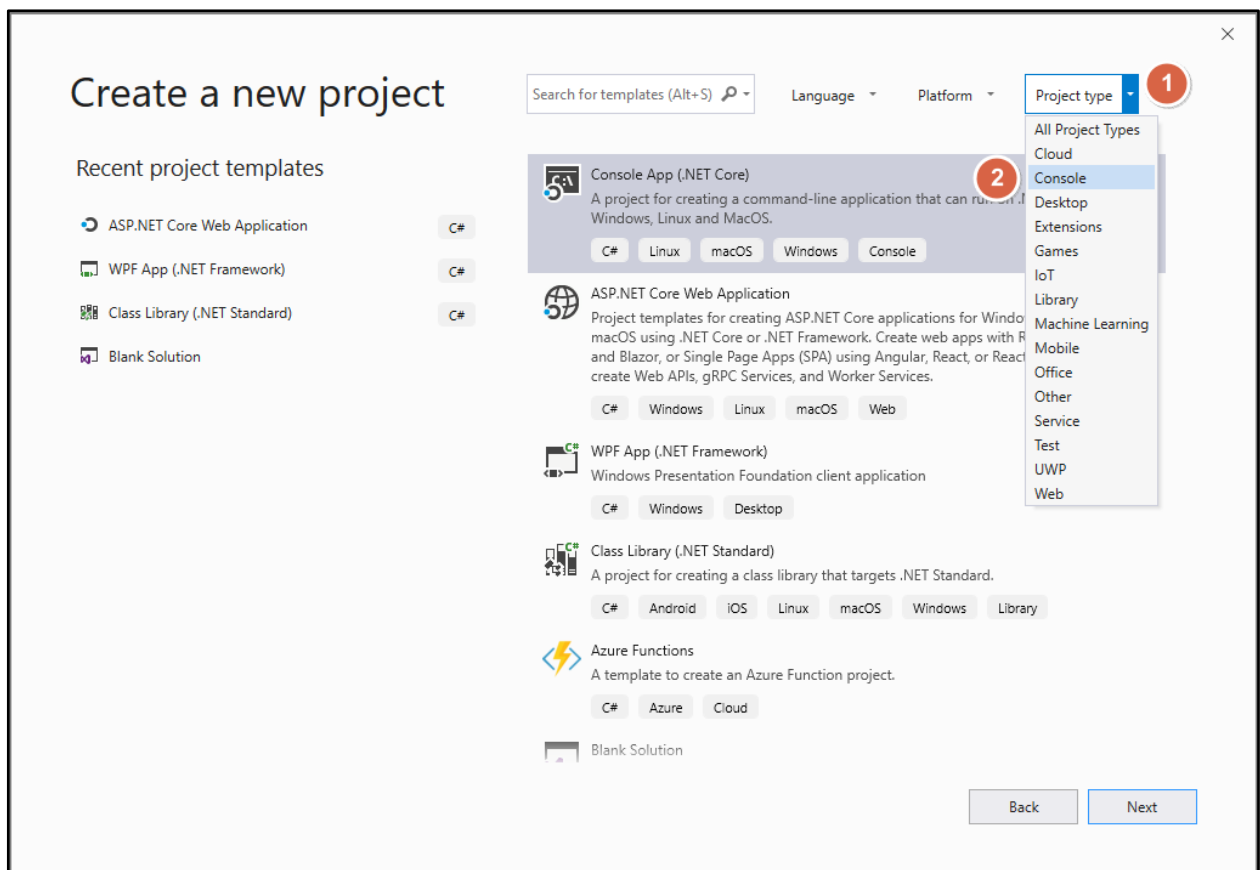
Je gaat nu zo'n Hello World applicatie een aantal keer bouwen in een aantal verschillende applicaties die .NET ons biedt.

4.1 CONSOLE APPLICATIE

Je maakt nu je eerste project, een Console applicatie. Dat zijn programma's die uitgevoerd worden in de command line en louter textueel zijn. Ze kunnen tekst inlezen en afbeelden op het scherm. Dat laatste is wat we voorlopig zullen doen.

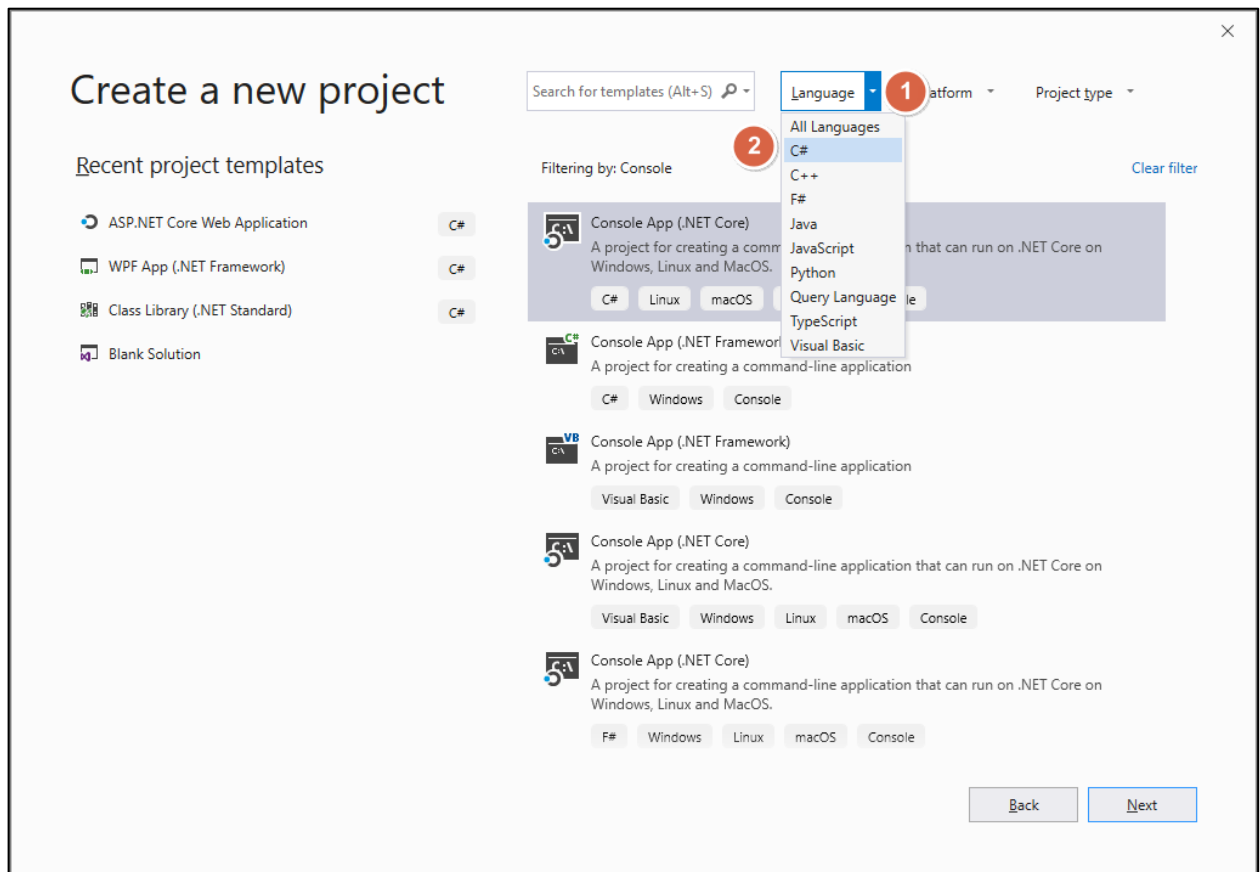
4.1.1 EEN NIEUW PROJECT MAKEN

1. Start Visual Studio en kies **Create a new project** (Indien je Visual Studio al opgestart was kan je ook kiezen voor **File → New Project**)
2. In het nieuw dialoogvenster kun je filteren op bepaalde projecten. Klik op **Project Type** en kies voor **Console**:

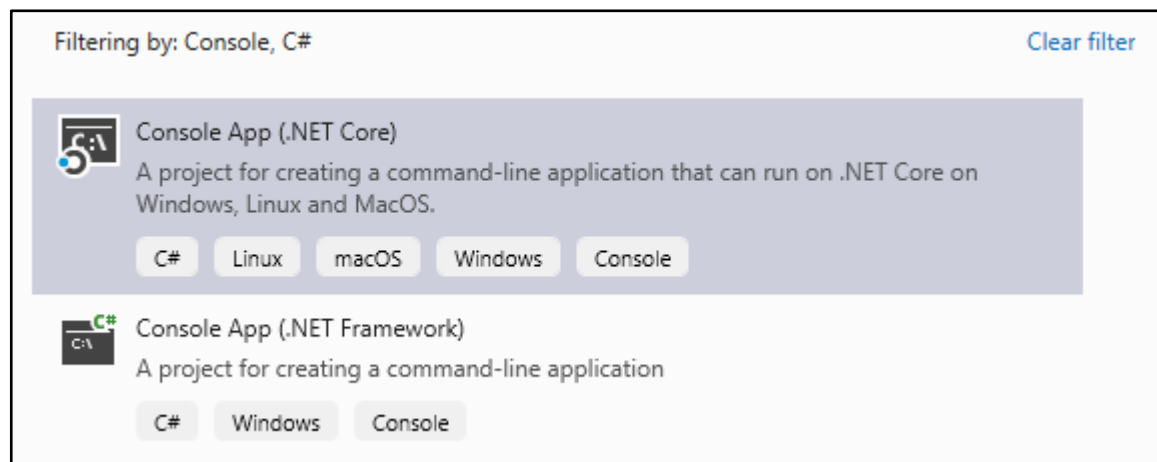


We zien nu enkel en alleen de Console template projecten. Aangezien we Visual Studio ook kunnen gebruiken om een console applicatie in Visual Basic te programmeren in plaats van C#, zien we ook deze opties nog staan in de lijst.

