

WPF

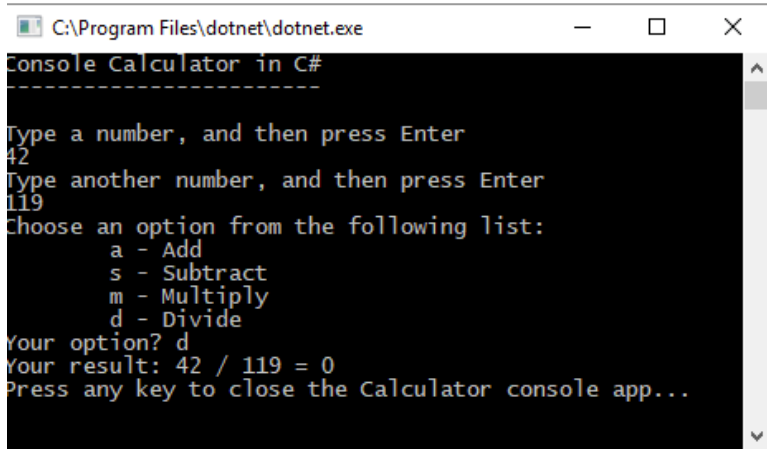
Bachelor IT

Sven Mariën

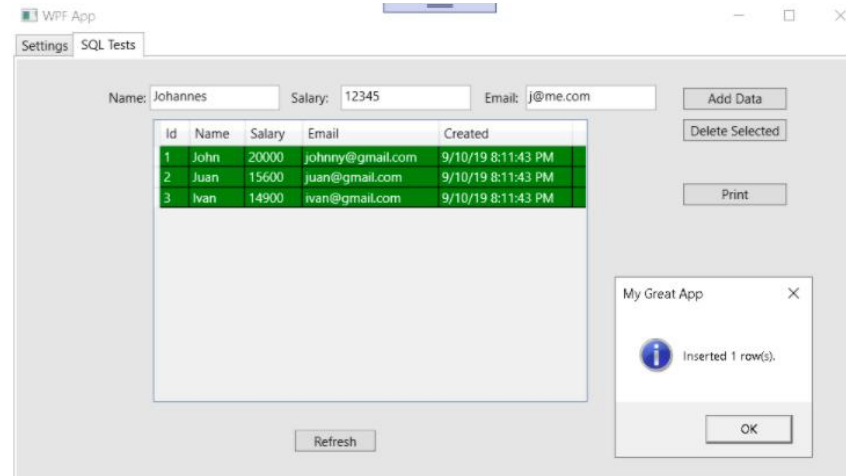
(sven.marien01@ap.be)

WPF ???

- Console applicatie versus Desktop applicatie
- CUI versus GUI (Graphical User Interface)

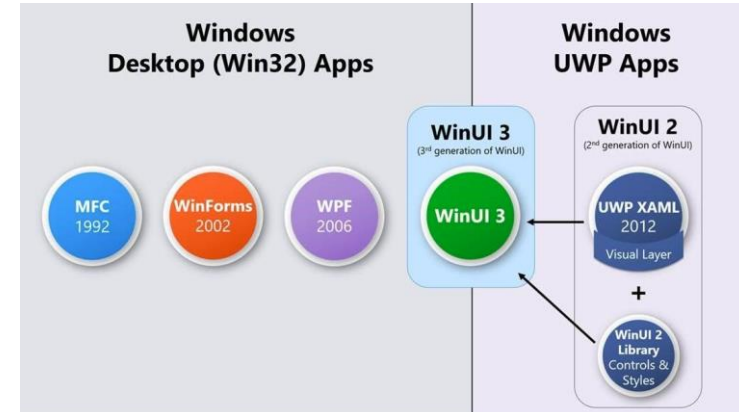


```
C:\Program Files\dotnet\dotnet.exe
Console Calculator in C#
-----
Type a number, and then press Enter
42
Type another number, and then press Enter
119
Choose an option from the following list:
a - Add
s - Subtract
m - Multiply
d - Divide
Your option? d
Your result: 42 / 119 = 0
Press any key to close the Calculator console app...
```

