C# Essentials - WPF

Inhoudsopgave

1. Introductie

1.1. Van tekst naar vensters

Tot nu toe werkte je met consoletoepassingen. Je gaf opdrachten via code en kreeg de uitvoer in een zwart tekstvenster te zien. Maar de meeste programma's werken niet zo. Denk aan apps zoals Spotify, Microsoft Word of een eenvoudige rekenmachine - die hebben allemaal een grafische gebruikersinterface, of GUI (**spreek uit: "gioewie"**).

Een GUI bestaat uit knoppen, tekstvakken, menu's, schuifbalken, afbeeldingen… Met andere woorden: alles wat je ziet en aanklikt.

Daarom maken we vanaf nu de overstap naar **WPF**.

1.2. Wat is WPF?

WPF staat voor **Windows Presentation Foundation**. Het is een technologie van Microsoft waarmee je desktoptoepassingen kan bouwen die er professioneel uitzien. Die toepassingen bestaan uit vensters met knoppen, tekstvelden, lijsten, sliders en nog veel meer.

WPF is dus:

- speciaal ontworpen voor **desktopsoftware op Windows**
- een manier om **grafische toepassingen** te bouwen
- gebaseerd op een combinatie van **XAML** en **C#**

1.3. WPF is event-driven

Tot nu toe werkte je in een consoletoepassing. Daar werd je programma altijd stap voor stap uitgevoerd van boven naar beneden. Jij als programmeur bepaalde de volledige volgorde van wat er moest gebeuren.

Een WPF-toepassing werkt helemaal anders: je schrijft niet één rechte lijn van code, maar een verzameling van kleine stukjes code die pas uitgevoerd worden wanneer de gebruiker iets doet.

We noemen dit een **event-driven** applicatie.

1.3.1. Wat is event-driven?

Event-driven betekent: je programma wacht op gebeurtenissen (**events**) en voert pas dan iets uit. Zo'n gebeurtenis kan zijn:

- de gebruiker klikt op een knop
- de gebruiker selecteert iets uit een lijst
- de gebruiker beweegt de muis over een afbeelding

Je programmeert dan een **eventhandler**. Dat is een methode die automatisch wordt uitgevoerd als die gebeurtenis plaatsvindt.

1.3.2. Vergelijking

Consoleapplicatie	WPF-toepassing
Code wordt stap voor stap uitgevoerd van begin tot einde	De applicatie blijft draaien en reageert op gebeurtenissen
Je bepaalt zelf wat er eerst, dan en daarna gebeurt	De gebruiker bepaalt het verloop door interactie
Geen visuele interface, enkel tekst	Visuele interface met knoppen, lijsten, velden…

- Een consoletoepassing is zoals een kookrecept dat je stap voor stap volgt."
- Een WPF-toepassing is zoals een koffiemachine: die doet niets totdat jij op een knop drukt.

1.4. XAML

1.4.1. Wat is XAML?

XAML staat voor **eXtensible Application Markup Language**. Het is een speciale opmaaktaal waarmee je het uitzicht van je vensters beschrijft.

In WPF gebruik je XAML om:

- vensters en schermen op te bouwen
- knoppen, tekstvakken en andere controls toe te voegen
- de eigenschappen van elke control visueel te beschrijven

Je kan een venster opbouwen door de juiste elementen en attributen te gebruiken.

1.4.2. Waarom XAML?

Je vraagt je misschien af: waarom niet gewoon alles in C# doen?

Je **kan** alles in C# schrijven, maar dat is niet zo handig. XAML maakt het makkelijker om:

- het **uitzicht** van je venster op een visuele manier te beschrijven;
- **design en functionaliteit** van elkaar te scheiden;
- samen te werken: de ene persoon kan aan het ontwerp werken (XAML), de andere aan de logica (C#);
- en het werkt perfect samen met de **Designer** in Visual Studio.

1.4.3. Structuur van XAML

XAML lijkt op HTML of XML. Elk element start met een tag en eindigt met een sluit-tag:

```
<Button>Save</Button>
```

Als je de inhoud van een element als attribuut kan meegeven is een sluit-tag niet verplicht, je spreekt dan van een **zeflsluitende tag**:

```
<Button Content="Save" />
```

1.4.4. Geneste elementen

Je kan elementen ook **nesten** binnen elkaar. Dat betekent dat je een control plaatst binnen een container (zoals een `Grid` of `StackPanel`):

```
<StackPanel>
    <TextBlock Text="Enter your name:" />
    <TextBox Width="200" />
</StackPanel>
```

In dit voorbeeld bevinden de `TextBlock` en `TextBox` zich binnen het `StackPanel`. De volgorde van de elementen in de XAML komt overeen met hoe ze in het venster getoond worden (van boven naar onder).

1.4.5. Element of attribuut?

Soms kan je kiezen:

- Wil je een eigenschap snel instellen? → gebruik een attribuut
- Wil je meerdere instellingen of complexere inhoud? → gebruik een genest element

Bijvoorbeeld voor kleuren of afbeeldingen gebruik je vaak geneste notatie:

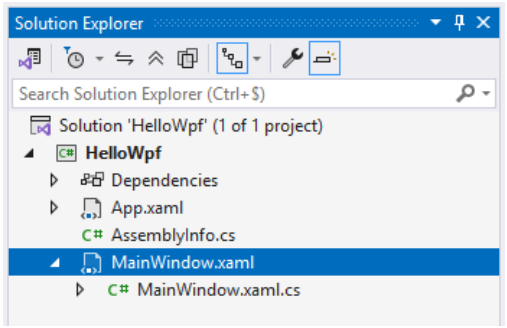
```
<Button>
    <Button.Background>
        <SolidColorBrush Color="LightBlue" />
    </Button.Background>
</Button>
```

1.4.6. Samenvatting

Term	Betekenis
Tag	Een XAML-element zoals `<Button>` of `<TextBox>`
Attribuut	Instelling binnen een tag (vb. `Width="150"`)
Nesting	Een element dat een ander element bevat

Term	Betekenis
Zelfsluitende tag	Een element dat geen inhoud heeft (vb. `<TextBlock />`)

1.5. En wat met C#?



Elk venster in een WPF-project heeft twee delen:

1. het XAML-bestand met de visuele opbouw (bijvoorbeeld `MainWindow.xaml`)
2. een gekoppeld C#-bestand met de logica (bijvoorbeeld `MainWindow.xaml.cs`)

Dit C#-bestand noemen we het **code-behind**-bestand.

In dat bestand:

- schrijf je methodes voor events (zoals klikken op een knop)
- krijg je toegang tot alle elementen met een naam (`x:Name`)
- pas je eigenschappen aan of lees je waarden uit

1.5.1. Voorbeeld

Als je in XAML schrijft:

```
<Button x:Name="saveButton" Content="Save" Click="saveButton_Click" />
```

Dan staat in het gekoppelde C#-bestand (`MainWindow.xaml.cs`):

```
private void saveButton_Click(object sender, RoutedEventArgs e)
{
    // Hier schrijf je de code die moet worden uitgevoerd bij een klik
}
```

Je kan het code-behind-bestand openen door in de Solution Explorer op het kleine pijltje naast de XAML-bestandsnaam te klikken.

1.6. Wat komt hierna?

In het volgende hoofdstuk tonen we hoe je een WPF-project opstart in Visual Studio. Je zal zien dat

Visual Studio automatisch een `.xaml`-bestand aanmaakt voor het venster en daar een bijhorende `.cs`-bestand aan koppelt.

Daarna leren we hoe je zelf knoppen, labels en tekstvakken toevoegt aan je venster en hoe je deze kan laten reageren op muiskliks of toetsen.

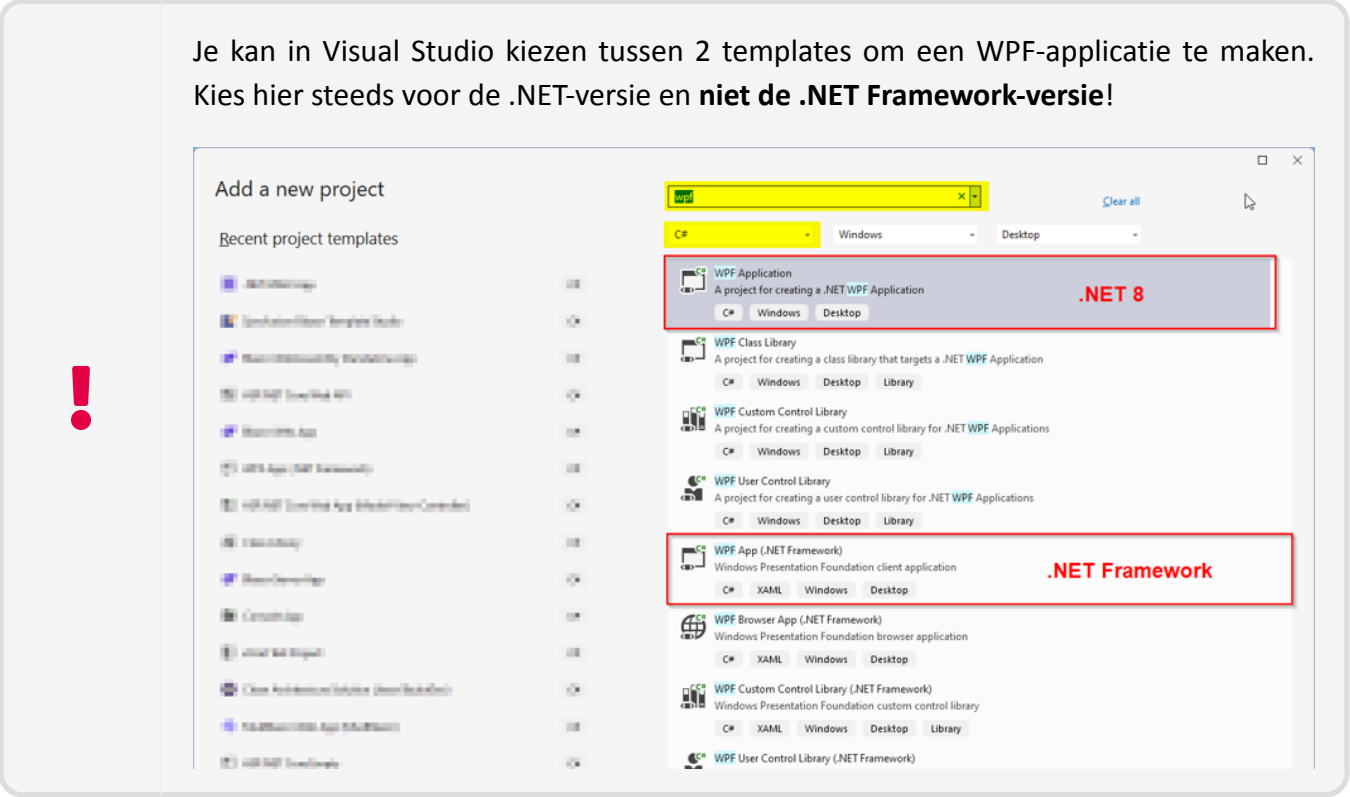
2. Visual Studio

2.1. Een nieuw WPF-project aanmaken

Om een grafische toepassing te bouwen met WPF, moet je een nieuw project aanmaken in Visual Studio. Volg deze stappen:

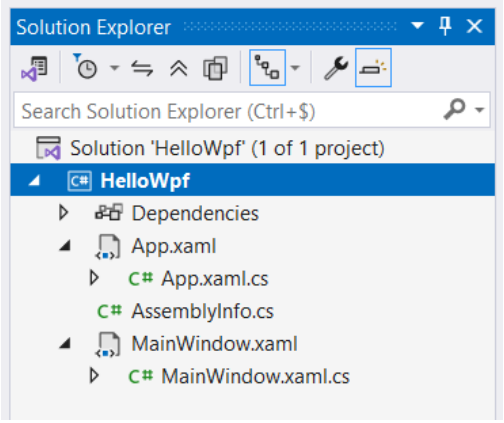
1. Open Visual Studio.
2. Klik op 'Create a new project'.
3. Zoek naar 'WPF Application' en selecteer dit type.
4. Klik op 'Next'.
5. Geef je project een naam, bijvoorbeeld `HelloWpf` en kies de gewenste locatie.
6. Selecteer .NET8.0 als framework en klik op 'Create'

Visual Studio maakt nu een aantal bestanden aan die samen je applicatie vormen.