We filteren onze lijst met templates nog wat meer. Klik op **Language** en kies voor **C#**:

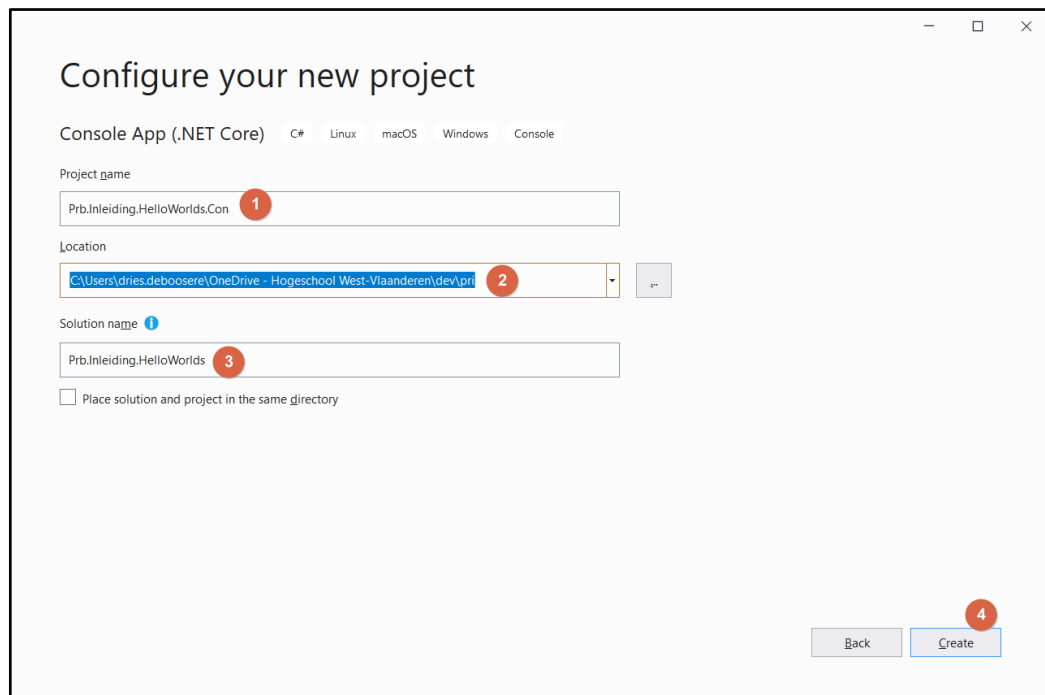


We houden nu nog twee opties over: Console App (.NET Core) en Console App (.NET Framework):

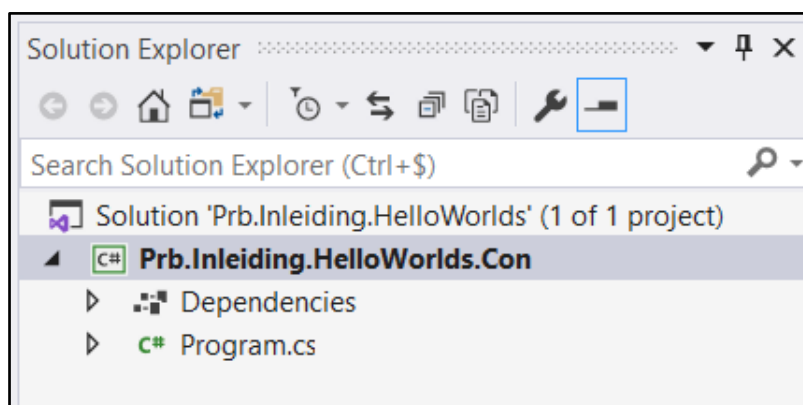


Zoals reeds eerder vermeld hebben we meerdere .NET Frameworks. Bij onze opties is al goed te zien dat we onze .NET Framework Console App enkel kunnen gebruiken op Windows. Waar we bij de .NET Core Console App zien dat we deze ook kunnen gebruiken voor Linux, macOS en Windows. Wij zullen in de module Programming Basics gebruik maken van .NET Core en **niet** van .NET Framework.

3. Selecteer **Console App (.NET Core)** en klik op **Next**.
4. Op het volgende scherm geef je volgende zaken in:
 - Je Project name (Prb.Inleiding.HelloWorlds.Cons)
 - Je Location (de locatie van je project op je computer)
 - Je Solution name (Prb.Inleiding.HelloWorlds)
 - Klik daarna op de knop Create.



5. Visual Studio genereert nu een basisproject aan de hand van het gekozen sjabloon.
6. Je kan de inhoud van je project zien in het **Solution Explorer** venster van Visual Studio. Je vindt dit meestal aan de rechterkant van je scherm, of via **View → Solution Explorer**.



7. Je merkt dat er al een aantal bestanden zijn aangemaakt, onder meer een C# bestand genaamd **Program.cs**. Deze bestanden zijn uiteraard ook op je bestandssysteem terug te vinden, met name in de map die je hebt gekozen toen je het project genereerde.

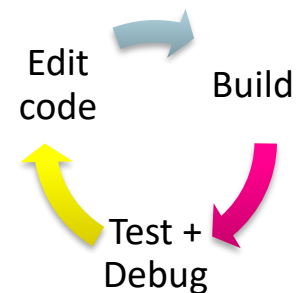
8. Een overzicht van wat er te zien valt in de Solution Explorer:

Solution "Prb.Inleiding.HelloWorlds"	Het solutionbestand. Een solution bundelt projecten zodat je ze in hetzelfde Visual Studio hoofdvenster instantie kan bewerken.
Prb.Inleiding.HelloWorlds.Cons	Het C# projectbestand. Een project verzamelt files met broncode, figuren, compilatie instellingen, ... voor een c-sharp project.
Dependencies	Een map die de verwijzingen bijhoudt die vanuit je project legt naar andere assemblies die ofwel deel uitmaken van het .NET Framework, aangeleverd werden door een andere ontwikkelaar of gewoon door jezelf werden geprogrammeerd.
Program.cs	Een eerste C# codefile dat reeds wat code bevat. Dit bestand werd automatisch aangemaakt door Visual Studio en werd alvast geopend in een venster.

4.1.2 CODE, BUILD, TEST, REPEAT

De programmeercyclus bestaat uit het wijzigen of toevoegen van codebestanden, waarop de broncode gecompileerd wordt (Build). De programma uitvoering wordt vervolgens getest (Test) op het gewenste resultaat, en fouten worden opgespoord (Debug). Hierna worden er opnieuw wijzigen aangebracht en herhaalt deze cyclus zich tot het eindresultaat bereikt wordt.

Het is een goede gewoonte om regelmatig de effecten van je code te bekijken volgens dit vereenvoudigd patroon. Zo niet kunnen de fouten zich opstapelen tot een onoverzichtelijk kluwen die voor een beginnende programmeur vele uren kan kosten.



1.1.1.1 CODE

Open het bestand **Program.cs** door erop te dubbelklikken in de Solution Explorer. De inhoud van dit bestand ziet er zo uit:

```
using System;

namespace Prb.Inleiding.HelloWorlds.Cons
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

In het codebestand zie je reeds een aantal belangrijke keywords uit de C# syntax:

methode	Het stukje code dat begint met <code>static void Main</code> is een methode . Methodes bevatten instructies tussen de accoladen <code>{</code> en <code>}</code> die één voor één worden uitgevoerd. De <code>Main</code> methode is hier een zeer belangrijke methode; het vormt het startpunt van de applicatie.
class	Een basiscomponent voor object-georiënteerd programmeren. In een klasse worden eigenschappen, methoden en andere zaken gedefinieerd die structureel en/of logisch gezien bij elkaar horen. De naam van een klasse moet steeds uniek zijn. Alles wat tot de klasse <code>Program</code> behoort staat tussen de bijbehorende accolades <code>class Program { ... }</code> .
namespace	<p>Verzamelnaam voor een groep gerelateerde klassen. De volledige naam van een klasse wordt steeds voorafgegaan door zijn namespace. De klasse <code>Program</code> heet dus eigenlijk <code>Prb.Inleiding.HelloWorlds.Cons.Program</code>.</p> <p>Door het gebruik van namespaces blijft het mogelijk om een tweede klasse genaamd <code>Program</code> te maken, het zij dan wel in een andere namespace zodat de uniciteit gerespecteerd blijft.</p>

Maak nu volgende wijzigingen in dit bestand:

```
using System; ④

namespace Prb.Inleiding.HelloWorlds.Cons
{
    class Program
    {
        static void Main(string[] args)
        {
            /* let's output some text ③
             * in the console window */
            ① Console.WriteLine("Hello World");
            ② string input = Console.ReadLine(); //wait for user to hit enter and read ③
all text input
        }
    }
}
```

Welke wijzigingen hebben we nu precies gedaan?

1. De eerste instructie in de `Main()` methode is het afbeelden van een stukje tekst. Dit doe je met de methode `WriteLine()` die te vinden is in de klasse genaamd `Console`.
2. De tweede en laatste instructie is het inlezen van tekst tot de gebruiker op de enter-toets drukt (en dus een nieuwe lijn ingeeft). Deze tekst wordt bewaard in een stukje geheugen dat je `input` hebt genoemd. Dit kan met behulp van de `Readline()` methode, die eveneens behoort tot de `Console` klasse.
3. De groene tekst is commentaar. Deze wordt genegeerd door de compiler en wordt dus niet uitgevoerd. Gebruik het om jezelf en andere programmeurs te verklaren wat bepaalde instructies precies doen.

`/* */` Plaatst commentaar over meerdere lijnen heen. Hiermee kan je uitgebreide uitleg verschaffen over bepaalde stukken code.

`//` Commentaar voor de rest van de lijn. Hierna kunnen er geen instructies meer volgen. De rest van de lijn wordt genegeerd. Handig voor korte uitleg.

4. Een aantal `using` instructies bovenaan het codebestand werden verwijderd omdat ze toch niet gebruikt worden. De enige `using` statement die benut wordt is `using System;`

Dankzij het keyword `using` gevolgd door de naam van een namespace hoef je niet de volledige naam van een klasse te gebruiken. Alle klassen die bevat zijn in die namespace kan je gewoon in hun korte notatie schrijven.

In dit geval importeren we de .NET Core namespace genaamd `System`. In deze namespace zit onder meer de klasse genaamd `Console`. Had je deze namespace niet geïmporteerd, dan moest je de volledige naam van die klasse gebruiken: `System.Console`.

Dat zou dus langere, onleesbaardere code opleveren. Het gebruik van `using` is overigens volledig vrijblijvend, maar bijzonder nuttig. Het verwijderen van ongebruikte `using` statements heeft geen enkel effect op de uitvoering van het programma, maar levert ook weer nettere code.



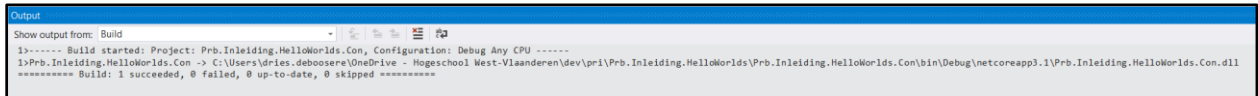
Opgelet

- Elke statement in C# wordt afgesloten met een **puntkomma** ;
- C# syntax is **hoofdlettergevoelig**: een h is dus niet gelijk aan een H.

1.1.1.2 BUILD

Je bent nu klaar voor de compilatie van de kersverse wijzigingen. De compilatie wordt gedaan met het programma `csc.exe` dat op de achtergrond zal worden uitgevoerd. Je doet dat via het menu **Build → Build Solution** of de sneltoets **CTRL + SHIFT + B**.

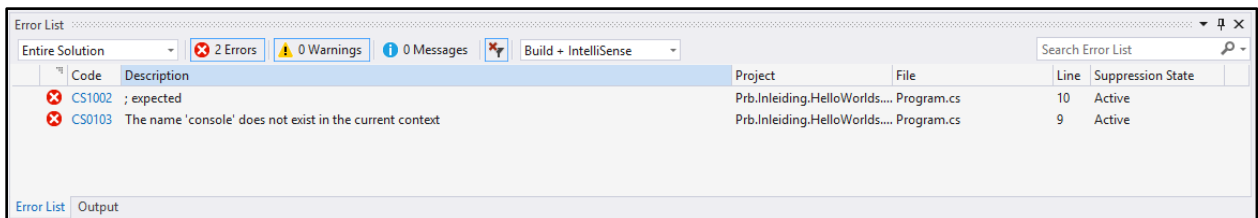
Onderaan in de Output venster zie je de voortgang van de compilatie. Indien er fouten optreden dan wordt je daarvan op de hoogte gesteld.