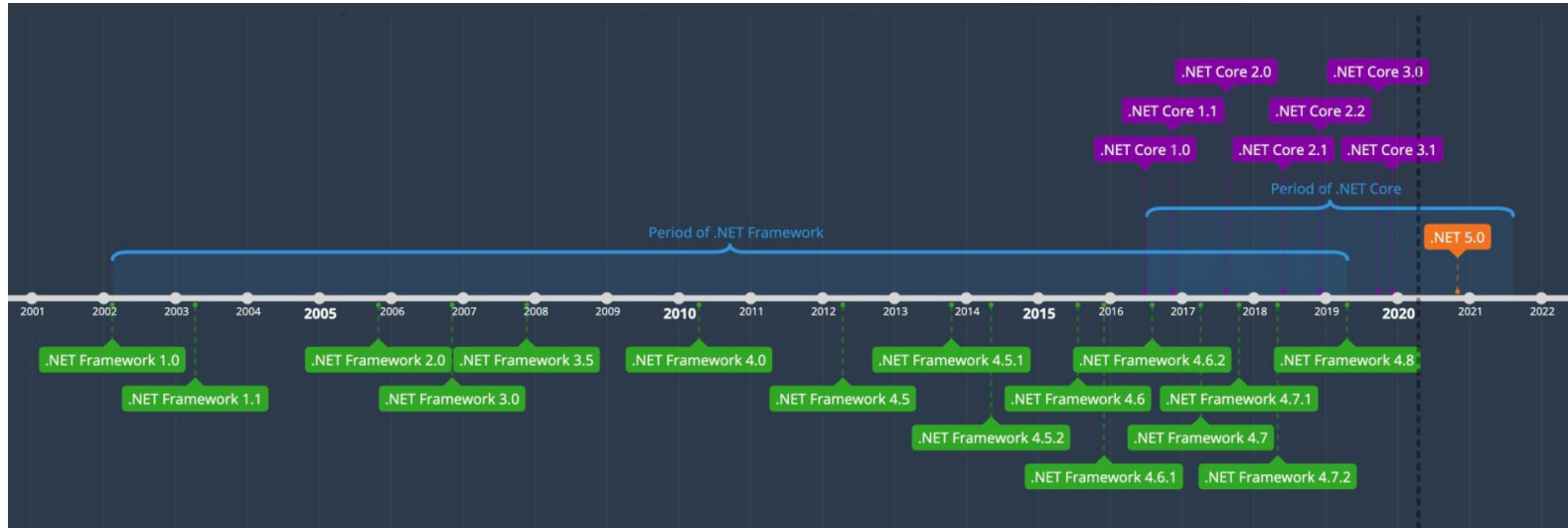


WPF ?

- Evolutie van (Microsoft) ontwikkel tools:
 - WinForms
 - Eerste versie bij .NET platform
 - WPF
 - Windows Presentation Foundation
 - Meer mogelijkheden dan Winforms
 - UWP, WinUI,.. ?
 - “Beperkt” tot Windows 10 (PC, tablet, Hololens,...)