Je kan in Visual Studio kiezen tussen 2 templates om een WPF-applicatie te maken. Kies hier steeds voor de .NET-versie en **niet de .NET Framework-versie**!

2.2. Overzicht van de projectstructuur



Na het aanmaken van een WPF-project zie je verschillende onderdelen in de Solution Explorer:

- `MainWindow.xaml`: Dit bestand beschrijft het uitzicht van je venster in XAML. Je zal hier de structuur van je UI tekenen.
- `MainWindow.xaml.cs`: Hier komt de bijhorende C#-code. In dit bestand schrijf je wat er moet gebeuren wanneer de gebruiker op een knop klikt of een veld invult.
- `App.xaml` en `App.xaml.cs`: Dit zijn algemene instellingen van je applicatie. Voorlopig hoef je hier nog niets in aan te passen.

2.3. De werkomgeving van Visual Studio

Wanneer je het project opent, zie je meestal twee vensters naast elkaar:

- Links: het XAML-bestand (`MainWindow.xaml`) met bovenaan twee tabjes:
  - **Design**: toont het venster zoals het eruit zal zien
  - **XAML**: toont de bijhorende code
- Rechts: de **Solution Explorer**, waarin je alle bestanden van het project terugvindt

Daarnaast zijn er nog enkele belangrijke vensters:

2.3.1. Toolbox

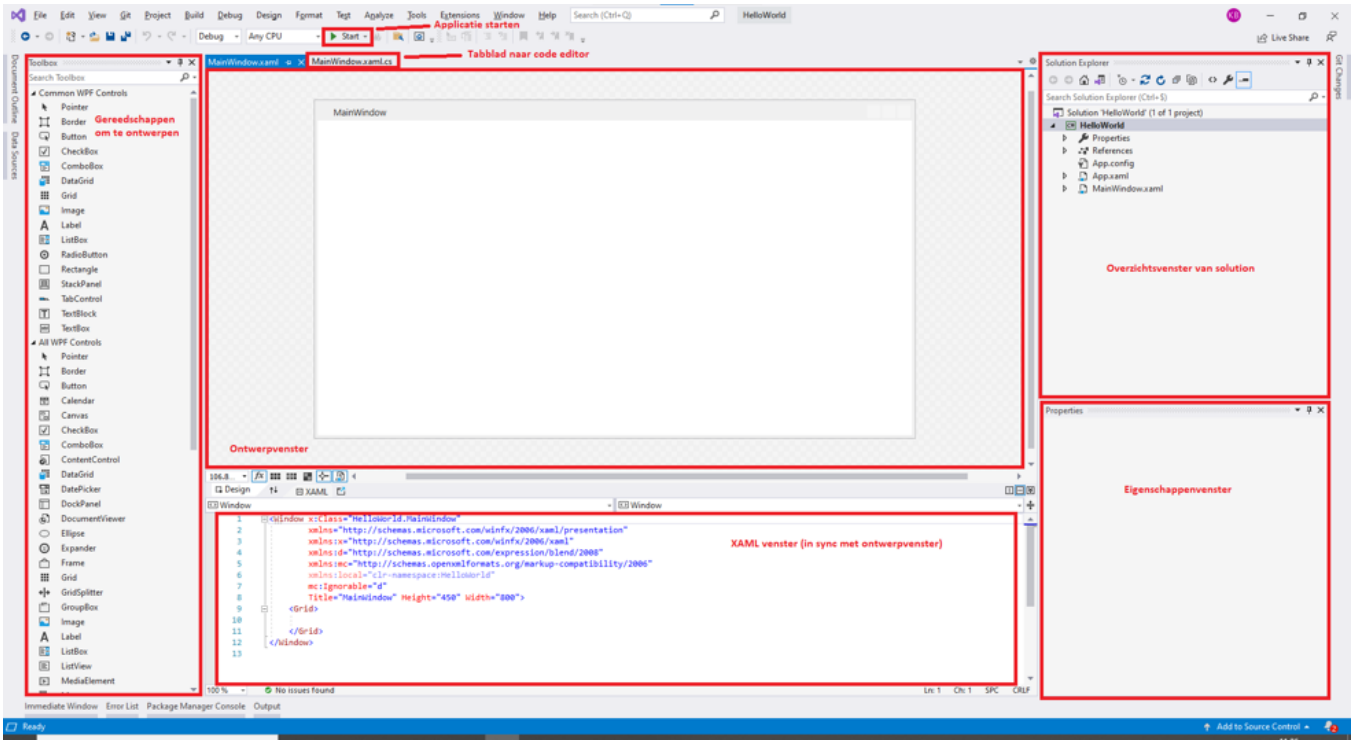
De Toolbox bevindt zich meestal links of linksboven. Hierin vind je alle WPF-controls zoals `Button`, `Label`, `TextBox`, `Slider`, enzovoort. Je kan een control slepen naar het venster, of zelf de XAML-code typen.

2.3.2. Properties

Wanneer je een control selecteert (bv. een knop), verschijnt het Properties-venster. Hierin kan je eigenschappen aanpassen zoals `Content`, `Name`, `Width`, `Height`, `FontSize`, `Background`…

2.3.3. Output & Error List

Onderin het scherm vind je de **Output** en **Error List**. Die geven info over wat er gebeurt wanneer je je project start, of welke fouten er eventueel zijn.



2.4. MainWindow

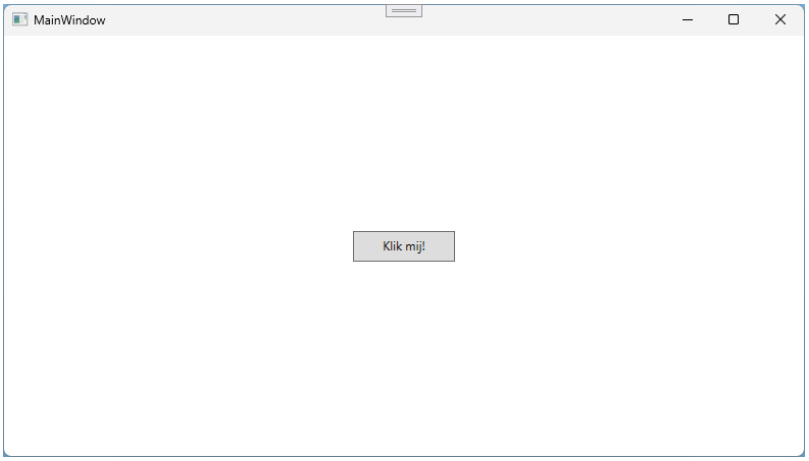
Je startproject bevat standaard een leeg venster met als XAML-code ongeveer dit:

```
<Window x:Class="HelloWpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:HelloWpf"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

Tussen de `<Grid>`-tags kan je beginnen te bouwen aan je UI. Probeer bijvoorbeeld eens deze extra regel toe te voegen:

```
<Button Content="Klik mij!" Width="100" Height="30" />
```

Sla op en klik op de groene startknop (of druk op `F5`) om je applicatie uit te voeren. Je ziet nu een venster met een knop.


MainWindow
Klik mij!

3. Controls

3.1. Wat is een control?

Een **control** is een element op je venster waar de gebruiker iets mee kan doen, of waarop informatie wordt getoond.

Voorbeelden van controls zijn:

- een knop die je kan aanklikken (`Button`)
- een tekstveld waarin je iets kan typen (`TextBox`)
- een label dat tekst toont (`Label`)
- een lijst met keuzemogelijkheden (`ComboBox`)

Je gebruikt deze bouwstenen om zelf je eigen vensters samen te stellen.

3.2. Hoe voeg je een control toe?

Er zijn verschillende manieren om een control aan je venster toe te voegen. Visual Studio maakt dit heel gemakkelijk:

3.2.1. Via de Toolbox

1. Open de **Toolbox** (meestal links).
2. Zoek de gewenste control (bijvoorbeeld `Button`).
3. Sleep de control naar je venster (of dubbelklik).

Wanneer je een control toevoegt via de toolbox wordt de bijhorende XAML-code automatisch toegevoegd!

3.2.2. Zelf XAML-code schrijven

Je kan ook rechtstreeks XAML-code typen. Dit vereist iets meer oefening, maar geeft meer controle.

```
<Button Content="Klik mij!" Width="100" Height="30" />
```

Deze regel voegt een knop toe met tekst 'Klik mij!'', en stelt de breedte en hoogte in.

3.2.3. Via C#-code (niet aanbevolen voor beginners)

Soms kan je ook in de C#-code een control aanmaken. Dat is handig voor dynamische interfaces, maar voorlopig raden we aan om de Toolbox of XAML te gebruiken.

```
Button button = new Button();
button.Content = "Klik mij!";
button.Height = 30;
```