```
Output
Show output from: Build
1>----- Build started: Project: Prb.Inleiding.HelloWorlds.Con, Configuration: Debug Any CPU -----
1>Prb.Inleiding.HelloWorlds.Con -> C:\Users\dries.deboosere\OneDrive - Hogeschool West-Vlaanderen\dev\prj\Prb.Inleiding.HelloWorlds\Prb.Inleiding.HelloWorlds.Con\bin\Debug\netcoreapp3.1\Prb.Inleiding.HelloWorlds.Con.dll
***** Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped *****
```

Na een succesvolle compilatie wordt het executable bestand gemaakt. In dit geval is dat **HelloWorlds.Cons.exe** dat zich bevindt in de **/bin/debug/net5.0-windows** map van ons project.

Mocht je foutieve code hebben ingevoerd, dan zal de compiler stoppen. Er wordt geen exe bestand aangemaakt. In plaats daarvan verschijnt het venster genaamd Error List, dat je vertelt waar de fout zich bevindt. In het onderstaande voorbeeld heb ik twee fouten gemaakt. Kan je ontcijferen welke?



Dit zijn de fouten:

- Op lijn 9 staat “console” met een kleine “c” geschreven. De compiler begrijpt dit niet. Ik bedoelde wellicht de klasse “Console” met een hoofdletter. Onthoud dat C# hoofdlettergevoelig is.
- Op lijn 10 wordt er een puntkomma verwacht. Eigenlijk betekent dit dat ik op lijn 9 een puntkomma vergeten te plaatsen ben om het statement te beëindigen.

Probeer gerust zelf eens een fout te introduceren in de code zodat het compileren mislukt. Bestudeer het Error List venster en corrigeer de fouten alvorens je verder gaat.

1.1.1.3 TEST + DEBUG

Om een project uit te voeren ga je naar **Debug → Start Debugging**. Je kan ook op de **F5** toets drukken of de **Start** knop in de debug toolbar gebruiken.

Voer de applicatie uit via **Debug → Start Debugging**. Het consolescherm verschijnt met het verwachte resultaat.



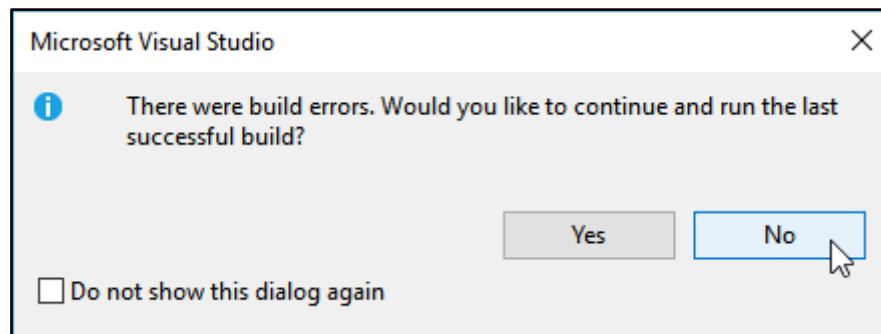
Tijdens de uitvoering met de Debugger kan je de broncode niet meer wijzigen. Let ook op de gewijzigde toolbar in Visual Studio:



Omdat de laatste instructie van het programma een `Console.ReadLine()` betreft, moet je op de enter-toets drukken om verder te gaan. De applicatie wordt automatisch afgesloten na uitvoering van de laatste instructie.

Wanneer je de broncode wijzigt dan zal het starten van de applicatie automatisch een hercompilatie tot gevolg hebben. Je kan dus bij elke wijziging direct op **F5** drukken om het resultaat van je wijzigingen te zien.

Indien je code compilatiefouten bevat en je drukt op **F5**, dan krijg je het volgende dialoogvenster:

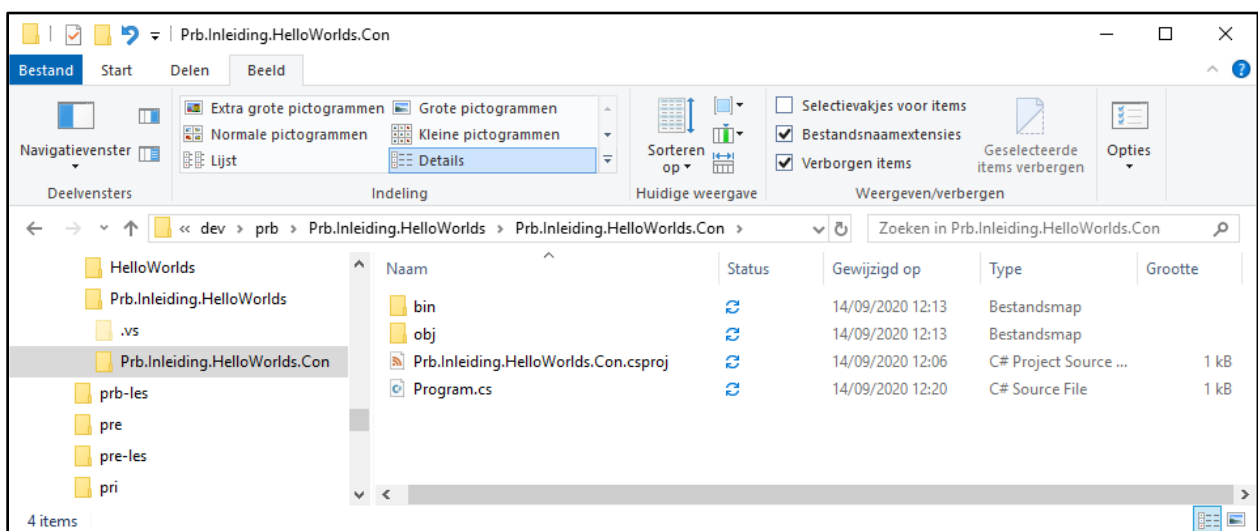


Kies bij dit scherm steeds voor **No**, want je wil immers de meest recente versie starten. Je moet dan eerst de fouten corrigeren zodat je project gecompileerd kan worden.

4.1.3 BUILD OUTPUT

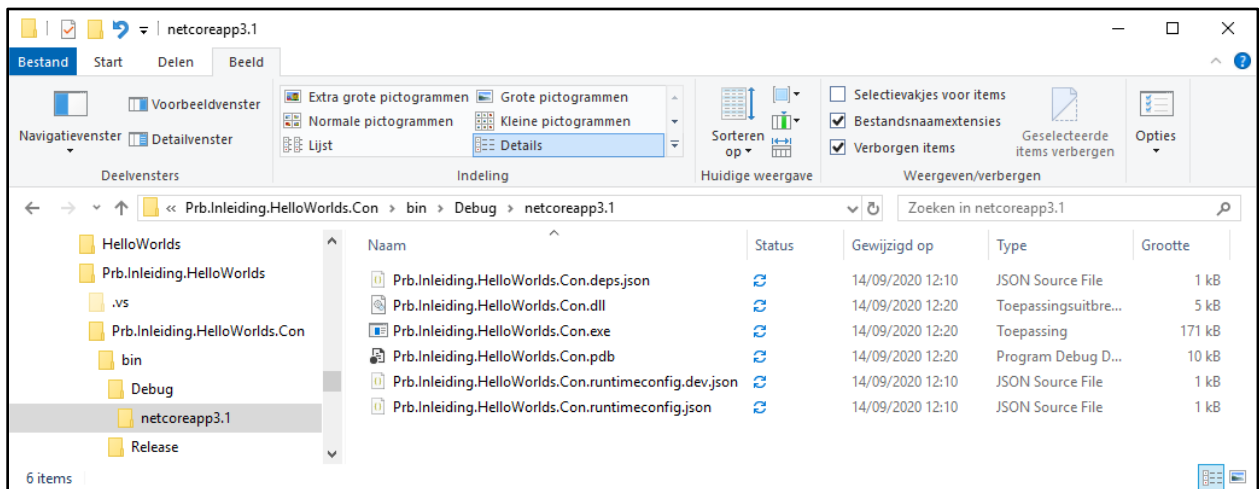
Wanneer je het project compileert, manueel met behulp van **CTRL+SHIFT+B** of automatisch bij uitvoering ervan, dan wordt het resultaat daarvan in de **/bin** map geplaatst. De bin map (staat voor “binaries”) bevindt zich in de projectmap.