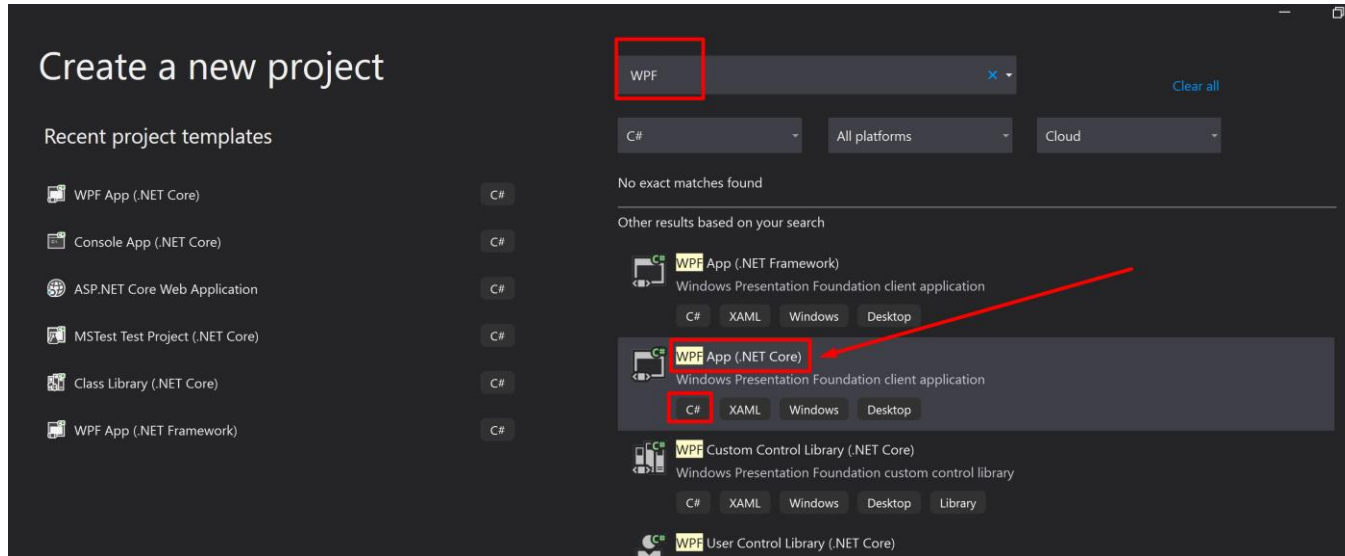
Evolutie van .NET framework



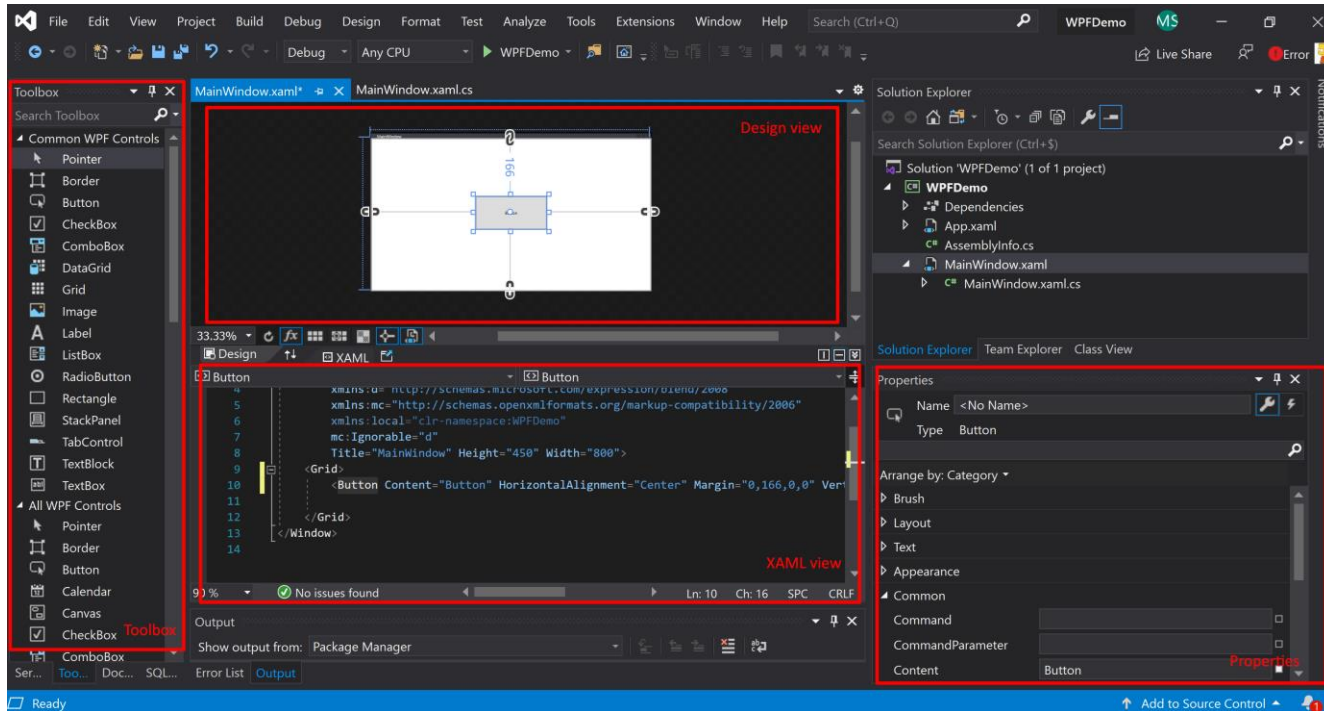
- Als je een nieuw project aanmaakt, kies dus best ofwel voor WPF **.NET Core** ofwel WPF **.NET** als Applicatie
- (en dus niet meer voor .NET Framework)