```
button.Width=100;
```

Dit soort code heb je nu nog niet nodig, maar onthou wel dat het kan.

3.3. Eigenschappen

3.3.1. Wat zijn eigenschappen?

Elke control heeft een hele reeks **eigenschappen** (properties). Dat zijn stukjes informatie die bepalen hoe de control eruit ziet of zich gedraagt.

Bijvoorbeeld:

- `Content`: wat er op een knop of label getoond wordt
- `Width` en `Height`: de afmetingen
- `Name`: de naam waarmee je de control herkent in je C#-code
- `IsEnabled`: bepaalt of de control actief of grijs is
- `Background`: de achtergrondkleur
- `FontSize`: hoe groot de tekst wordt weergegeven

Je kan deze eigenschappen instellen in XAML, of aanpassen via de designer.

3.3.2. Properties-venster

Wanneer je een control selecteert in de Designer of in de XAML-code, verschijnt het **Properties**-venster (meestal rechts).

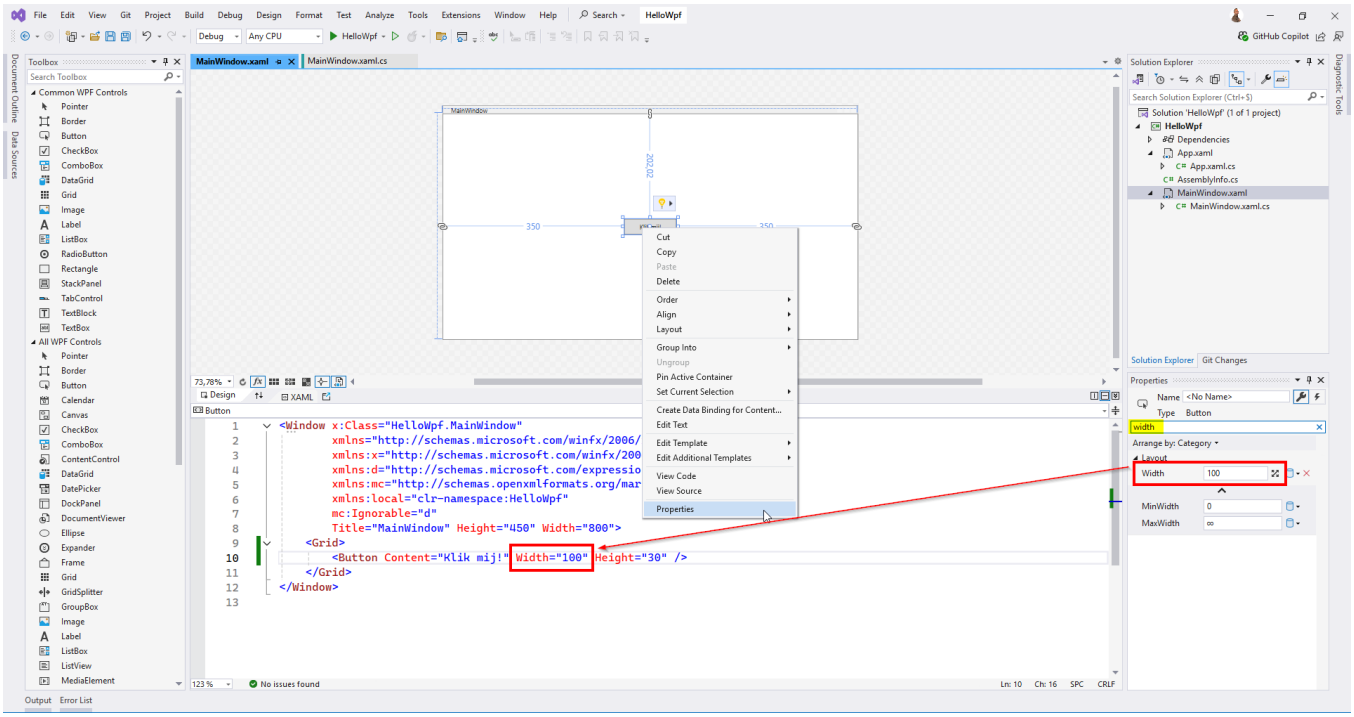
Met dit venster kan je eigenschappen aanpassen zonder zelf XAML-code te schrijven. Je ziet alle eigenschappen in een overzichtelijke lijst, vaak gegroepeerd per categorie (zoals 'Layout', 'Brush', 'Text'...).

Voorbeeld: als je een `Button` selecteert, kan je eenvoudig:

- de `Content` aanpassen naar bijvoorbeeld 'Verzenden'
- de `Width` instellen op 150
- de `Background` wijzigen naar een kleur zoals LightBlue

Elke aanpassing in het Properties-venster wordt automatisch omgezet naar XAML-code.

- Gebruik het zoekveld bovenaan in het Properties-venster om snel een eigenschap te vinden.
- Wanneer het Properties-venster niet zichtbaar is kan je dit eenvoudig tonen door met de rechtermuisknop een control te selecteren en dan de optie 'Properties' te kiezen.



3.3.3. Attributen

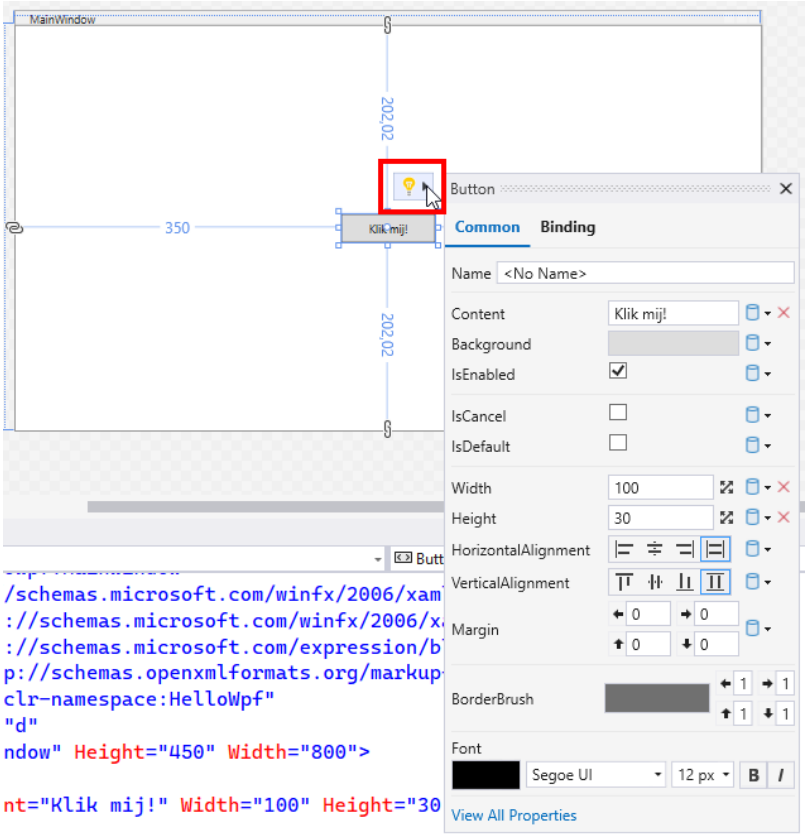
Je kan de eigenschappen van een control ook rechtstreeks beheren in de XAML-code. In XAML schrijf je die eigenschap als een **attribuut** binnen het element:

```
<Button Content="Klik mij!" Width="100" />
```

- De waarde van een attribuut staat altijd tussen twee `"` (dubbele quotes)
- Elk element kan meerdere attributen hebben

3.3.4. Smart Tag

De Smart Tag verschijnt als een klein lichtgrijs lampicoontje rechtsboven bij een geselecteerde control in de XAML Designer. Als je erop klikt, krijg je een lijstje met veelgebruikte eigenschappen of acties die je snel kan aanpassen.



> Niet alle controls tonen een Smart Tag. Alleen bepaalde (meestal veelgebruikte) controls zoals `Button`, `TextBox`, `ComboBox`, … bieden deze snelle configuratie.

3.4. Naam van een control

Als je een control in je code wil gebruiken (bijvoorbeeld om de tekst op te vragen of te wijzigen), dan moet je die een naam geven met de eigenschap `x:Name`.

Voorbeeld:

```
<TextBox x:Name="userNameTextBox" Width="200" />
```

In je C#-code kan je dan verwijzen naar `userNameTextBox.Text`.

> Geef alleen een naam aan controls die je ook echt wil gebruiken in je code. Een statisch label heeft meestal geen naam nodig.

3.4.1. Naming conventions voor controls

Technisch gezien is een control een variabele, daarom wordt dezelfde naming convention gebruikt maar dan met een suffix:

- Kies een **beschrijvende naam** die de functie van de control aanduidt
- Gebruik een **Engelse** benaming

- Gebruik **camelCase**: kleine letter bij de start, hoofdletters voor elk nieuw woord
- Sluit af met de **volledige naam van het type** als suffix (zoals `Button`, `TextBox`, `ComboBox`, …)

Voorbeelden:

Functie	Correcte naam
knop om op te slaan	`saveButton`
invoerveld voor gebruikersnaam	`userNameTextBox`
keuzelijst met opleidingen	`coursesComboBox`
lijst met studenten	`studentListBox`
vinkje voor nieuwsbrief	`newsletterCheckBox`
slider voor volume	`volumeSlider`

Waarom deze conventie gebruiken?

- De naam is leesbaar en betekenisvol
- Je weet altijd wat de control doet én welk type het is
- Het werkt goed samen met IntelliSense en is conform moderne .NET-aanbevelingen

Visual Studio stelt standaard namen zoals `button1` of `textBox2` voor, maar die pas je best zelf aan via het eigenschappenvenster.

3.5. Events

3.5.1. Wat zijn events?

Tot nu toe heb je geleerd hoe je een control toevoegt en de eigenschappen aanpast. Maar wat als je wil reageren op een actie van de gebruiker, zoals een klik op een knop?

Daarvoor gebruik je een **event**.

Een event is een gebeurtenis die optreedt, bijvoorbeeld:

- een knop wordt aangeklikt
- de muis beweegt over een element
- de gebruiker selecteert iets in een lijst

Om op zo'n event te reageren, moet je een **eventhandler** toevoegen. Dat is een methode die automatisch wordt uitgevoerd wanneer het event plaatsvindt.

3.5.2. Eventhandlers

Om een methode uit te voeren wanneer een event optreedt moet je de methode "koppelen" aan het event. Dat kan op verschillende manieren:

Via het Properties-venster

1. Selecteer een control (bijvoorbeeld een knop) in de designer
2. Ga naar het Properties-venster
3. Klik bovenaan op het bliksemicoontje ⚡ (tabblad 'Events')
4. Zoek het gewenste event in de lijst, zoals `Click`
5. Dubbelklik in het veld naast het gewenste event → er wordt automatisch een nieuwe methode aangemaakt in je C#-code

Als je Visual Studio de methode automatisch laat aanmaken zal deze ook automatisch een naam genereren. Maar je kan ook zelf de naam van de methode bepalen door deze eerst in te geven en daarna te bevestigen met de `Enter`-toets. Een veelgebruikte prefix voor een eventhandler is ***On***.

[custom event name] | *eventhandler-propertiewindow.gif*

Via de XAML-code

1. Selecteer een control in de XAML-code
2. Type de naam van het event gevolgd door `=`
3. Selecteer `<New Event Handler>` → er wordt automatisch een nieuwe methode aangemaakt in je C#-code

Ook via deze manier kan je zelf de naam van de methode bepalen door de naam eerst in te geven en daarna te bevestigen met de `F12`-toets.

[custom event name] | *eventhandler-xamlcode.gif*

Voorbeeld

```
<Button Content="Verzenden" Click="OnSend_Click" />
```

In de C#-code van `MainWindow.xaml.cs` wordt dan automatisch deze methode toegevoegd:

```
private void OnSend_Click(object sender, RoutedEventArgs e)
{
    // Hier komt de code die moet worden uitgevoerd wanneer de gebruiker op de
knop klikt
}
```

3.5.3. Wat zijn sender en e?

Elke eventhandler die je toevoegt in WPF heeft een vaste structuur met twee parameters:

```
private void exampleButton_Click(object sender, RoutedEventArgs e)
{
    // code bij klikken
}
```

Maar wat betekenen `sender` en `e`?

Parameter	Betekenis
`sender`	Verwijst naar het object dat het event heeft veroorzaakt (bijvoorbeeld de knop die geklikt werd)
`e`	Bevat extra informatie over het event (zoals de muispositie, toetsenstatus, … afhankelijk van het type event)

Je kan bijvoorbeeld het `sender`-object gebruiken om te achterhalen **welke knop** geklikt werd, als meerdere knoppen dezelfde methode gebruiken:

```
private void handleClick(object sender, RoutedEventArgs e)
{
    Button clickedButton = (Button)sender;
    clickedButton.Content = "Clicked!";
}
```

Zo zie je dat `sender` je toelaat om te werken met generieke eventhandlers en toch te weten wie het event veroorzaakte.

3.6. Veel voorkomende controls

De lijst van bestaande controls is vrij uitgebreid. In deze cursus beperken we ons tot de meest gebruikte. Op basis van hun doel kunnen we deze verdelen in verschillende categorieën:

Categorie	Controls	Omschrijving
Display controls	Label, TextBlock, Image	Informatie tonen aan de gebruiker via tekst of afbeelding
Selection controls	CheckBox, RadioButton, ComboBox, ListBox, Slider	De gebruiker laat een keuze achter, één of meerdere tegelijk
Input controls	TextBox, PasswordBox	De gebruiker voert tekst of wachtwoorden in

Categorie	Controls	Omschrijving
Date controls	Calendar, DatePicker	De gebruiker kiest een datum uit een kalender
Layout controls - basis	Grid, StackPanel, WrapPanel, DockPanel, Canvas	De plaatsing en verdeling van controls in het venster
Layout controls - structuur	GroupBox, Border, Panel	Extra visuele structuur of afbakening binnen het venster
Other controls	Menu, StatusBar, Tooltip, TabControl, DataGrid	Overige nuttige onderdelen zoals menu's, tabbladen of datatabellen

In de volgende hoofdstukken leer je de belangrijkste kenmerken van deze controls!

Een volledige lijst van alle bestaande controls in WPF vind je terug in de Microsoft documentatie.

4. Het Window element

4.1. Wat is een Window?

- Een `Window` is het hoofdscherm van je applicatie. Het wordt automatisch aangemaakt wanneer je een nieuw WPF-project maakt.
- Alles wat je in XAML schrijft, zit eigenlijk **binnen het `Window`-element**.
- Net zoals andere controls heeft een `Window` eigenschappen die je kan instellen in XAML of aanpassen via C#.