Om snel naar de projectmap in je Windows Verkenner zichtbaar te maken kan je rechtsklikken op de projectnaam in de Solution Explorer en kies je voor **Show Folders in File Explorer**.



Je bemerkt de **bin** en **obj** mappen. Deze bevatten elk een submap genaamd **Debug**. We hebben het project immers gecompileerd als ontwikkelingsversie. Een productieve versie resulteert in een gelijkaardige map genaamd Release.

Open de **/bin/Debug/net5.0-windows** map en bekijk de inhoud. Je vindt er drie bestanden die dezelfde naam hebben als je project. Zet de weergave van bestandsnaamextensies in Windows Verkenner gemakshalve aan.



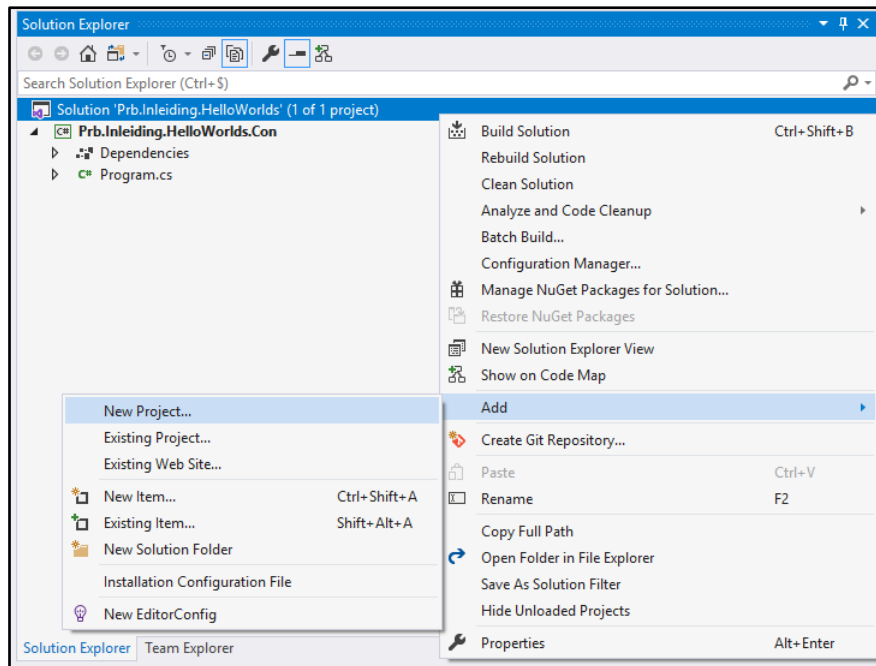
- .exe** Dit is de resulterende **assembly** van het project. Het bevat de IL bytecode die door de .NET Framework CLR kan worden uitgevoerd. Je kan jouw applicatie dus ook van hieruit starten. Wanneer je op **F5** drukt is dit het bestand dat wordt uitgevoerd.
- .pdb** Dit bestand wordt gebruikt om fouten op te kunnen sporen in Debug mode. We komen daar later op terug.
- .json** Deze bestanden verwijzen naar de dependencies of runtime configuraties.
- .dll** Dynamic Link Library. Dit bestand is een bibliotheek die code en gegevens bevat om door het programma (.exe) gebruikt wordt of kan gebruikt worden.

4.2 WINDOWS PRESENTATIONS FORMS (WPF)

Je maak nu een tweede applicatie met een Graphical User Interface (GUI) in de vorm van een venster. Het project dat deze applicatie bevat mag deel uitmaken van dezelfde solution.

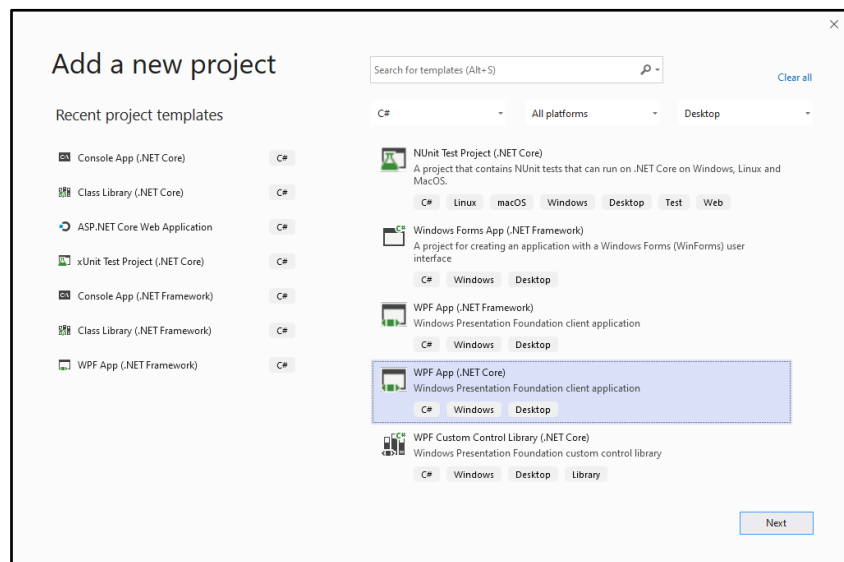
4.2.1 EEN PROJECT TOEVOEGEN AAN EEN SOLUTION

1. Zorg dat je solution genaamd **Prb.Inleiding.HelloWorlds** geopend is in Visual Studio.
2. Rechtsklik op de solution in de Solution Explorer en kies **Add → New Project**



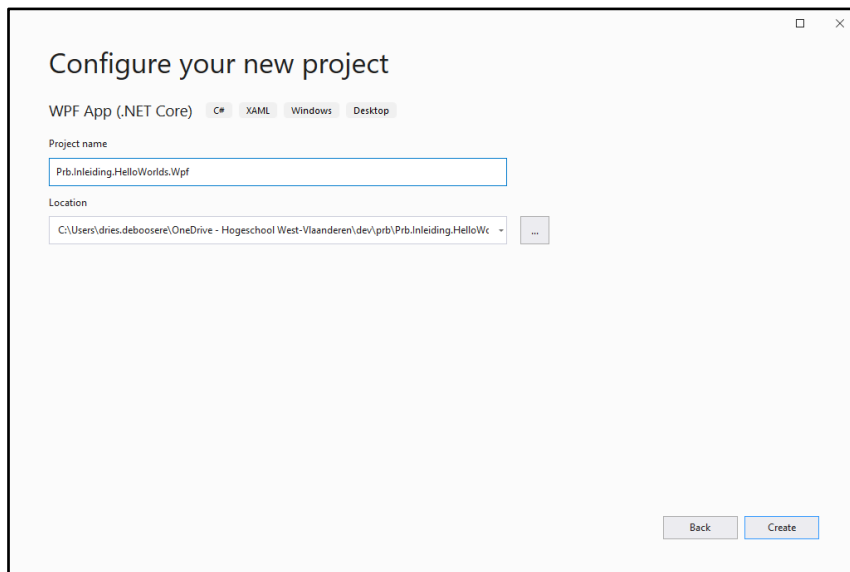
3. In het **New Project** dialoogvenster kies je de volgende opties:

- Project type: Desktop
- Project template: WPF App (.NET Core)
- Klik op Next



Kies een naam voor dit nieuw project:

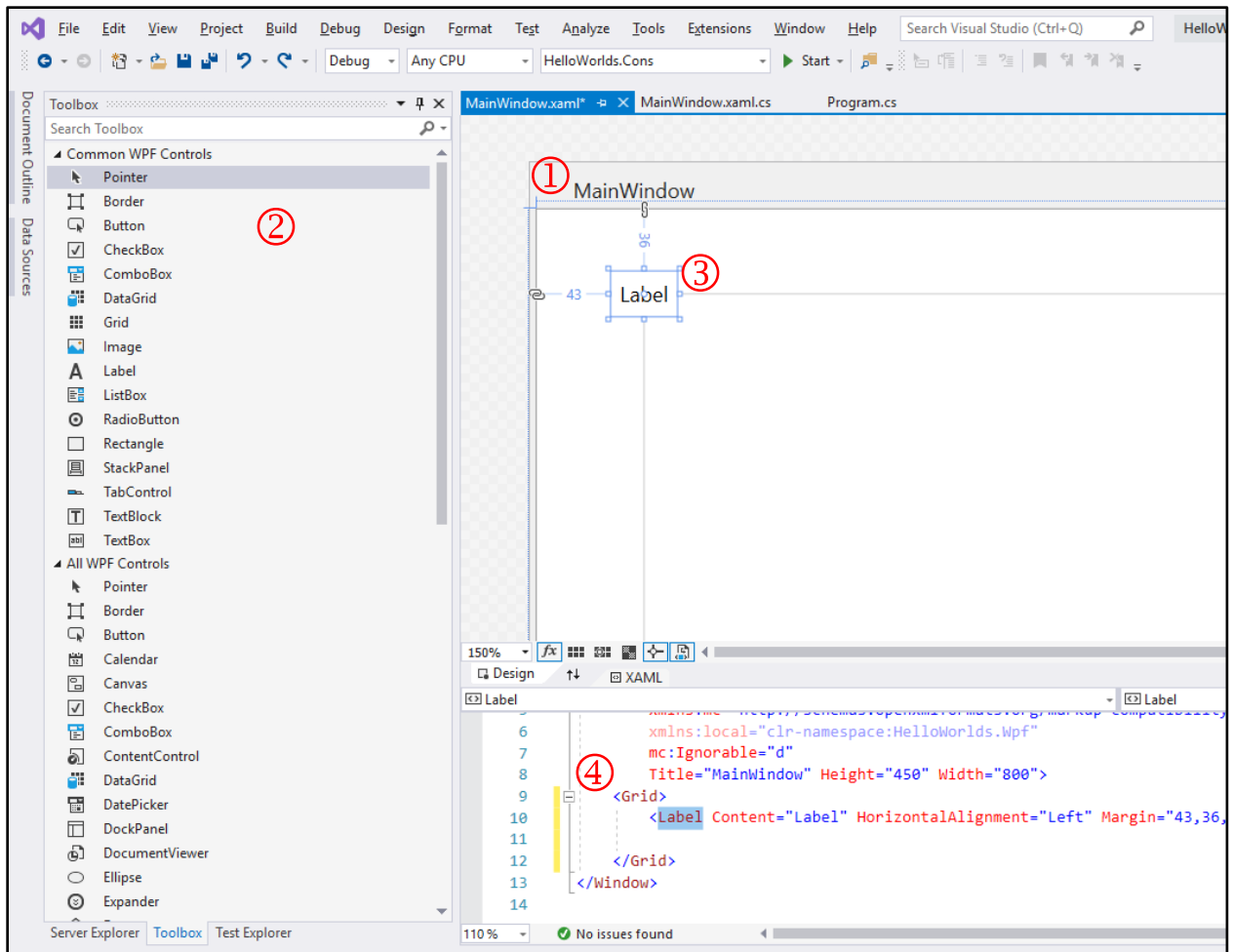
- Projectnaam: **Prb.Inleiding.HelloWorlds.Wpf**



Nadat Visual Studio het projectsjabloon genereert, zie je een grafische omgeving van een typische Windows desktopapplicatie. Het hoofdscherm, `MainWindow` genaamd, wordt geopend in een nieuw Visual Studio venster. Je kan dit venster inrichten zoals je zelf wil: door middel van tekstvakken, knoppen, lijsten en vele andere **controls** stel je de gebruiker in staat om de applicatie te besturen. Je vindt deze controls in de **Toolbox**, wanneer het `MainWindow` venster zichtbaar hebt gemaakt in Visual Studio.

4. Open de **Toolbox** via het menu **View → Toolbox**.
5. Sleep een **Label** control vanuit de Toolbox naar de layout van het `MainWindow`.

Je zou nu de volgende situatie moeten hebben:



- ① Het MainWindow venster kan je openen via de Solution Explorer en stelt je in staat om jouw GUI te ontwerpen met behulp van **Controls**.
- ② De **Toolbox** bevat alle WPF Controls waarmee je jouw ontwerp kan inrichten.