Nieuw WPF project aanmaken

- Vanuit Visual studio

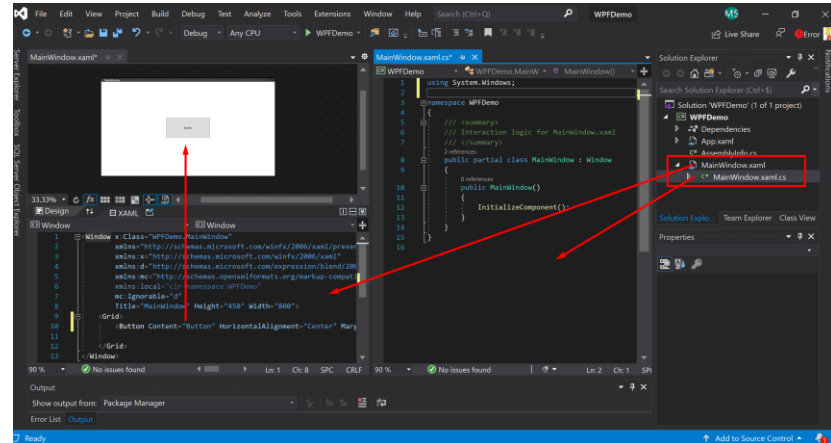


Verschillende “views” bij een WPF project



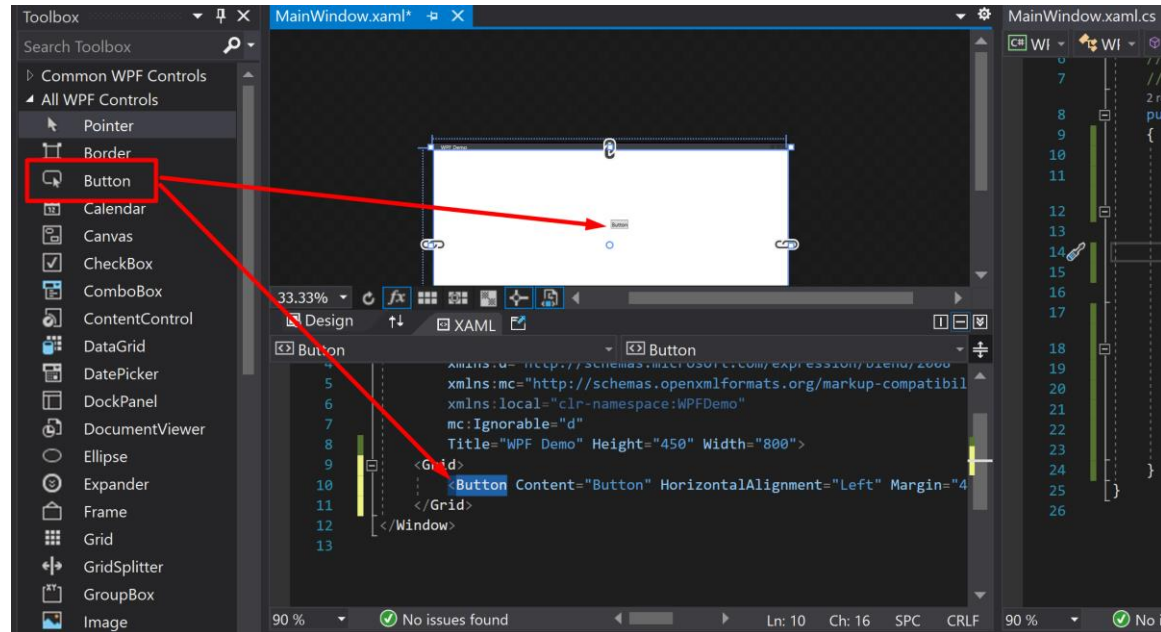
Vensters

- Een WPF applicatie wordt opgebouwd met 1 (of meerdere) “window(s)”
- Elk window bestaat uit 2 bestanden:
 - **XAML** (= inhoud, layout & opmaak)
 - **cs** (=c# code die achter de schermen zal worden uitgevoerd)
- Design view wordt gegenereerd adhv. xaml (en ook in sync gehouden bij aanpassingen)



Controls (Buttons, input fields,...)

- Een control toevoegen kan door deze te slepen vanuit de toolbox naar de design view (of de XAML view)



XAML

- Staat voor :eXtensible Application Markup Language
- Wordt uitgesproken als : “zammel”
- “Beschrijft” de UI (User interface)
- Is gebaseerd op XML

```
<Window x:Class="WPFDemo.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:WPFDemo"
        mc:Ignorable="d"
        Title="WPF Demo" Height="450" Width="800">
    <Grid>
        <Button Content="Button" HorizontalAlignment="Left" Margin="400,157,0,0" VerticalAlignment="Top"/>
    </Grid>
</Window>
```

```