```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        Title="My First Window"
        Height="300"
        Width="400">

    <!-- Hier komen je andere elementen zoals knoppen of tekstvakken -->

</Window>
```

4.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Title`	De titel die bovenaan het venster verschijnt
`Width` / `Height`	De afmetingen van het venster
`ResizeMode`	Toelaat of het venster groter/kleiner mag worden (`CanResize`, `NoResize`, `CanMinimize`)
`WindowStartupLocation`	Bepaalt waar het venster op het scherm verschijnt (`Manual`, `CenterScreen`, `CenterOwner`)
`Background`	Achtergrondkleur van het venster

4.2.1. Gebruik in C#

Je kan deze eigenschappen ook instellen of uitlezen in de code-behind:

```
this.Title = "Welcome!";
this.ResizeMode = ResizeMode.NoResize;
this.WindowStartupLocation = WindowStartupLocation.CenterScreen;
```

Het keyword `this` verwijst naar het huidige object waarin je je bevindt. In het code-

behind van een venster verwijst `this` dus naar het venster zelf.

Wanneer je schrijft:

```
this.Title = "Welkom!";
```

is dat exact hetzelfde als:

```
Title = "Welkom!";
```

`this` gebruiken is niet verplicht, maar het maakt soms duidelijker over welk object je spreekt, vooral als er verwarring kan ontstaan met andere variabelen of parameters.

4.3. Veelgebruikte events

Event	Wanneer getriggerd?
`Loaded`	Zodra het venster volledig geladen is
`Closing`	Net voor het venster sluit (kan geannuleerd worden)
`Closed`	Zodra het venster effectief gesloten is

4.4. Veelgebruikte methodes

Ook een `Window` heeft methodes die je kan oproepen vanuit C#. De belangrijkste in deze fase van de cursus is `Close()`.

Methode	Uitleg
`Close()`	Sluit het huidige venster. De applicatie blijft draaien zolang er nog andere vensters open zijn.

4.4.1. Gebruik in C#

```
private void OnExitButton_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}
```

- Wanneer `Close()` wordt uitgevoerd, zal het `Closing`-event worden getriggerd.
- Als het venster gesloten is, stopt je applicatie automatisch.

5. Button

5.1. Wat is een Button?

Een **Button** is één van de meest gebruikte controls in WPF. De gebruiker kan erop klikken om iets uit te voeren, zoals een berekening starten, een venster openen of gegevens tonen. Buttons bevatten meestal tekst, maar kunnen ook afbeeldingen tonen.

5.2. Een eerste Button

Hier zie je een eenvoudige button in XAML:

```
<Button Content="Click me" />
```

De property `Content` bepaalt de tekst op de knop.

5.3. Belangrijke eigenschappen

Eigenschap	Betekenis
`Content`	Tekst of andere inhoud
`Width`	Breedte van de knop
`Height`	Hoogte van de knop
`Margin`	Buitenmarge rond de knop
`Padding`	Ruimte tussen tekst en rand
`HorizontalAlignment`	Horizontale uitlijning binnen de container
`VerticalAlignment`	Verticale uitlijning binnen de container
`IsEnabled`	Of de knop actief is.

Voorbeeld:

```
<Button Content="Calculate"
        Width="120"
        Height="40"
        Margin="10"
        HorizontalAlignment="Right"
        VerticalAlignment="Center" />
```

5.4. Veelgebruikte events

Event	Wanneer getriggerd?
`Click`	Als er op de knop geklikt wordt

```
<Button Content="Calculate"
        x:Name="calculateButton"
        Click="calculateButton_Click" />
```

De code-behind bevat de event procedure:

```
private void calculateButton_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("You clicked the button!");
}
```

5.5. Gebruik in C#

Je kunt eigenschappen wijzigen via de code-behind:

```
calculateButton_Click.Content = "Started";
calculateButton_Click.IsEnabled = false;
```

5.6. In het kort

- Een Button is één van de belangrijkste en meest gebruikte controls.
- Je bepaalt de tekst via `Content`.
- Via het `Click`-event reageer je op een klik.

6. Display controls

- een titel of uitleg op je formulier
- een melding na een actie
- een afbeelding of pictogram

De belangrijkste display controls in WPF zijn `Label`, `TextBlock` en `Image`.

6.1. Label

Een `Label` is een eenvoudige control om korte tekst te tonen, bijvoorbeeld naast een `TextBox`. Je kan ook de `Name` instellen zodat je via code de inhoud kan wijzigen.

```
<Label x:Name="nameLabel" Content="Naam:" FontSize="14" Margin="10"/>
```

6.1.1. Belangrijke eigenschappen

Eigenschap	Betekenis
`Content`	De tekst die je wilt tonen
`FontSize`	Grootte van de tekst
`FontWeight`	Dikte van de tekst (bijv. `Bold`)
`HorizontalAlignment`	Horizontale uitlijning (`Left`, `Center`, …)
`Margin`	Ruimte rondom het label

6.1.2. Gebruik in C#

```
nameLabel.Content = "First name:";
nameLabel.FontWeight = FontWeights.Bold;
```

6.1.3. Veelgebruikte events

Event	Wanneer getriggerd?
`MouseEnter`	Als de muis over het label gaat
`MouseLeave`	Als de muis het label verlaat

6.2. TextBlock

Een `TextBlock` lijkt op een `Label`, maar is krachtiger als je langere stukken tekst wil tonen. Je gebruikt `TextBlock` wanneer je meer controle wil over de opmaak, of wanneer de tekst niet statisch blijft.

```
<TextBlock x:Name="infoTextBlock" Text="Welcome to the app!" FontSize="16"
TextWrapping="Wrap" Margin="10"/>
```

6.2.1. Belangrijke eigenschappen

Eigenschap	Betekenis
`Text`	De tekst die je wilt tonen
`TextWrapping`	Of de tekst mag doorlopen op een nieuwe lijn
`FontStyle`	`Normal` of `Italic`
`TextAlignment`	Uitlijning binnen de `TextBlock`

6.2.2. Gebruik in C#

```
infoTextBlock.Text = "Thanks for using our app!";
infoTextBlock.FontStyle = FontStyles.Italic;
```

6.3. Wat is het verschil tussen Label en TextBlock?

- Gebruik `Label` voor korte aanduidingen bij invoervelden.
- Gebruik `TextBlock` voor langere of meer opgemaakte tekst, zoals uitleg of berichten.

6.4. Image

Met een `Image`-control toon je een afbeelding in je venster. Je kan dit gebruiken voor iconen, foto's of decoratieve elementen.

```
<Image x:Name="logoImage" Source="Images/logo.png" Width="150" Height="100"
Margin="10"/>
```

[add image] | *wpf-addimage.gif*

Om een afbeelding te kunnen tonen in een `Image`-control moet deze eerst worden toegevoegd aan je project:

1. Maak een map `Images` aan in je project.
2. Voeg daar het afbeeldingsbestand toe.
3. Selecteer het bestand in de Solution Explorer en stel `Build Action` in op `Resource` via het Properties-venster.

6.4.1. Belangrijke eigenschappen

Eigenschap	Betekenis
`Source`	Pad naar het afbeeldingsbestand
`Stretch`	Hoe de afbeelding zich aanpast (`Fill`, `Uniform`, `UniformToFill`, `None`)
`Width` / `Height`	Grootte van de afbeelding

6.4.2. Gebruik in C#

```
logoImage.Source = new BitmapImage(new Uri("Images/logo.png",
UriKind.Relative));
logoImage.Stretch = Stretch.Uniform;
```

7. Input controls

7.1. Wat zijn input controls?

Input controls zijn de besturingselementen van een WPF-scherm waarmee de gebruiker gegevens kan invoeren.

7.2. TextBox

Een `TextBox` is een invoerveld waarin de gebruiker tekst kan typen, zoals een naam, e-mailadres of vrije opmerking.

7.2.1. XAML-voorbeeld

```
<TextBox x:Name="userNameTextBox" Width="200" Margin="10" />
```

7.2.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Text`	De tekstinhoud van de textbox
`MaxLength`	Maximaal aantal tekens
`TextWrapping`	Laat toe dat tekst over meerdere regels loopt (`Wrap`)
`AcceptsReturn`	Maakt het mogelijk om enter te gebruiken (voor meerregelige tekst)
`IsReadOnly`	Voorkomt dat de gebruiker de inhoud aanpast
`HorizontalContentAlignment`	Uitlijning van de tekst in het veld

Gebruik in C#

```
string name = userNameTextBox.Text; // Leest de inhoud van de textbox
userNameTextBox.Text = "Anonymous"; // Wijzigt de inhoud van de textbox
userNameTextBox.IsReadOnly = true;
```

7.2.3. Veelgebruikte events

Event	Wanneer getriggerd?
`TextChanged`	Elke keer als de inhoud wijzigt
`GotFocus`	Wanneer de gebruiker het veld selecteert
`LostFocus`	Wanneer het veld de focus verliest

7.2.4. De Text property

De belangrijkste eigenschap van een `TextBox` is de `Text`-property. Daarmee kan je de inhoud van het invoerveld **uitlezen** of **aanpassen**. Dit is iets wat in de praktijk zeer vaak voorkomt.

Voorbeeld:

```
string name = userNameTextBox.Text;      // Leest de inhoud van de TextBox
resultLabel.Content = $"Welkom {name}!";

userNameTextBox.Text = "Onbekend";       // Past de inhoud van de TextBox aan
```

De waarde die je uit een `TextBox` haalt is altijd van het type `string`. Wil je er een getal van maken, dan moet je dit zelf omzetten (bv. met `int.Parse()` of `int.TryParse()`).

7.2.5. Veelgebruikte methodes

Een `TextBox` heeft enkele zeer handige methodes die je vaak zal gebruiken in WPF-toepassingen.

Methode	Uitleg
`Clear()`	Verwijdert alle tekst uit de TextBox.
`SelectAll()`	Selecteert alle tekst binnen de TextBox.
`Focus()`	Zet de cursor in de TextBox zodat de gebruiker meteen kan typen.

Gebruik in C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    nameTextBox.Clear(); // wis alle tekst
    nameTextBox.Focus(); // zet de cursor in de textbox
}

private void OnSelect_Click(object sender, RoutedEventArgs e)
{
    nameTextBox.SelectAll(); // selecteer alle tekst
}
```

7.3. PasswordBox

Een `PasswordBox` lijkt op een `TextBox`, maar verbergt de ingevoerde tekens met sterretjes of bolletjes. Je gebruikt dit bijvoorbeeld voor wachtwoorden of geheime codes.

7.3.1. XAML-voorbeeld

```
<PasswordBox x:Name="passwordBox" Width="200" Margin="10" />
```

7.3.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Password`	De ingevoerde waarde als string
`MaxLength`	Maximaal aantal toegelaten tekens
`HorizontalContentAlignment`	Uitlijning van de tekst
`IsEnabled`	Bepaalt of de gebruiker iets kan invullen

7.3.3. Gebruik in C#

```
string password = passwordBox.Password;
passwordBox.Password = string.Empty;
passwordBox.IsEnabled = false;
```

7.3.4. Veelgebruikte events

Event	Wanneer getriggerd?
`PasswordChanged`	Telkens als het wachtwoord wijzigt
`GotFocus`	Wanneer het veld geselecteerd wordt

7.3.5. Veelgebruikte methodes

De `PasswordBox` ondersteunt dezelfde nuttige methodes als een TextBox, maar heeft intern een andere werking.

Methode	Uitleg
`Clear()`	Verwijdert het volledige wachtwoord.
`SelectAll()`	Selecteert het volledige wachtwoord (zoals bij TextBox).
`Focus()`	Zet de cursor in de PasswordBox.

8. Selection controls

8.1. Wat zijn selection controls?

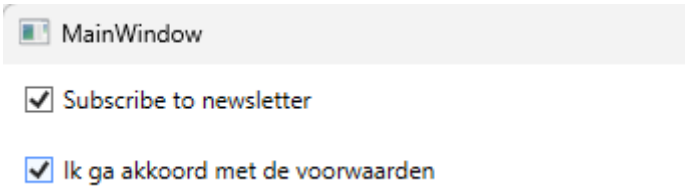
Selection controls zijn besturingselementen waarmee de gebruiker een keuze maakt. Dat kan gaan over eenvoudige ja/nee-vragen, het kiezen van een optie uit een lijst of het instellen van een getal via een schuifregelaar.

In WPF heb je verschillende types selection controls, elk met hun eigen typische gebruik:

- `CheckBox` - meervoudige keuze met vinkjes
- `RadioButton` - exclusieve keuze binnen een groep
- `ComboBox` - keuzelijst die standaard gesloten is
- `ListBox` - keuzelijst waarbij alle opties zichtbaar zijn
- `Slider` - continue of stapsgewijze selectie van een getal

8.2. CheckBox

Een `CheckBox` is een aankruisvakje waarmee de gebruiker een optie kan in- of uitschakelen. Je kan één of meerdere vakjes tegelijk aanvinken.