- ③ Wanneer je een **Label** control op het MainWindow sleept, dan kan je allerlei eigenschappen veranderen, zoals de plaats, de grootte en de inhoud ervan. Probeer het Label gerust eens te verplaatsen en te wijzigen in grootte.
- ④ Het onderste scherm toont de XAML-code die de GUI definieert. Ervaren WPF-programmeurs kunnen het scherm opbouwen door middel van deze code in plaats van het klik-en-sleep systeem erboven.

4.2.2 CONTROL EIGENSCHAPPEN WIJZIGEN

Je wenst nu de inhoud van het Label te wijzigen.

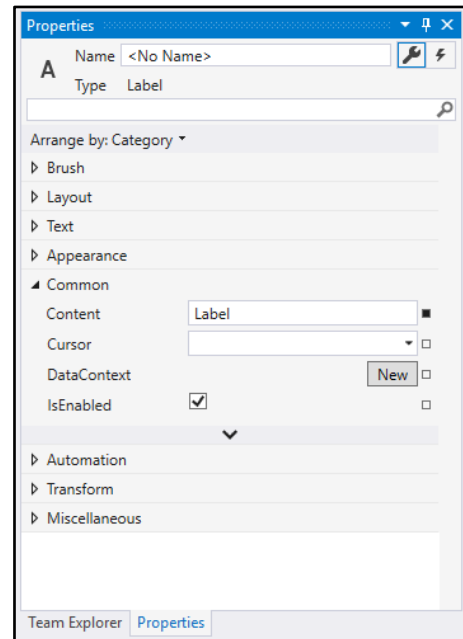
6. Selecteer het Label.
7. Open het **Properties** (eigenschappen) venster via **View → Properties Window** of via **F4**.
Dit venster staat standaard rechts onder in je Visual Studio omgeving.

Via het **Properties Window** kan je talloze eigenschappen van de geselecteerde Control (of Window) instellen.

Het is niet de bedoeling om deze allemaal van buiten te kennen, maar het is belangrijk om er je weg in te vinden zodat je een idee hebt wat er allemaal kan worden ingesteld.

8. Zoek de eigenschap genaamd **Content** en wijzig het naar "Hello World!".

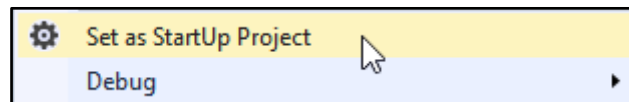
Merk op dat ook de tekst automatisch gewijzigd wordt in het MainWindow venster, en dat eigenschap eveneens in de XAML code wordt aangepast.



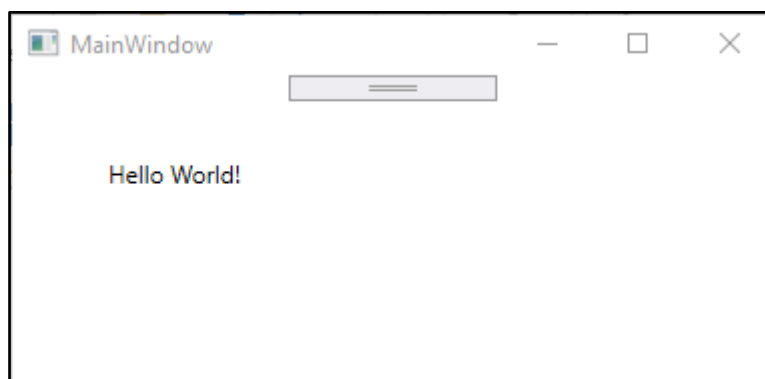
Je bent nu klaar om je GUI applicatie te testen.

Omdat de solution meerdere projecten bevat, moet je nog kiezen welke applicatie er moet worden gestart bij een druk op de **F5** knop.

9. Stel HelloWorlds.Wpf in als **start up project** door een rechtsklik op het project in de Solution Explorer. Kies in het contextmenu voor "**Set as Startup Project**"



10. Druk op **F5** om de applicatie uit te voeren.



11. **Sluit** de applicatie zodat je **verder kan werken** in Visual Studio.

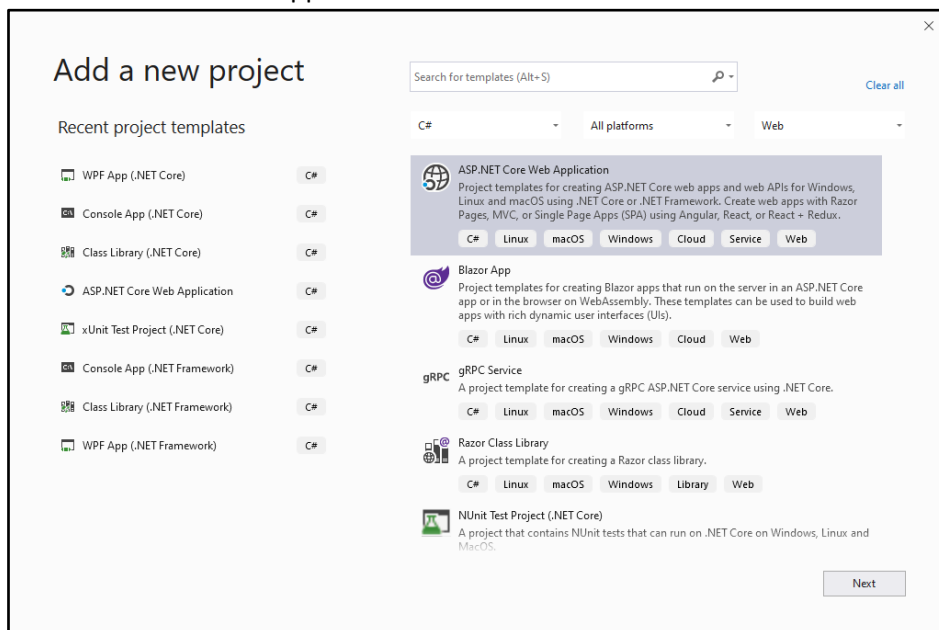
4.3 WEBAPPLICATIE

Als laatste voorbeeldapplicatie maak je een webapplicatie aan. Dit zijn applicaties die uitgevoerd worden door een Webserver. Wanneer een bezoeker een correcte URL intikt in zijn browser zal de webapplicatie de inhoud van de gevraagde webpagina terugsturen naar de browser.

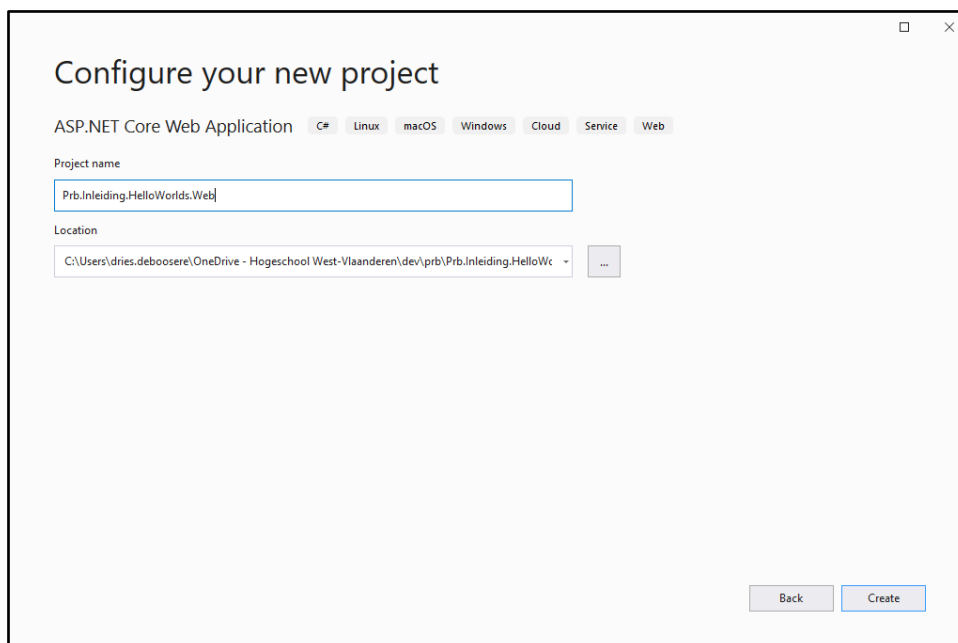
4.3.1 EEN WEBAPPLICATIE PROJECT MAKEN

1. Zorg dat je solution genaamd **HelloWorlds** geopend is in Visual Studio.
2. Rechtsklik op de solution in de Solution Explorer en kies **Add → New Project**
3. In het **New Project dialoogvenster** kies de volgende opties:

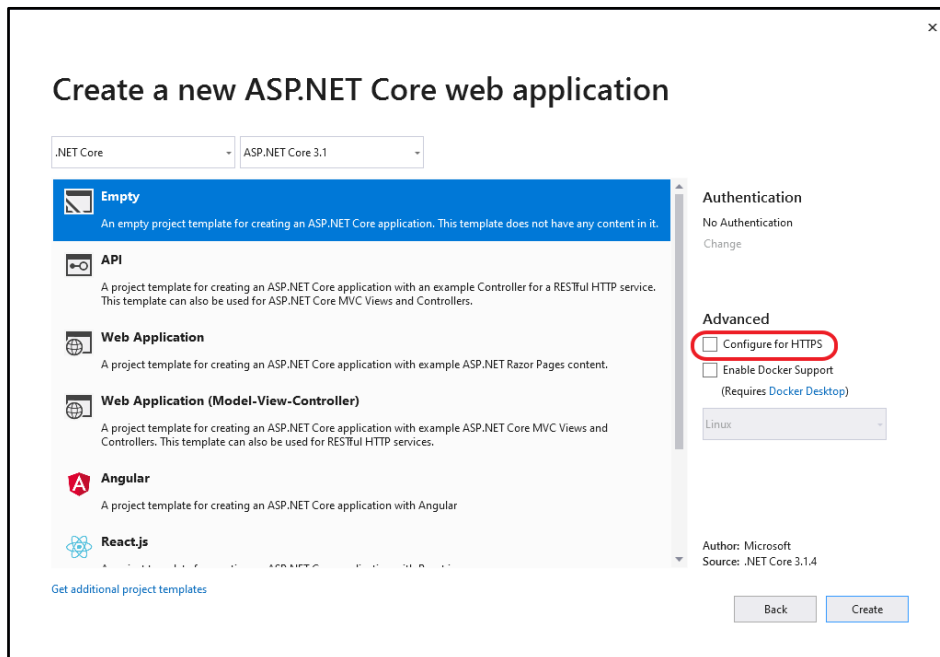
- Sjabloon: ASP.NET Core Web Application



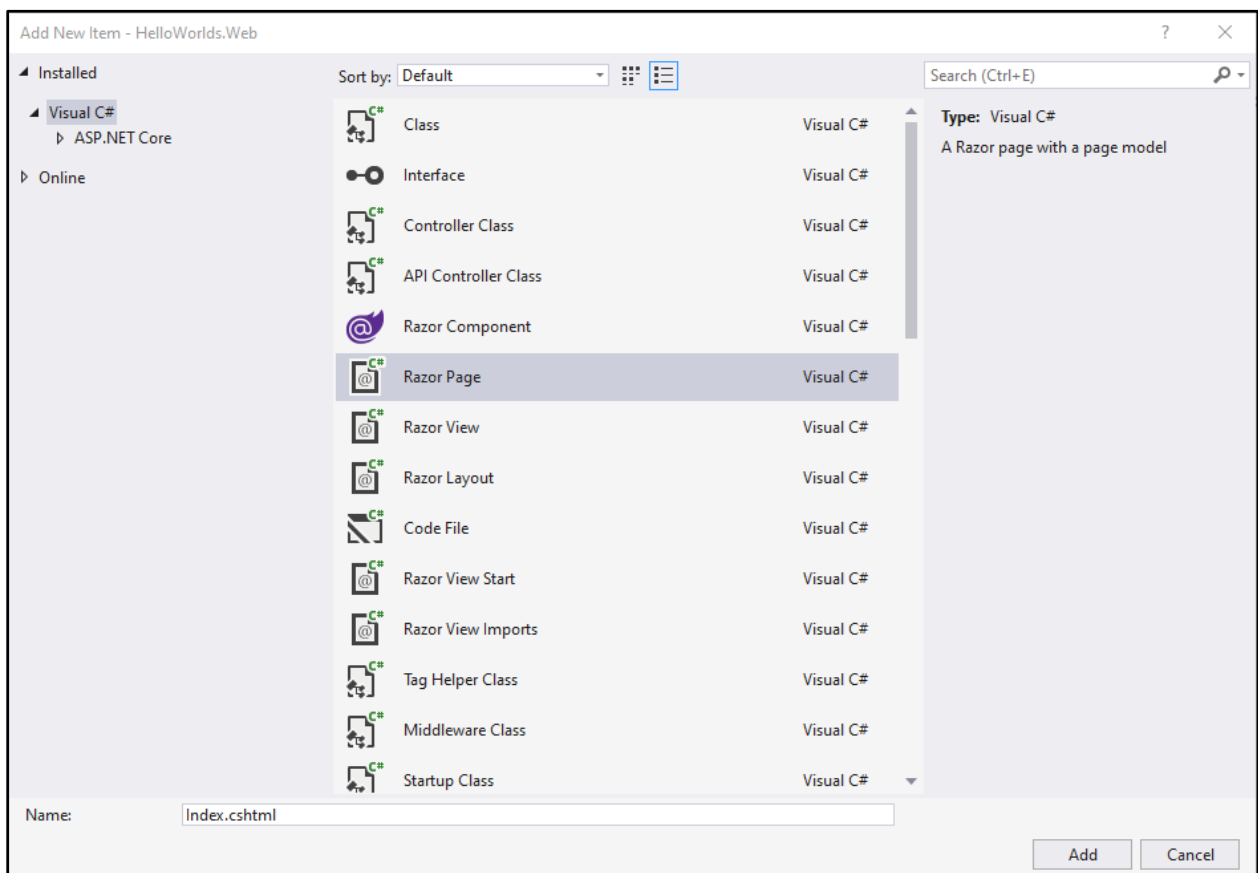
- Projectnaam: **Prb.Inleiding.HelloWorlds.Web**



4. In het scherm dat hierop volgt kies je voor een “Empty” sjabloon.
5. Vink **Configure for HTTPS** uit



6. Rechtsklik op het project in de Solution Explorer en kies **Add → New Item**
7. Kies voor een **Razor Page** en geef het de naam **Index.cshtml**.



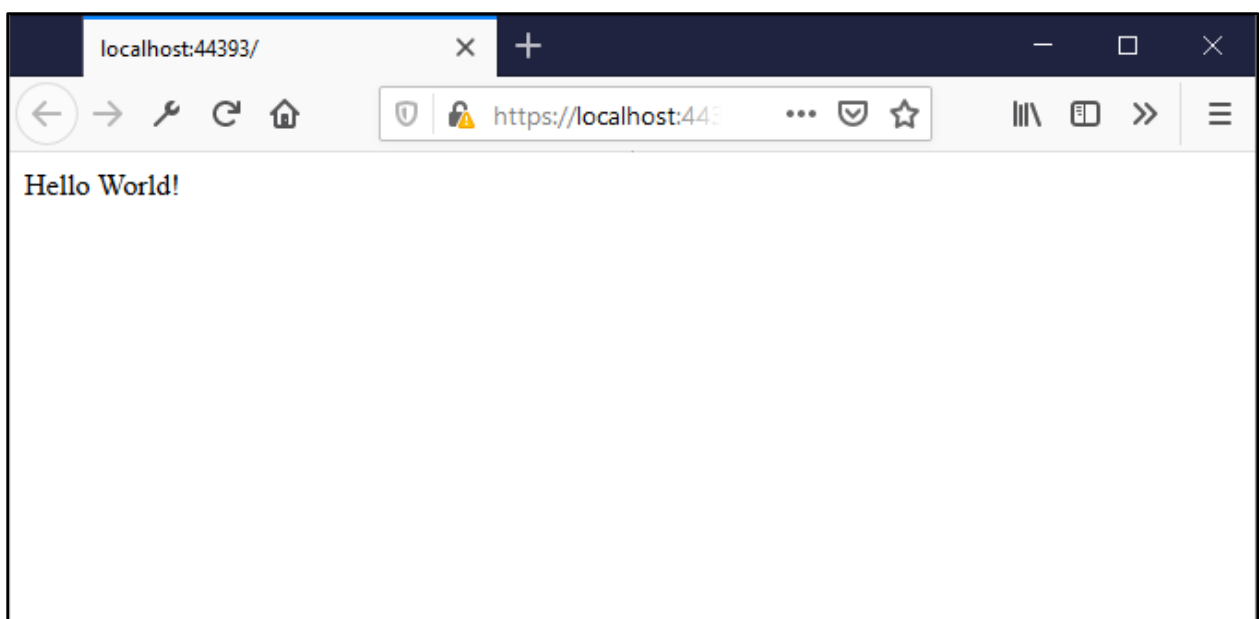
8. Open Index.cshtml (*niet verwarren met Index.cshtml.cs!*) En voeg tussen een p-element de tekst “Hello World!” toe.

De html code ziet er nu als volgt uit:

```
@page
@model Prb.Inleiding.HelloWorlds.Web.IndexModel
@{
}

<p>Hello World!</p>
```

9. Stel HelloWorlds.Web in als startup project via een rechtsklik in de Solution Explorer en de optie **Set as Startup Project**.
10. Voer de applicatie uit met **F5** of de startknop bovenaan.
De inhoud van je webpagina wordt nu getoond in je standaardbrowser.




11. Stop de applicatie of sluit de Browser om verder te kunnen werken in Visual Studio.



Code repository

De volledige broncode van deze applicatie is te vinden op

 `git clone` <https://github.com/howest-gp-prb/cu-inleiding-hello-worlds.git>

Het werken met git en GitHub wordt verder uitgelegd in de module Continuous Integration Basics.

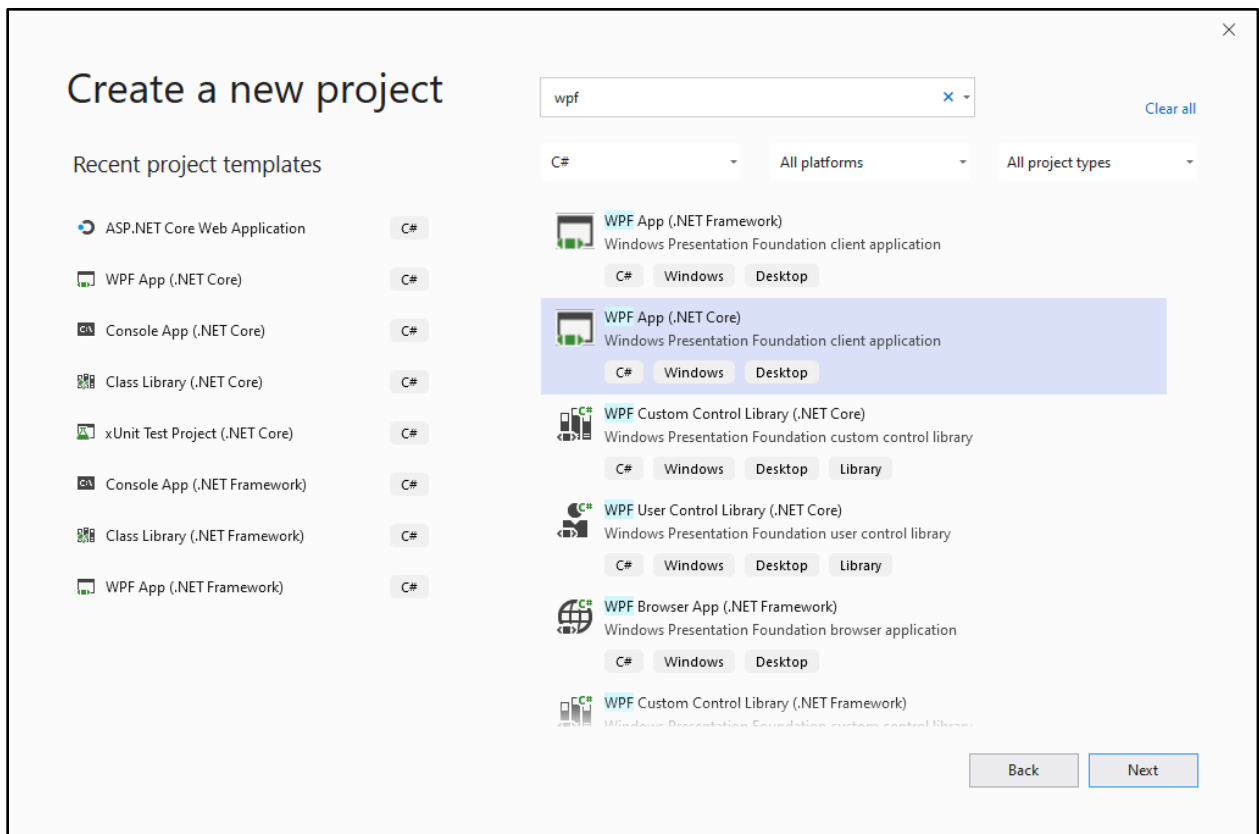
5 EEN EERSTE WPF APPLICATION

In deze syllabus leer je voornamelijk programmeren met C# aan de hand van WPF applicaties. Om WPF verder te verkennen maak je nu een uitgebreidere, interactieve GUI-applicatie.

In deze applicatie wordt het MainWindow het enige venster dat de gebruiker te zien krijgt. Het zal een tekstvak en een knop bevatten. De gebruiker moet zijn naam intikken in het tekstvak en op de knop klikken. De applicatie begroet de gebruiker dan met de ingevoerde naam.

5.1 PROJECT AANMAKEN

1. Start Visual Studio en kies **Create New Project**
2. In het **New Project** dialoogvenster kies je de volgende zaken:
 - Runtime: **.NET Core**
 - Sjabloon: **WPF App (.NET Core)**
 - Project name: **Prb.Inleiding.FirstApp.Wpf**
 - Solution name: **Prb.Inleiding.FirstApp**



×

Configure your new project

WPF App (.NET Core) C# Windows Desktop

Project name

Location

Solution name ⓘ

☐ Place solution and project in the same directory

Back Create

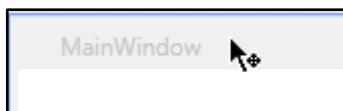
De namen voor het project en de solution zijn volledige vrij te kiezen. Door de naam van de solution algemener te maken, zorg je voor een meer logische indeling indien je zou beslissen om meerdere projecten aan de solution toe te voegen. De punten in de naam zorgen ervoor dat je code automatisch wordt ingedeeld in overeenkomstige namespaces, waarover later meer.

5.2 GUI ELEMENTEN TOEVOEGEN EN BEWERKEN

Voor je gaat programmeren verzorg je eerst de lay-out van je GUI-applicatie. Het venster dat getoond wordt bij het uitvoeren van een WPF applicatie is standaard de **MainWindow**.

1. Open **MainWindow.xaml** door erop te dubbelklikken in de Solution Explorer.
2. Wijzig de afmetingen van het Window met de muis:

- Selecteer het Window door een enkele klik in de titelbalk



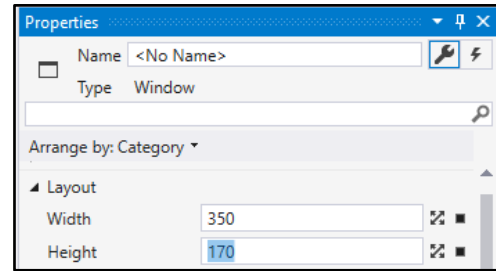
- Sleep het vierkante handvatje in de hoek rechtsonder naar de gewenste positie.



- Zorg voor een breedte van 300 pixels en een hoogte van 160 pixels.

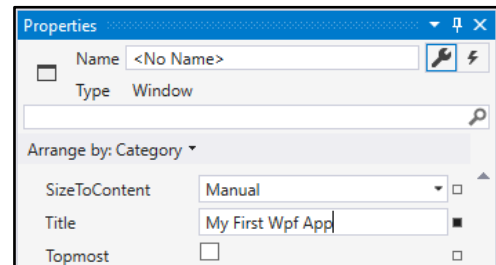
3. Wijzig de afmetingen van het Window via het **Properties** scherm:

- Selecteer het Window door een enkele klik in de titelbalk
- Zoek in het **properties** scherm, in de categorie **Layout**, de eigenschappen **Width** en **Height**.
- Tik de waarden in, Width: **350**, Height: **170** en druk enter om te bevestigen.

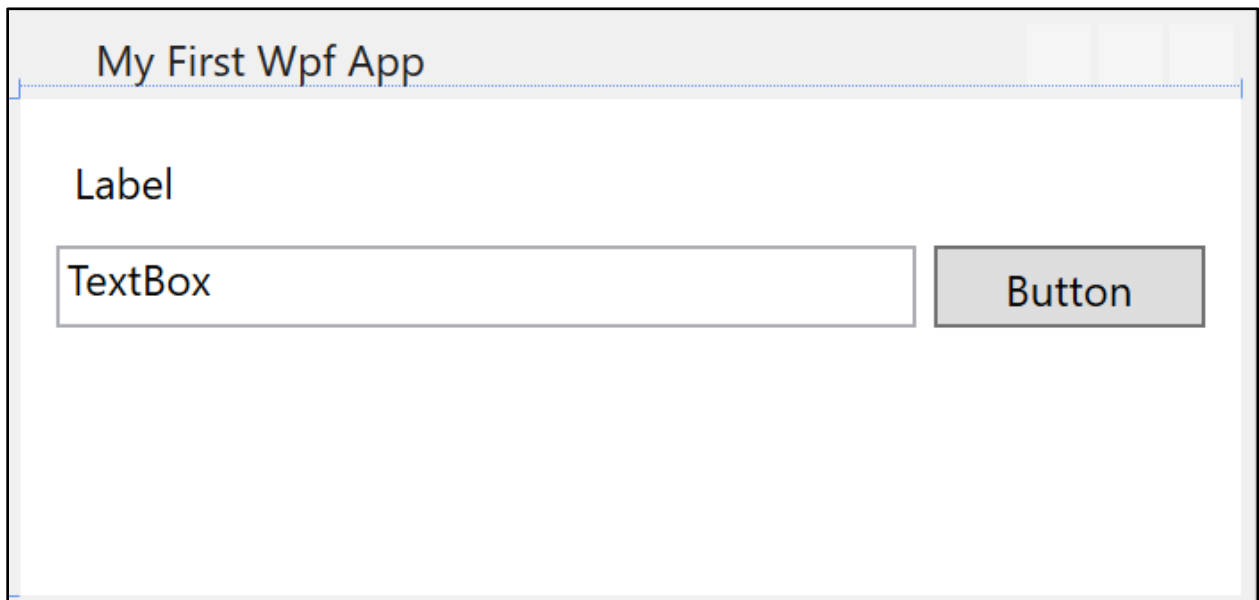


4. Wijzig de titel van het Window via het **Properties** scherm. De titel is zichtbaar in de titelbalk van de applicatie.

- Selecteer het Window door een enkele klik in de titelbalk
- Zoek in het **properties** scherm, in de categorie **Common**, de eigenschap **Title**.
- Geef het venster de titel “My First Wpf App” en druk **enter** om te bevestigen.

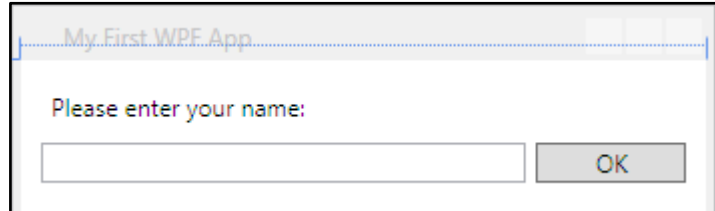