<?xml version="1.0">
<address>
    <name>Bert Bibber</name>
    <mobile>+3299999999</mobile>
    <email>bertje@gmail.com </email>
    <birthdate>2000-01-01</birthdate>
</address>
```

XAML (2)

- Eerste element (root) = “window”
 - Daaronder een “layout” element (hier: grid)
 - Daaronder 1 of meerdere “controls” (TextBox, TextBlock, Button,...)

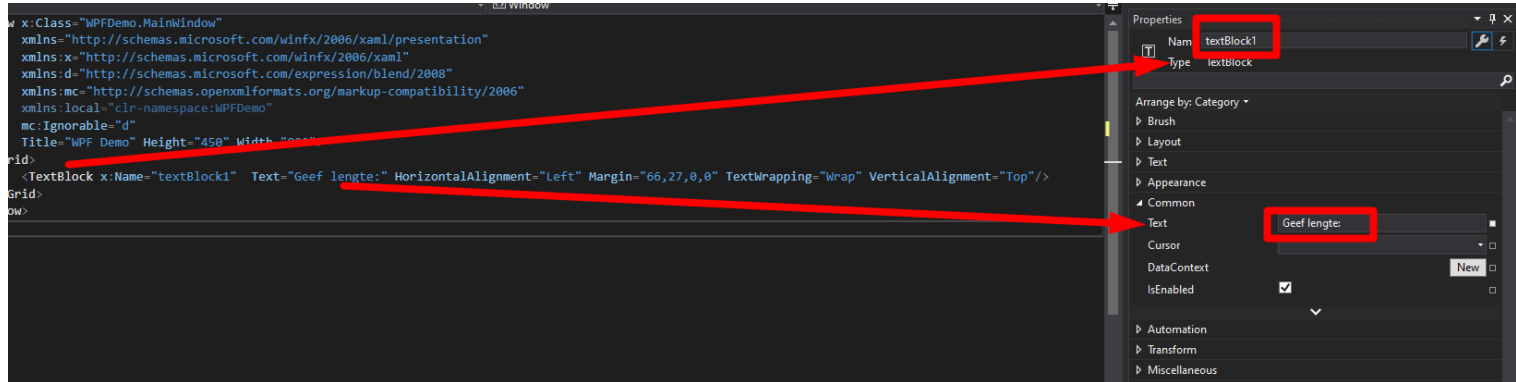


```
<Window x:Class="WPFDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPFDemo"
        mc:Ignorable="d"
        Title="WPFDemo" Height="450" Width="800">
    <Grid>
        <Button Content="Berekenen" HorizontalAlignment="Left" Margin="352,24,0,0" VerticalAlignment="Top" Height="37" Width="96"
        <TextBox x:Name="text1" HorizontalAlignment="Left" Margin="139,24,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="
        <TextBlock x:Name="textBlock1" HorizontalAlignment="Left" Margin="139,61,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
        <TextBlock HorizontalAlignment="Left" Margin="66,27,0,0" Text="Geef lengte:" TextWrapping="Wrap" VerticalAlignment="Top"
    </Grid>
</Window>
```