8.2.1. XAML-voorbeeld

```
<CheckBox x:Name="newsletterCheckBox" Content="Subscribe to newsletter"
IsChecked="True" Margin="10"/>
<CheckBox x:Name="confirmLicenseTermsCheckBox" Content="Ik ga akkoord met de
voorwaarden" IsChecked="True" Margin="10"/>
```

8.2.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Content`	De tekst naast het vinkje
`IsChecked`	`true`, `false` of `null` (indien `IsThreeState = true`)
`IsEnabled`	Bepaalt of de checkbox actief is
`IsThreeState`	Laat ook een onbepaalde (grijze) toestand toe

8.2.3. Gebruik in C#

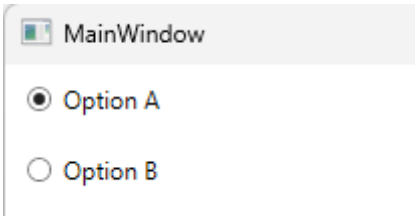
```
bool? isSubscribed = newsletterCheckBox.IsChecked;
newsletterCheckBox.IsEnabled = false;
```

8.2.4. Veelgebruikte events

Event	Wanneer getriggerd?
`Checked`	Wanneer de checkbox aangevinkt wordt
`Unchecked`	Wanneer het vinkje verwijderd wordt

8.3. RadioButton

Een `RadioButton` gebruik je wanneer je de gebruiker exact één optie wil laten kiezen uit een groep. Binnen dezelfde groep kan er altijd maar één knop tegelijk geselecteerd zijn.



8.3.1. XAML-voorbeeld

```
<RadioButton x:Name="optionOneRadioButton" GroupName="options" Content="Option
A" />
<RadioButton x:Name="optionTwoRadioButton" GroupName="options" Content="Option
B" />
```

8.3.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Content`	De tekst naast de keuzeknop
`IsChecked`	`true` als dit item geselecteerd is
`GroupName`	Zorgt ervoor dat knoppen in dezelfde groep exclusief zijn (je kan maar 1 knop per groep aanduiden)
`IsEnabled`	Bepaalt of de knop aanklikbaar is

8.3.3. Gebruik in C#

```
if (optionOneRadioButton.IsChecked == true)
{
```

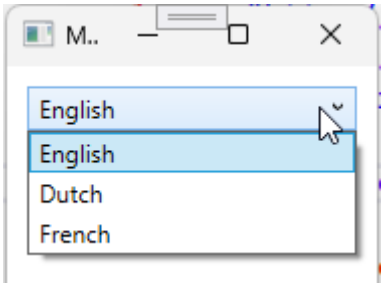
```
    // optie A werd geselecteerd
}
```

8.3.4. Veelgebruikte events

Event	Wanneer getriggerd?
`Checked`	Wanneer de knop geselecteerd wordt

8.4. ComboBox

Een `ComboBox` toont een uitklapbare lijst met opties. Standaard zie je alleen het geselecteerde item.



8.4.1. XAML-voorbeeld

```
<ComboBox x:Name="languageComboBox" SelectedIndex="0" Width="200">
  <ComboBoxItem Content="English" />
  <ComboBoxItem Content="Dutch" />
  <ComboBoxItem Content="French" />
</ComboBox>
```

8.4.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Items`	De opties in de lijst (via XAML of code)
`SelectedIndex`	Index van het geselecteerde item
`SelectedItem`	Het item zelf
`IsEditable`	Laat de gebruiker zelf tekst ingeven

8.4.3. Gebruik in C#

```
string selectedLanguage =
((ComboBoxItem)languageComboBox.SelectedItem).Content.ToString();
languageComboBox.SelectedIndex = 1;
```

8.4.4. Veelgebruikte events

Event	Wanneer getriggerd?
`SelectionChanged`	Wanneer een andere optie geselecteerd wordt

8.4.5. De Items-collectie

Je kan via C# elementen toevoegen of verwijderen uit een `ComboBox`, bijvoorbeeld op basis van een array of lijst.

```
// items toevoegen vanuit een string array
string[] languages = { "English", "Dutch", "French" };

languageComboBox.Items.Clear();
foreach (string lang in languages)
{
    languageComboBox.Items.Add(lang);
}
```

Andere nuttige methodes:

Methode	Uitleg
`Items.Add(item)`	Voegt een item toe aan de lijst
`Items.Remove(item)`	Verwijdert het item (indien gevonden)
`Items.Clear()`	Verwijdert alle items
`Items.Count`	Aantal items in de lijst

8.4.6. Objecten toevoegen

Je kan ook objecten van een eigen klasse toevoegen aan een `ComboBox`. De tekst die getoond wordt, komt dan uit de `ToString()`-methode van dat object.

```
public class Student
{
    public string FirstName { get; set; }

    public override string ToString()
    {
        return FirstName;
    }
}
```

Vervolgens voeg je de studenten toe aan de `ComboBox`:

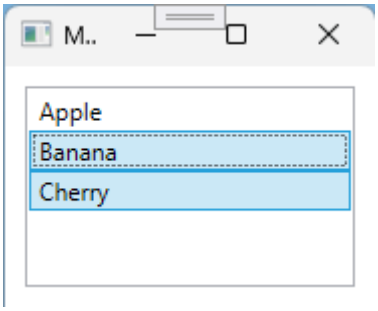
```
List<Student> students = new List<Student>
{
    new Student { FirstName = "Alice" },
    new Student { FirstName = "Bob" },
    new Student { FirstName = "Charlie" }
};

studentComboBox.Items.Clear();
foreach (Student student in students)
{
    studentComboBox.Items.Add(student);
}
```

Om het geselecteerde object op te vragen:

```
Student selectedStudent = (Student)studentComboBox.SelectedItem;
string name = selectedStudent.FirstName;
```

8.5. ListBox

Een `ListBox` toont een lijst met alle opties zichtbaar op het scherm. Je kan meerdere items tegelijk selecteren, afhankelijk van de instellingen.



8.5.1. XAML-voorbeeld

```
<ListBox x:Name="fruitListBox" SelectionMode="Multiple" Height="100">
  <ListBoxItem Content="Apple" />
  <ListBoxItem Content="Banana" />
  <ListBoxItem Content="Cherry" />
</ListBox>
```

8.5.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Items`	De lijst van keuzemogelijkheden

Eigenschap	Uitleg
`SelectionMode`	`Single`, `Multiple` of `Extended`
`SelectedItems`	Meerdere geselecteerde items
`SelectedIndex`	Index van het eerste geselecteerde item

8.5.3. Gebruik in C#

```
foreach (ListBoxItem item in fruitListBox.SelectedItems)
{
    string value = item.Content.ToString();
    // doe iets met value
}
```

8.5.4. Veelgebruikte events

Event	Wanneer getriggerd?
`SelectionChanged`	Wanneer selectie wijzigt

8.5.5. De Items-collectie

Net als bij een `ComboBox` kan je de inhoud van een `ListBox` dynamisch aanpassen via code.

```
// items toevoegen vanuit een lijst
List<string> fruits = new List<string> { "Apple", "Banana", "Cherry" };

fruitListBox.Items.Clear();
foreach (string fruit in fruits)
{
    fruitListBox.Items.Add(fruit);
}
```

Andere nuttige methodes:

Methode	Uitleg
`Items.Add(item)`	Voegt een nieuw item toe
`Items.Remove(item)`	Verwijdert een item
`Items.Clear()`	Leegt de hele lijst
`Items.Count`	Telt hoeveel items er zijn

8.6. SelectedItem en SelectedIndex

Zowel bij `ComboBox` als bij de `ListBox` zijn de eigenschappen `SelectedItem` en `SelectedIndex` onmisbaar om met gebruikerskeuzes te werken. Je kan met deze eigenschappen achterhalen **wat** de gebruiker geselecteerd heeft of via code instellen wat de selectie moet worden.

8.6.1. SelectedIndex

`SelectedIndex` geeft de **positie (index)** terug van het geselecteerde item in de lijst.

- De index start altijd vanaf `0`.
  - Eerste item geselecteerd → `SelectedIndex` is `0`
  - Tweede item geselecteerd → `SelectedIndex` is `1`
- Geen selectie → `SelectedIndex` is `-1`

Voorbeeld:

```
int index = languageComboBox.SelectedIndex;

if (index != -1)
{
    // er is een selectie
}
```

```
languageComboBox.SelectedIndex = 2;
```

8.6.2. SelectedItem

`SelectedItem` bevat het **effectief geselecteerde object** uit de lijst.

Wat dat object is, hangt af van **hoe je de items hebt toegevoegd**:

- `ComboBoxItem` of `ListBoxItem`
- een `string`
- een object van een eigen klasse

Voorbeeld:

```
ComboBoxItem item = (ComboBoxItem)languageComboBox.SelectedItem;
string language = item.Content.ToString();
```

```
string selectedFruit = (string)fruitListBox.SelectedItem;
```

```
Student selectedStudent = (Student)studentComboBox.SelectedItem;

if (selectedStudent != null)
{
    string name = selectedStudent.FirstName;
}
```

`SelectedItem` kan `null` zijn als er nog niets geselecteerd is. Controleer dit altijd.

8.6.3. SelectedIndex vs SelectedItem

Eigenschap	Wat krijg je?	Wanneer gebruiken?
`SelectedIndex`	Een getal (positie in de lijst)	Als de volgorde of positie belangrijk is
`SelectedItem`	Het geselecteerde object zelf	Als je met de inhoud of data wil werken

8.6.4. SelectionChanged

Vaak gebruik je `SelectedItem` of `SelectedIndex` binnen het `SelectionChanged` event.

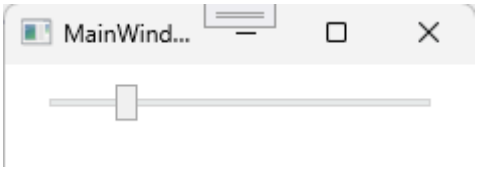
```
private void FruitListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (fruitListBox.SelectedItem != null)
    {
        string fruit = fruitListBox.SelectedItem.ToString();
    }
}
```

8.6.5. In het kort

- `SelectedIndex` geeft de positie van het geselecteerde item
- `SelectedItem` geeft het geselecteerde object zelf
- Geen selectie → `SelectedIndex = -1` en `SelectedItem = null`

8.7. Slider

Een `Slider` laat de gebruiker een waarde kiezen binnen een bereik. Je kan kiezen of je vaste tussenstappen toont met streepjes.



8.7.1. XAML-voorbeeld

```
<Slider x:Name="volumeSlider" Minimum="0" Maximum="100" Value="50"
TickFrequency="10" IsSnapToTickEnabled="True" Width="200" />
```

8.7.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Minimum`, `Maximum`	Het bereik waarbinnen gekozen wordt
`Value`	De actuele waarde
`TickFrequency`	Afstand tussen de streepjes
`IsSnapToTickEnabled`	Waarde springt naar dichtstbijzijnde streepje
`TickPlacement`	Positie waar de streepjes getoond worden

8.7.3. Gebruik in C#

```
int volume = (int)volumeSlider.Value;
volumeSlider.Value = 75;
```

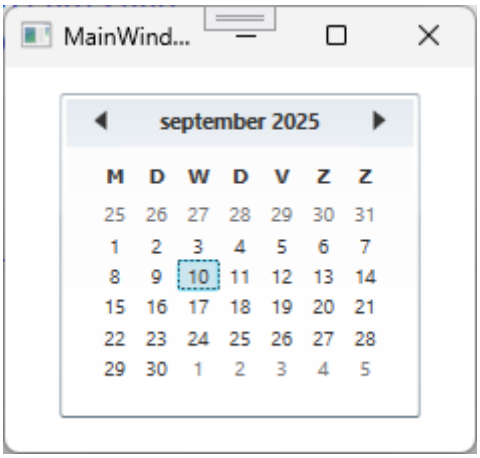
8.7.4. Veelgebruikte events

Event	Wanneer getriggerd?
`ValueChanged`	Wanneer de gebruiker de slider verschuift

9. Date controls

9.1. Calendar

Een `Calendar` toont een volledige maandweergave waarin de gebruiker één of meerdere datums kan aanklikken. Dit neemt redelijk wat plaats in beslag.