5. Plaats nu vanuit het **Toolbox** venster de volgende controls op het MainWindow: **Label**, **TextBox** en **Button**. Positioneer ze met behulp van de visuele hints om de elementen precies te plaatsen waar jij dat wil.



6. Wijzig de volgende eigenschappen van de Label, de TextBox en de Button door het betreffende element eerst te selecteren met een enkele klik en vervolgens de eigenschappen in het Properties venster aan te passen:

- **Label**
 - Content: "Voer je naam in"
- **TextBox**
 - Text: (blanco maken)
 - Name: txtName
- **Button**
 - Content: "OK"
 - Name: btnOk



Door elementen een **Name** te geven, kan je deze aanspreken vanuit C# code. Het is een goede gewoonte om elementen steeds een Name te geven zodra je ze hebt toegevoegd.

Voor de naamgeving van visuele elementen wordt er meestal de zogenaamde Hongaarse notatie gebruikt, waarbij het type van dat element de prefix vorm. Voor een TextBox is dat bijvoorbeeld **txt** en voor een Button is dat bijvoorbeeld **btn**. Wij gebruiken de volgende prefixen:

- Label: **lbl**
- TextBox: **txt**
- TextBlock: **tbk**
- ComboBox: **cmb**
- Button: **btn**
- ListBox: **lst**
- StackPanel: **stp**
- Grid: **grd**

Door deze prefixen kunnen we gemakkelijk de namen van controls terugvinden via de intellisense. Door een prefix in te tikken krijgen we de lijst met de controls van een bepaalde soort.

Merk op dat alles wat je in het **Design** scherm en het Properties scherm hebt gedaan een rechtstreekse weerslag heeft in de XAML code van de MainWindow:

```
<Grid>
  <Label Content="Please enter your name:" HorizontalAlignment="Left" Margin="10,10,0,0"
    VerticalAlignment="Top"/>
  <Button x:Name="btnOk" Content="OK" HorizontalAlignment="Left" Margin="257,41,0,0"
    VerticalAlignment="Top" Width="75"/>
  <TextBox x:Name="txtName" HorizontalAlignment="Left" Height="20" Margin="10,41,0,0"
    TextWrapping="Wrap" VerticalAlignment="Top" Width="242"/>
</Grid>
```

Ervaren WPF programmeurs kunnen de lay-out dus ook rechtstreeks als XAML code intikken.

5.3 EVENTS AFHANDELEN

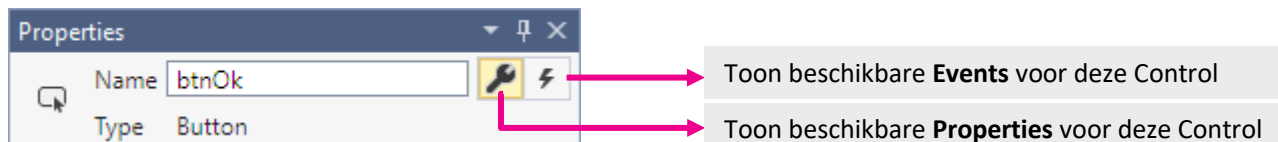
Alle WPF controls, inclusief `Window` hebben zogenaamde **Events** of gebeurtenissen waarop je kan inschrijven zodat jouw programmalogica wordt uitgevoerd.

Een `Window` heeft bijvoorbeeld een **Loaded** event, dat zich voordoet wanneer het venster volledig is ingeladen. Een `TextBox` heeft een **TextChanged** event, dat zich voordoet wanneer de gebruiker tekst heeft gewijzigd in het tekstvak en een `Button` heeft een **Click** event, dat zich voordoet wanneer de gebruiker op de knop klikt.

Om je eigen programmacode uit te voeren wanneer zo'n event zich voordoet moet je een stukje code, in de vorm van een **methode** koppelen aan de betreffende control. Die code wordt dan uitgevoerd zodra het event zich voordoet.

Je wenst nu wat code uit te voeren wanneer de gebruiker op de knop klikt. Je gaat nu als volgt te werk:

1. Selecteer de knop genaamd **btnOk** in de Designer met een enkele klik.
2. Ga naar het **Properties** venster en zoek de knop met bliksem symbool.



3. Klik op de knop met het **bliksempje** om alle beschikbare Events van de Button control weer te geven.
4. Zoek het Event genaamd **Click** en dubbelklik met de muis in het lege tekstvak ernaast.



Eens je dat hebt gedaan gebeuren er twee zaken:

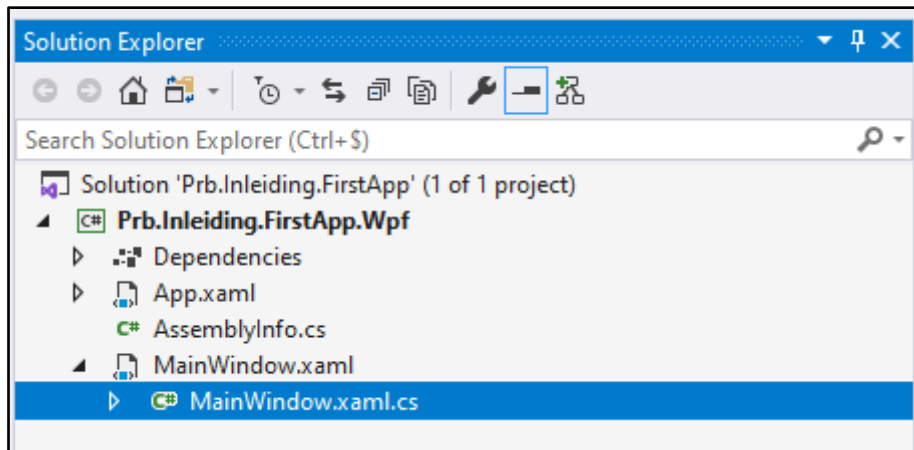
- Er wordt een methode gegenereerd, genaamd `btnOk_Click`
- De methode wordt gekoppeld aan het `Click` event van de knop.

Visual Studio opent nu automatisch het bestand genaamd **MainWindow.xaml.cs**, een C# codebestand waar je de (lege) methode ziet staan:

```
private void btnOk_Click(object sender, RoutedEventArgs e)
{
}
}
```

Het codebestand, genaamd `MainWindow.xaml.cs` wordt ook wel de **CodeBehind** genoemd. Het bevat de programmacode die rechtstreeks aan de visuele layout van het `MainWindow` is gekoppeld.

Je kan het bestand steeds terugvinden in de Solution Explorer, wanneer je de MainWindow.xaml node open tikt. Dat zie je in het volgende screenshot.

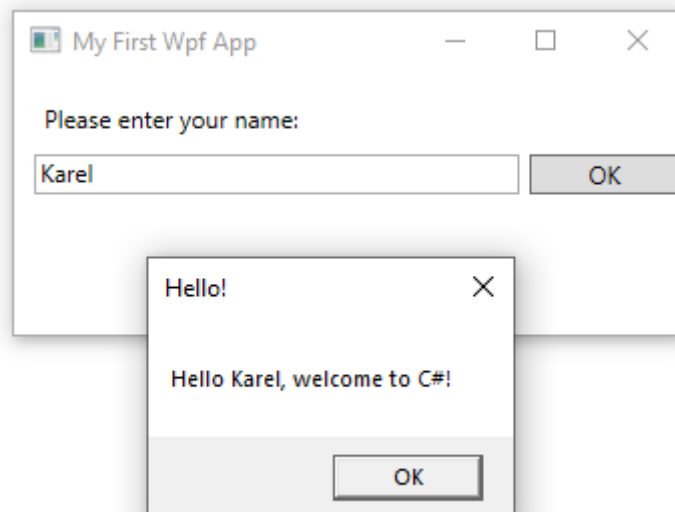


Je wijzigt nu de methode genaamd `btnOk_Click` als volgt:

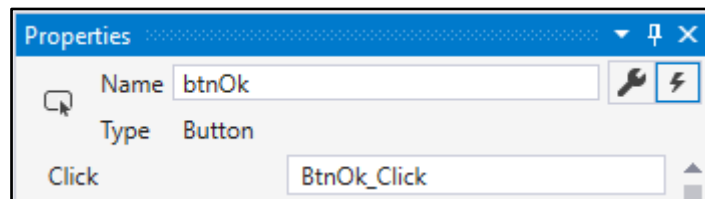
```
//Event Handler method for the Button Click event
private void btnOk_Click(object sender, RoutedEventArgs e)
{
    //display a messagebox (popup dialog) with the user's name
    MessageBox.Show("Hello " + txtName.Text + ", welcome to C#", "Hello!");
}
```

De toegevoegde code bevat een aantal commentaarlijnen om duidelijk te maken wat we doen, en een instructie die een `MessageBox` oproept. De inhoud van de `MessageBox` bevat wat tekst die onder meer de ingevoerde naam van de gebruiker toont.

5. Voer de applicatie uit met **F5** en test de functionaliteit.



Een dergelijke methode, gekoppeld aan een event, wordt ook wel een Event Handler genoemd. In het Design venster, MainWindow.xaml zie je nu ook de naam van die methode staan naast het Event in het Properties venster.



Je hebt nu een eerste basis applicatie in WPF gemaakt die gebruikersinteractie toestaat door middel van een GUI. De meeste voorbeelden en opdrachten in deze syllabus zullen deze basishandelingen vereisen. Zorg dus dat je dit goed onder de knie hebt.



Code repository

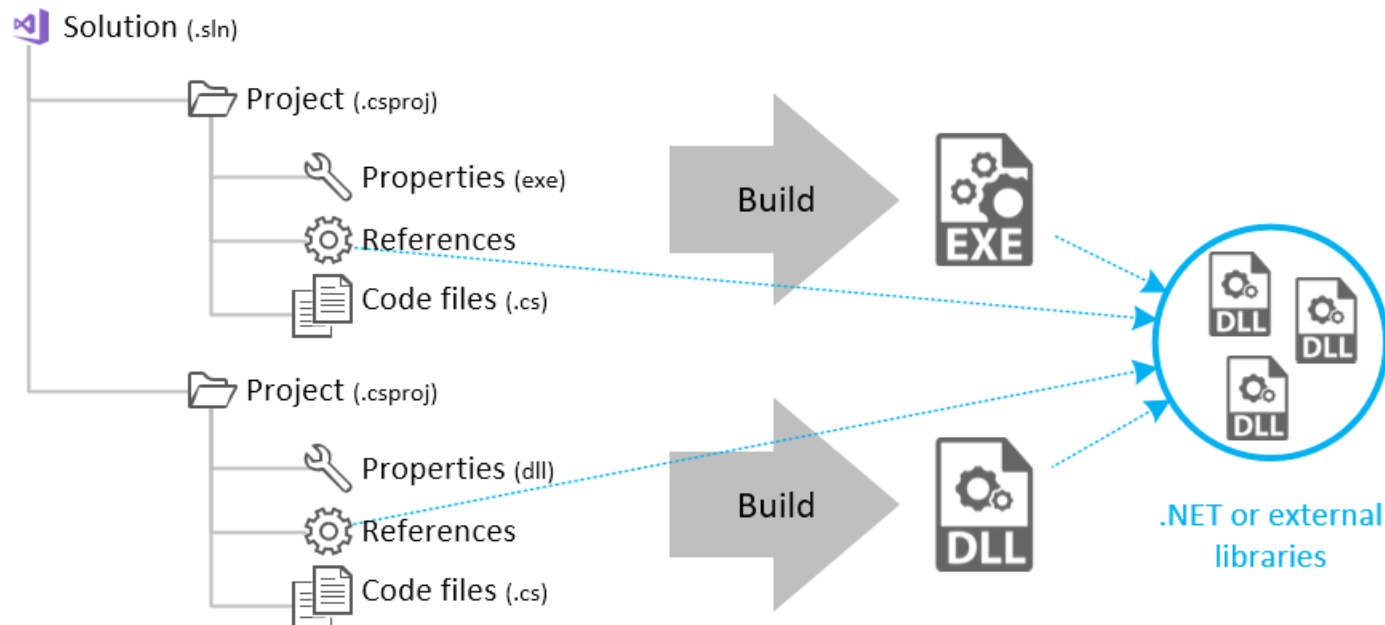
De volledige broncode van deze applicatie is te vinden op

 `git clone` <https://github.com/howest-gp-prb/cu-inleiding-wpf-basics.git>

6 WPF REFERENTIEMATERIAAL

6.1 SOLUTIONS EN PROJECTEN

Om de relaties tussen Solutions, Projecten, assemblies en het .NET framework te herhalen kan je het volgende schema beschouwen:



6.2 REFERENTIEMATERIAAL

Deze syllabus bevat geen volledig overzicht van alle WPF Controls, hun properties en events. Het is ook zinloos om deze allemaal uit het hoofd te kennen. Een goede programmeur weet zich te behelpen met opzoekwerk en experiment. De onderstaande bronnen vormen hier een goed startpunt voor, die je gedurende deze module kan blijven benutten.



Meer informatie

- wpf-tutorial.com
Een handleiding voor de meeste gebruikte WPF Controls
- wpftutorial.net
Een uitgebreide handleiding voor WPF
- [MSDN: System.Windows.Controls](https://docs.microsoft.com/en-us/dotnet/framework/wpf/controls/system-windows-controls-overview)
Microsoft's overzicht van alle WPF Controls

Wie op een vlotte manier WPF layouts wenst te maken, moet zeker een kijkje nemen in het tutorial over WPF Panels. Dat is te vinden op <http://www.wpf-tutorial.com/panels/introduction-to-wpf-panels>

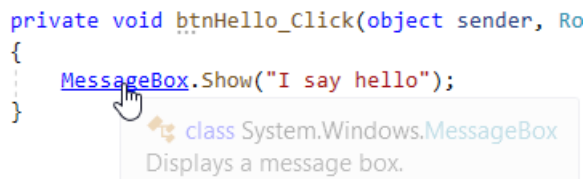
7 DOCUMENTATIE OVER C#

De mogelijkheden met C# zijn schier oneindig. Leuk, maar waar veel mogelijkheden zijn, verliezen we soms het overzicht en hebben we hulp nodig om door de bomen het bos te zien.

7.1 IN VISUAL STUDIO

Naast de help-functie (Ctrl + F1) die ons naar een algemene documentatie-pagina op het internet brengt, kun je ook binnen Visual Studio op onderzoek gaan.

Om wat informatie te krijgen (structuur van de klasse, waarover meer in latere hoofdstukken) over een object, kun je op de naam van het object klikken met de control-toets ingedrukt.



```
private void btnHello_Click(object sender, Ro
{
    MessageBox.Show("I say hello");
}
```

class System.Windows.MessageBox
Displays a message box.

Dit brengt je naar een pagina met de kenmerken van de aangeklikte klasse.

Assembly PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856

```
namespace System.Windows
{
    ...public sealed class MessageBox
    {
        ...public static MessageBoxResult Show(string messageBoxText);
        ...public static MessageBoxResult Show(string messageBoxText, string caption);
        ...public static MessageBoxResult Show(string messageBoxText, string caption, Mes
        ...public static MessageBoxResult Show(string messageBoxText, string caption, Mes
        ...public static MessageBoxResult Show(string messageBoxText, string caption, Mes
```

In het begin zal deze informatie weinig betekenis hebben voor jou, maar naarmate je meer inzicht krijgt in de structuur van .Net kan dit interessant worden.