9.1.1. XAML-voorbeeld

```
<Calendar x:Name="eventCalendar" SelectionMode="SingleDate" />
```

9.1.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`SelectedDate`	De geselecteerde datum (of `null`)
`SelectionMode`	`SingleDate`, `SingleRange`, `MultipleRange`
`DisplayDate`	Datum die standaard getoond wordt
`IsTodayHighlighted`	Geeft vandaag een opvallende stijl

9.1.3. Gebruik in C#

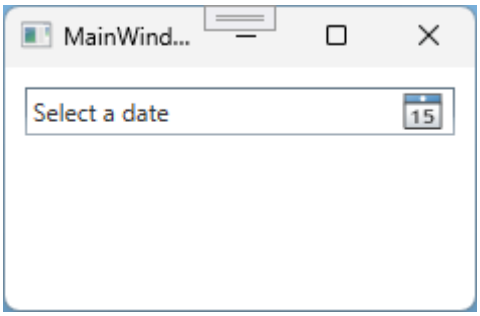
```
DateTime? selected = eventCalendar.SelectedDate;
eventCalendar.DisplayDate = DateTime.Today;
```

9.1.4. Veelgebruikte events

Event	Wanneer getriggerd?
`SelectedDatesChanged`	Wanneer de gebruiker een andere datum selecteert

9.2. DatePicker

Een `DatePicker` is compacter dan een `Calendar`. De gebruiker ziet één veld met een kalendericoon om een datum te kiezen.



9.2.1. XAML-voorbeeld

```
<DatePicker x:Name="birthDatePicker" SelectedDate="{x:Null}" />
```

9.2.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`SelectedDate`	De gekozen datum (of `null`)
`DisplayDateStart` / `DisplayDateEnd`	Min/max toegestane datum
`Text`	De tekst in het veld (meestal automatisch gegenereerd)
`IsDropDownOpen`	Bepaalt of de kalender opengeklapt is

9.2.3. Gebruik in C#

```
DateTime? birthDate = birthDatePicker.SelectedDate;
birthDatePicker.SelectedDate = DateTime.Today;
```

9.2.4. Veelgebruikte events

Event	Wanneer getriggerd?
`SelectedDateChanged`	Wanneer een nieuwe datum gekozen wordt

10. Layout controls

10.1. Wat zijn layout controls?

Layout controls zijn containers: ze bevatten andere elementen en bepalen **hoe** en **waar** die elementen getoond worden.

Elk WPF-venster heeft minstens één layout control als root (vaak een `Grid`).

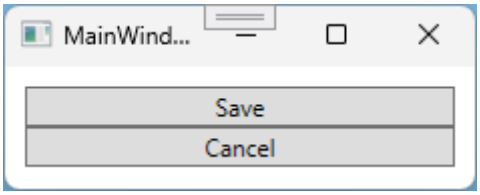
Control	Typisch gebruik
`StackPanel`	Elementen onder elkaar of naast elkaar
`WrapPanel`	Elementen automatisch naast elkaar, springt naar volgende lijn indien nodig
`DockPanel`	Elementen vastklikken aan boven-, onder-, linker- of rechterrand
`Grid`	Tabelstructuur met rijen en kolommen
`Canvas`	Absolute plaatsing met x- en y-coördinaten
`GroupBox`	Groepeert visueel een aantal controls
`Border`	Voegt een rand en achtergrond toe aan één element

10.2. StackPanel

Een `StackPanel` plaatst elementen automatisch **onder elkaar** of **naast elkaar**, afhankelijk van de `Orientation`.

10.2.1. XAML-voorbeeld

```
<StackPanel Orientation="Vertical">
    <Button Content="Save" />
    <Button Content="Cancel" />
</StackPanel>
```



10.2.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Orientation`	`Vertical` (standaard) of `Horizontal`
`Margin`	Ruimte rond de stack zelf
`HorizontalAlignment` / `VerticalAlignment`	Uitlijning binnen de parentcontainer

10.2.3. Gebruik in C#

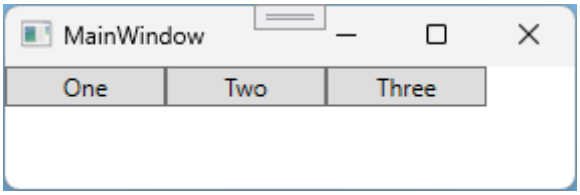
```
myStackPanel.Orientation = Orientation.Horizontal;
```

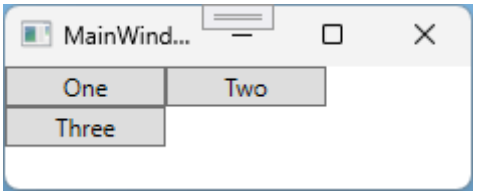
10.3. WrapPanel

Een `WrapPanel` plaatst elementen naast elkaar, maar springt automatisch naar een volgende rij als er geen plaats meer is.

10.3.1. XAML-voorbeeld

```
<WrapPanel Orientation="Horizontal" ItemWidth="80">
    <Button Content="One" />
    <Button Content="Two" />
    <Button Content="Three" />
</WrapPanel>
```





10.3.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Orientation`	Richting van de plaatsing (`Horizontal` of `Vertical`)
`ItemHeight` / `ItemWidth`	Afmetingen van elk kindelement

10.4. DockPanel

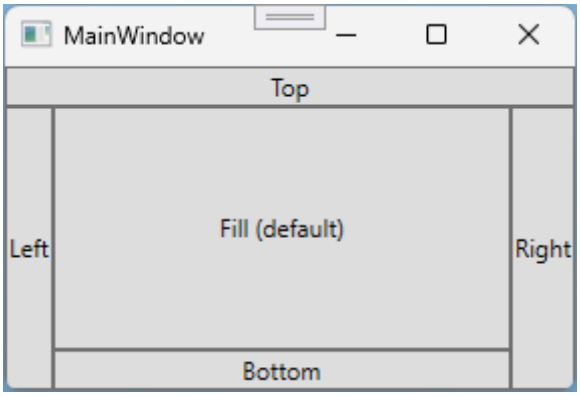
Een `DockPanel` laat je toe om elementen **vast te klikken aan een kant** van de container: boven, onder,

links of rechts. De resterende ruimte wordt automatisch ingevuld door het laatste element (dit gedrag is standaard ingeschakeld).

10.4.1. XAML-voorbeeld

Je plaatst elk element in een specifieke positie met behulp van de eigenschap `DockPanel.Dock`.

```
<DockPanel>
    <Button Content="Top" DockPanel.Dock="Top" />
    <Button Content="Left" DockPanel.Dock="Left" />
    <Button Content="Right" DockPanel.Dock="Right" />
    <Button Content="Bottom" DockPanel.Dock="Bottom" />
    <Button Content="Fill (default)" />
</DockPanel>
```



In dit voorbeeld:

- De eerste knop wordt bovenaan geplaatst
- De tweede aan de linkerkant
- De derde rechts
- De vierde onderaan
- De vijfde krijgt automatisch de resterende ruimte (omdat `LastChildFill = true`)

10.4.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`LastChildFill`	Laat het laatste element automatisch de resterende ruimte vullen
`DockPanel.Dock` (per child)	Bepaalt waar het child-element wordt vastgeklikt (`Top`, `Left`, `Right`, `Bottom`)

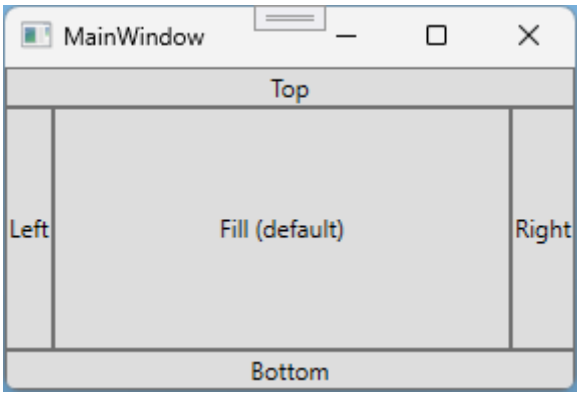
10.4.3. Volgorde is belangrijk

Een `DockPanel` verwerkt zijn child-elementen in volgorde. Elk nieuw element **neemt een deel van de resterende ruimte in beslag**.

De volgorde waarin je elementen plaatst in XAML is dus cruciaal voor het eindresultaat.

Als voorbeeld nemen we opnieuw een `dockpanel` met 5 knoppen, maar deze keer verplaatsen we de `button` onderaan een paar regels hoger in de XAML-code

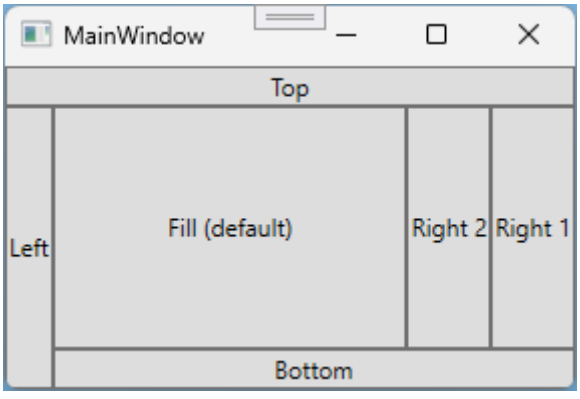
```
<DockPanel>
    <Button Content="Top" DockPanel.Dock="Top" />
    <Button Content="Bottom" DockPanel.Dock="Bottom" />
    <Button Content="Left" DockPanel.Dock="Left" />
    <Button Content="Right" DockPanel.Dock="Right" />
    <Button Content="Fill (default)" />
</DockPanel>
```



10.4.4. Meerdere elementen aan dezelfde kant

Je mag gerust meerdere elementen aan dezelfde zijde vastklikken. Ze worden dan **in volgorde** onder of naast elkaar geplaatst — afhankelijk van de zijde.

```
<DockPanel>
    <Button Content="Top" DockPanel.Dock="Top" />
    <Button Content="Left" DockPanel.Dock="Left" />
    <Button Content="Bottom" DockPanel.Dock="Bottom" />
    <Button Content="Right 1" DockPanel.Dock="Right" />
    <Button Content="Right 2" DockPanel.Dock="Right" />
    <Button Content="Fill (default)" />
</DockPanel>
```



In dit voorbeeld worden de 2 knoppen **naast elkaar** gezet aan de rechterkant, in de volgorde waarin ze in de XAML staan. Daarna vult de laatste knop de resterende ruimte.

Ook in dit voorbeeld zie je duidelijk hoe de volgorde van de child-elementen invloed heeft op het resultaat.

10.5. Grid

Een `Grid` verdeelt de ruimte in **rijen en kolommen**, zoals een tabel. Dit is de meest gebruikte layout-container in WPF.

10.5.1. XAML-voorbeeld

```
<Grid Margin="10">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="100"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <Label Content="Rij 0, Kolom 0" Grid.Row="0" Grid.Column="0"
Background="LightBlue" />
    <Label Content="Rij 0,&#10;Kolom 1" Grid.Row="0" Grid.Column="1"
Background="LightGreen" />
    <Label Content="Rij 0, Kolom 2" Grid.Row="0" Grid.Column="2"
Background="LightYellow" />
    <Label Content="Rij 0, Kolom 3" Grid.Row="0" Grid.Column="3"
Background="LightCoral" />

    <Label Content="Rij 1, Kolom 0" Grid.Row="1" Grid.Column="0"
Background="LightCyan" />
    <Label Content="Rij 1, Kolom 1" Grid.Row="1" Grid.Column="1"
Background="LightGoldenrodYellow" />
    <Label Content="Rij 1, Kolom 2" Grid.Row="1" Grid.Column="2"
Background="Honeydew" />
    <Label Content="Rij 1, Kolom 3" Grid.Row="1" Grid.Column="3"
Background="MistyRose" />

    <Label Content="Rij 2, Kolom 0" Grid.Row="2" Grid.Column="0"
Background="Lavender" />
    <Label Content="Rij 2, Kolom 1" Grid.Row="2" Grid.Column="1"
```

```
Background="LightSalmon" />
    <Label Content="Rij 2, Kolom 2" Grid.Row="2" Grid.Column="2"
Background="LightSteelBlue" />
    <Label Content="Rij 2, Kolom 3" Grid.Row="2" Grid.Column="3"
Background="PaleGreen" />
</Grid>
```

10.5.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`RowDefinitions` / `ColumnDefinitions`	Rijen en kolommen definiëren
`Grid.Row` / `Grid.Column`	Opgeven van locatie per element
`*`, `Auto`, getal	Manieren om breedte of hoogte te verdelen

10.5.3. Gebruik in C#

```
int rowCount = myGrid.RowDefinitions.Count;
```

10.5.4. Werken met rijen en kolommen

Een `Grid` lijkt op een onzichtbare tabel waarin je zelf bepaalt hoeveel rijen en kolommen er zijn. Je moet dit **altijd expliciet opgeven** via `RowDefinitions` en `ColumnDefinitions`.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
</Grid>
```

Bovenstaande `Grid` bestaat uit 2 rijen en 2 kolommen.

Je plaatst daarna elk element op de juiste positie met de eigenschappen `Grid.Row` en `Grid.Column`.

```
<Button Content="Links boven" Grid.Row="0" Grid.Column="0" />
<Button Content="Rechts onder" Grid.Row="1" Grid.Column="1" />
```

10.5.5. Breedtes en hoogtes instellen

Elke `RowDefinition` en `ColumnDefinition` kan op verschillende manieren worden geschaald:

Waarde	Betekenis
`Auto`	Past zich aan aan de grootte van de inhoud
`*`	Verdeelt de beschikbare ruimte evenredig
Getal (vb. `100`)	Vaste breedte of hoogte in pixels

Je kan `*` ook combineren:

```
<ColumnDefinition Width="2*" />
<ColumnDefinition Width="*" />
```

In dit voorbeeld krijgt de eerste kolom **2 delen**, en de tweede kolom **1 deel** van de resterende ruimte (verhouding 2:1).

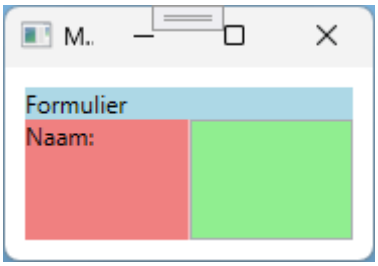
10.5.6. Cellen combineren met RowSpan en ColumnSpan

Soms wil je dat een element over meerdere rijen of kolommen loopt. Dit doe je met `Grid.RowSpan` en `Grid.ColumnSpan`.

Voorbeeld: een titel die over twee kolommen loopt

```
<Grid>
```

```
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <TextBlock Background="LightBlue" Text="Formulier" Grid.Row="0"
Grid.Column="0" Grid.ColumnSpan="2" />
    <TextBlock Background="LightCoral" Text="Naam:" Grid.Row="1" Grid.Column="0"
/>
    <TextBox Background="LightGreen" Grid.Row="1" Grid.Column="1" />
</Grid>
```



Op dezelfde manier kan je ook een element over meerdere rijen laten lopen met `Grid.RowSpan`.

Let op: je geeft de waarde mee als een getal, bijvoorbeeld `Grid.RowSpan="2"`.

10.5.7. Belangrijk om te onthouden

- Je moet het aantal rijen/kolommen altijd op voorhand definiëren
- Elementen zonder `Grid.Row` of `Grid.Column` komen standaard in rij 0, kolom 0
- Meerdere elementen kunnen op dezelfde cel geplaatst worden, maar dat leidt vaak tot overlappende inhoud

10.6. Canvas

Een `Canvas` laat je toe om elementen **manueel te positioneren** op basis van coördinaten. Dit gebruik je zelden in standaardformulieren, maar is handig voor tekenvensters of games.

10.6.1. XAML-voorbeeld

```
<Canvas>
    <Button Content="Click me" Canvas.Left="50" Canvas.Top="30"/>
</Canvas>
```

10.6.2. Belangrijkste eigenschappen

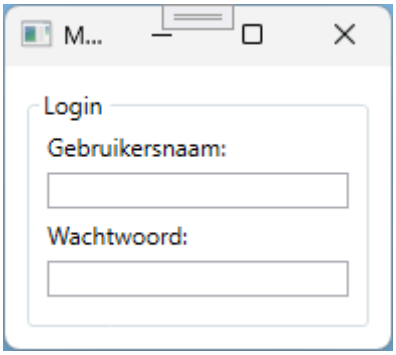
Eigenschap	Uitleg
`Canvas.Left` / `Canvas.Top`	De x- en y-positie van een element binnen de canvas
`Width` / `Height`	Afmetingen van de canvas (indien niet automatisch)

10.7. GroupBox

Een `GroupBox` groepeert elementen die logisch bij elkaar horen. Meestal gebruik je dit voor invoervelden die eenzelfde thema hebben.

10.7.1. XAML-voorbeeld

```
<GroupBox Header="Login" Margin="10">
    <StackPanel>
        <Label Content="Gebruikersnaam:"/>
        <TextBox Width="150" />
        <Label Content="Wachtwoord:"/>
        <PasswordBox Width="150" />
    </StackPanel>
</GroupBox>
```



Een `GroupBox` kan slechts **één child-element** bevatten.

Wil je meerdere controls groeperen? Plaats dan eerst een container (zoals een `StackPanel` of `Grid`) binnen de `GroupBox`. Die container telt als het ene toegestane child-element.

10.7.2. Belangrijkste eigenschappen

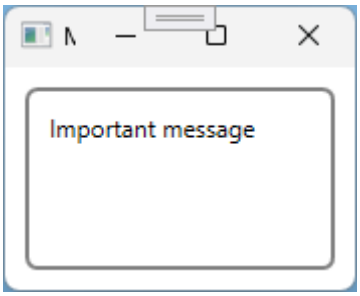
Eigenschap	Uitleg
`Header`	Titel van de groep
`Margin` / `Padding`	Ruimte rond of binnen de box
`Content`	Het element dat binnen de groep staat (vaak een container)

10.8. Border

Een `Border` gebruik je om een enkel element van een rand, achtergrondkleur of afronding te voorzien.

10.8.1. XAML-voorbeeld

```
<Border BorderBrush="Gray" BorderThickness="2" CornerRadius="5" Padding="10">
    <TextBlock Text="Important message" />
</Border>
```



10.8.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`BorderBrush`	Kleur van de rand
`BorderThickness`	Dikte van de rand
`CornerRadius`	Afronding van de hoeken
`Padding`	Ruimte tussen rand en inhoud
`Child`	Het enige element binnen de border

10.9. ViewBox

Een `ViewBox` zorgt ervoor dat de inhoud **automatisch schaalt** zodat ze altijd past binnen de beschikbare ruimte.

In tegenstelling tot andere layout controls bepaalt een `ViewBox` niet waar elementen staan, maar **hoe groot ze worden weergegeven**. De volledige inhoud wordt proportioneel groter of kleiner gemaakt wanneer het venster van grootte verandert.

Een `ViewBox` bevat **exact één child-element**. In de praktijk is dat meestal een `Grid`, `StackPanel` of een `Image`.

Een `ViewBox` schaalt de **inhoud**, niet de container zelf. Dat maakt dit control fundamenteel anders dan een `Grid` of `StackPanel`.

10.9.1. Eenvoudig voorbeeld

In dit voorbeeld schaalt de tekst automatisch mee met de grootte van het venster:

```
<ViewBox>
    <TextBlock Text="Deze tekst schaalt mee"
               FontSize="24"
               FontWeight="Bold" />
</ViewBox>
```

Wanneer je het venster groter maakt, wordt de tekst groter. Wanneer je het venster verkleint, wordt de tekst kleiner.

10.9.2. ViewBox met meerdere controls

Omdat een `ViewBox` slechts één child mag hebben, plaats je er meestal een container in, zoals een `Grid`:

```
<ViewBox>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Label Content="Naam:" />
        <TextBox Grid.Row="1" Width="200" />
    </Grid>
</ViewBox>
```

Hier schaalt **de volledige layout** (label en textbox samen) mee.

10.9.3. Wanneer gebruik je een ViewBox?

Een `ViewBox` is vooral nuttig wanneer:

- de verhouding van de inhoud belangrijk is
- je een schaalbaar infopaneel of dashboard maakt
- je afbeeldingen, iconen of titels correct wil laten meegroeien

Gebruik een `ViewBox` liever niet:

- voor standaard formulieren met invoervelden
- wanneer leesbaarheid van tekst cruciaal is
- als je exacte controle over afmetingen nodig hebt

10.9.4. Belangrijk om te onthouden

- Een `ViewBox` schaalt zijn inhoud automatisch
- Er mag slechts één child-element in zitten
- Vaak combineer je een `ViewBox` met een `Grid`
- Gebruik dit control bewust en spaarzaam

10.10. Document Outline-venster

Wanneer je werkt met geneste layout containers zoals `Grid`, `StackPanel`, `GroupBox`, … kan het soms moeilijk zijn om het overzicht te bewaren.

Gelukkig biedt Visual Studio het **Document Outline**-venster aan. Dit toont je de volledige structuur van het venster in boomvorm: elke container, control en genest element netjes op een rij.

10.10.1. Hoe open je het?

1. Klik in de menubalk op `View` (Beeld)
2. Kies daarna `Other Windows` → `Document Outline`

Je ziet dan een overzicht zoals dit:

```
Window
 └── Grid
     ├── StackPanel
     │    ├── Label
     │    └── TextBox
     └── Button
```

Je kan klikken op een element in de boom om het meteen te selecteren in het XAML-bestand of in de Designer. Het is een handig hulpmiddel bij debugging van layoutproblemen of bij het selecteren van deeply nested elementen.

11. Andere WPF-controls

11.1. TabControl

Een `TabControl` laat je toe om inhoud op te splitsen in verschillende tabbladen. Elk tabblad bevat zijn eigen content.

11.1.1. XAML-voorbeeld

```
<TabControl x:Name="mainTabControl" Margin="10">
    <TabItem Header="Home">
        <TextBlock Text="Welkom op het hoofdscherm." Margin="10"/>
    </TabItem>
    <TabItem Header="Instellingen">
        <CheckBox Content="Geluid aan" Margin="10"/>
    </TabItem>
</TabControl>
```

11.1.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Items`	De tabbladen (meestal `TabItem`-elementen)
`SelectedIndex`	Huidig geselecteerd tabblad (via index)
`SelectedItem`	Het actieve tabbladobject

11.1.3. Gebruik in C#

```
int index = mainTabControl.SelectedIndex;
mainTabControl.SelectedIndex = 1;
```

11.1.4. Veelgebruikte events

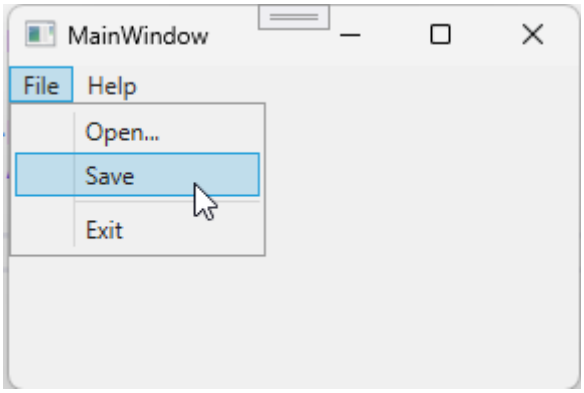
Event	Wanneer getriggerd?
`SelectionChanged`	Wanneer de gebruiker van tabblad wisselt

11.2. Menu

Een `Menu` gebruik je om klassieke menustructuren te maken bovenaan je venster (zoals `Bestand`, `Bewerken`, …).

11.2.1. XAML-voorbeeld

```
<Menu>
    <MenuItem Header="_File">
        <MenuItem Header="Open..." />
        <MenuItem Header="Save" />
        <Separator />
        <MenuItem Header="Exit" />
    </MenuItem>
    <MenuItem Header="_Help">
        <MenuItem Header="About" />
    </MenuItem>
</Menu>
```



In dit voorbeeld:

- Een underscore `_` geeft aan welke toetscombinatie met Alt werkt
- `Separator` voegt een visuele scheiding toe tussen items

11.2.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`Header`	De tekst die in het menu wordt weergegeven
`Items`	Submenu-items (andere `MenuItem`s)

11.2.3. Gebruik in C#

```
MenuItem exitItem = new MenuItem { Header = "Exit" };
fileMenuItem.Items.Add(exitItem);
```

11.2.4. Veelgebruikte events

Event	Wanneer getriggerd?
`Click` (op `MenuItem`)	Wanneer de gebruiker op een item klikt

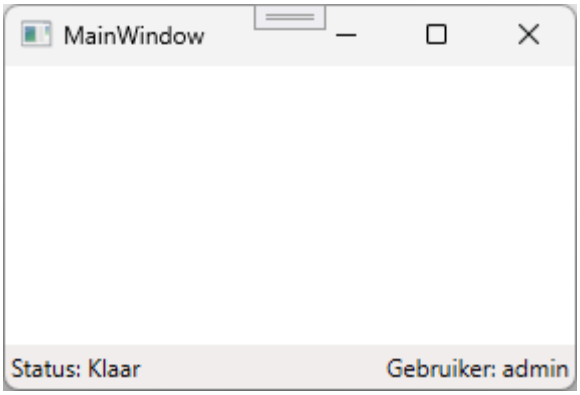
het Menu zelf genereert meestal geen events — je koppelt events aan de MenuItems.

11.3. StatusBar

Een `StatusBar` staat meestal onderaan het venster en toont informatie over de status van de toepassing.

11.3.1. XAML-voorbeeld

```
<StatusBar>
    <StatusBarItem>
        <TextBlock Text="Klaar" />
    </StatusBarItem>
    <StatusBarItem HorizontalAlignment="Right">
        <TextBlock Text="Gebruiker: admin" />
    </StatusBarItem>
</StatusBar>
```



11.3.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`StatusBarItem`	Een item binnen de statusbalk
`HorizontalAlignment`	Kan gebruikt worden om items naar rechts te duwen

Eigenschap	Uitleg
`Content`	Wat er effectief getoond wordt

11.3.3. Gebruik in C#

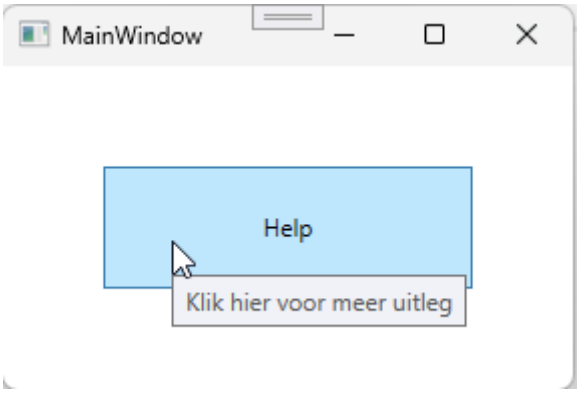
```
statusTextBlock.Text = "Bestand opgeslagen";
```

11.4. ToolTip

Een `ToolTip` is een kleine tekstballon die verschijnt als je met de muis over een element beweegt.

11.4.1. XAML-voorbeeld

```
<Button Content="Help">
    <Button.ToolTip>
        Klik hier voor meer uitleg
    </Button.ToolTip>
</Button>
```



Je kan ook een `ToolTip` als attribuut toevoegen:

```
<Button Content="Save" ToolTip="Bestand opslaan" />
```

11.4.2. Belangrijkste eigenschappen

Eigenschap	Uitleg
`ToolTip`	De tekst die verschijnt bij hover
`Placement`, `StaysOpen`, …	Meer geavanceerde opties (nu niet nodig)

11.4.3. Gebruik in C#

```
saveButton.ToolTip = "Klik om op te slaan";
```

11.4.4. Veelgebruikte events

Event	Wanneer getriggerd?
`ToolTipOpening`	Net voordat de tooltip verschijnt
`ToolTipClosing`	Net voordat de tooltip verdwijnt

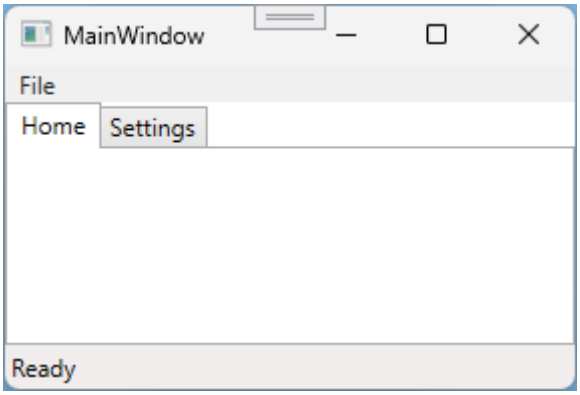
11.5. Layouttip: DockPanel

De controls `Menu`, `TabControl` en `StatusBar` worden vaak samen gebruikt in één venster. Een handige manier om dit te structureren is via een `DockPanel`:

- het `Menu` wordt bovenaan vastgedockt
- de `StatusBar` onderaan
- de `TabControl` of hoofdinhoud vult de resterende ruimte

11.5.1. Voorbeeld

```
<DockPanel>
    <Menu DockPanel.Dock="Top">
        <MenuItem Header="File" />
    </Menu>

    <StatusBar DockPanel.Dock="Bottom">
        <StatusBarItem>
            <TextBlock Text="Ready" />
        </StatusBarItem>
    </StatusBar>

    <TabControl DockPanel.Dock="Top">
        <TabItem Header="Home" />
        <TabItem Header="Settings" />
    </TabControl>
</DockPanel>
```



Vergeet niet dat het **laatste element** in een `DockPanel` automatisch de resterende

ruimte inneemt (indien `LastChildFill = true`, wat standaard het geval is). Plaats je `TabControl` dus **als laatste** in de XAML om het volledige midden te vullen.

12. Hulpmiddelen

12.1. MessageBox

Een `MessageBox` toont een eenvoudige pop-up met tekst en knoppen zoals `OK`, `Yes`, `No`, enz.

Je roept dit op via een statische methode in de klasse `MessageBox`.

12.1.1. Voorbeeld

```
MessageBox.Show("Wil je afsluiten?");
```

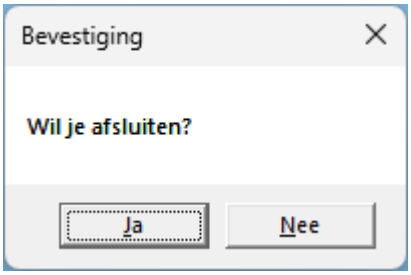
12.1.2. Opties

1. Je kan zelf de titel bepalen:

```
MessageBox.Show("Wil je afsluiten?", "Bevestiging");
```

2. Ook de knoppen zijn (beperkt) aanpasbaar:

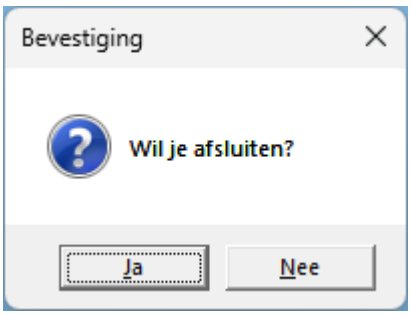
```
MessageBox.Show("Wil je afsluiten?", "Bevestiging", MessageBoxButton.YesNo);
```



3. Als je wil kan je zelfs kiezen uit verschillende iconen:

```
MessageBox.Show("Wil je afsluiten?", "Bevestiging", MessageBoxButton.YesNo,
```

```
MessageBoxImage.Question);
```



4. Wil je weten wat de gebruiker gekozen heeft?

```
MessageBoxResult result = MessageBox.Show("Wil je opslaan?", "Bevestiging",
MessageBoxButton.YesNoCancel);

if (result == MessageBoxResult.Yes)
{
    // Opslaan
}
```

12.1.3. Veelgebruikte opties

Knoppen

Optie	Betekenis
`MessageBoxButton.OK`	Toont alleen een OK-knop
`MessageBoxButton.OKCancel`	OK en Annuleren
`MessageBoxButton.YesNo`	Ja en Nee
`MessageBoxButton.YesNoCancel`	Ja, Nee en Annuleren

Iconen

Icoon	Betekenis
`MessageBoxImage.Information`	Toont een informatie-icoon (blauw i)
`MessageBoxImage.Warning`	Toont een waarschuwingsicoon (geel driehoekje)
`MessageBoxImage.Error`	Toont een fouticoon (rood kruis)
`MessageBoxImage.Question`	Toont een vraagteken (meestal bij bevestigingen)

12.2. InputBox

Een `InputBox` toont een eenvoudig dialoogvenster met een tekstvak. Deze functionaliteit komt uit de `Microsoft.VisualBasic` namespace, maar werkt perfect in een WPF-project.

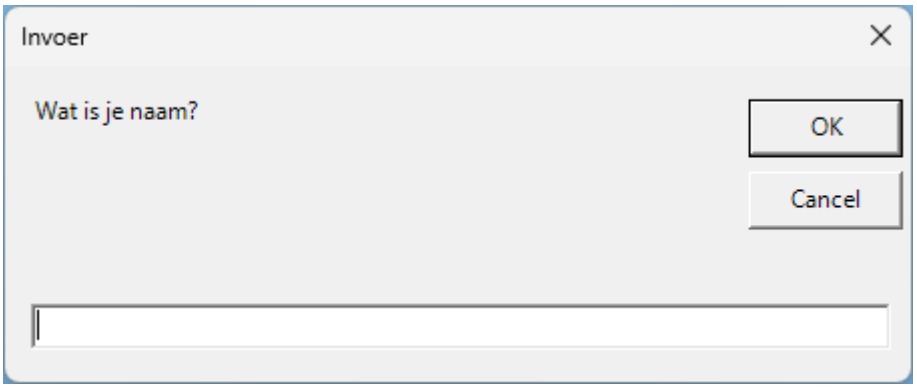
12.2.1. Opgelet

Om een `InputBox` te kunnen gebruiken, moet je bovenaan toevoegen:

```
using Microsoft.VisualBasic;
```

De methode ziet er als volgt uit:

```
string name = Interaction.InputBox("Wat is je naam?", "Invoer");
```



Je kan daarna de invoer gebruiken zoals elke andere string.

12.2.2. Belangrijke opmerkingen

- De `InputBox` is functioneel, maar minder flexibel qua layout
- Voor meer controle bouw je beter je eigen dialoogvenster (maar dat behandelen we niet in deze cursus)

12.3. DispatcherTimer

Een `DispatcherTimer` laat je toe om **op vaste tijdsintervallen code uit te voeren** in een WPF-applicatie.

Je gebruikt dit bijvoorbeeld om:

- een klok te tonen
- een teller te laten lopen
- periodiek iets te controleren
- eenvoudige spel- of animatielogica te maken

Een `DispatcherTimer` werkt op de **UI-thread**, wat betekent dat je vanuit de timer **rechtstreeks UI-elementen mag aanpassen**.

Omdat de timer op de UI-thread draait, is hij ideaal voor eenvoudige taken die regelmatig moeten gebeuren.

12.3.1. Basisidee

Een `DispatcherTimer` werkt altijd volgens hetzelfde patroon:

1. Je maakt een timer aan
2. Je koppelt een methode aan het `Tick`-event
3. Je stelt het interval in
4. Je start de timer

12.3.2. Voorbeeld: klok tonen

In dit voorbeeld tonen we de huidige tijd en werken we deze elke seconde bij.

```
using System.Windows.Threading;

private DispatcherTimer _timer = new DispatcherTimer();

public MainWindow()
{
    InitializeComponent();

    _timer.Interval = TimeSpan.FromSeconds(1);
    _timer.Tick += Timer_Tick;
    _timer.Start();
}

private void Timer_Tick(object sender, EventArgs e)
{
    timeLabel.Content = DateTime.Now.ToLongTimeString();
}
```

Elke seconde wordt de methode `Timer_Tick` uitgevoerd en wordt de inhoud van het label aangepast.

12.3.3. Interval instellen

Het interval bepaalt **hoe vaak** de `Tick`-methode wordt uitgevoerd.

Enkele voorbeelden:

```
_timer.Interval = TimeSpan.FromSeconds(1);    // elke seconde
_timer.Interval = TimeSpan.FromMilliseconds(500); // elke halve seconde
_timer.Interval = new TimeSpan(0, 0, 5);      // elke 5 seconden
```

12.3.4. Timer stoppen en opnieuw starten

Je kan een timer op elk moment stoppen of opnieuw starten:

```
_timer.Stop();   // timer pauzeren
_timer.Start();  // timer opnieuw starten
```

Dit is handig bij bijvoorbeeld:

- een start/stop-knop
- een spel dat gepauzeerd wordt
- een countdown die tijdelijk moet stoppen

12.3.5. Belangrijk om te onthouden

- Een `DispatcherTimer` voert code uit op vaste tijdsintervallen
- De code in `Tick` mag UI-elementen aanpassen
- Je moet de timer expliciet starten en stoppen
- Gebruik dit voor eenvoudige, periodieke taken