The screenshot shows a code editor with XAML code. A red arrow points from the top-left corner to the opening tag of the <Window> element. Another red arrow points from the top-left corner to the <Grid> element. A third red arrow points from the top-left corner to the <Button> element. A fourth red arrow points from the top-left corner to the <TextBlock> element. A red exclamation mark is placed next to the <TextBlock> element.

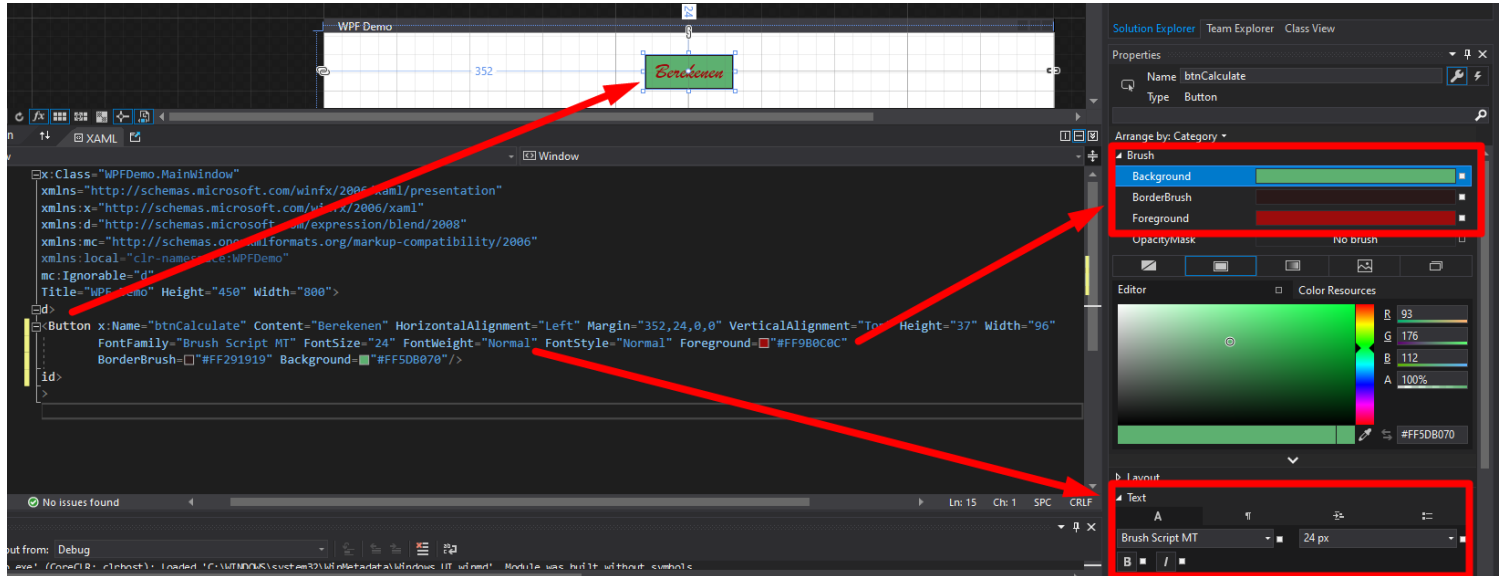
Controls en hun eigenschappen

- Elke control heeft een aantal instelbare eigenschappen (properties).
- Kunnen worden ingesteld in de XAML en/of via de properties view.
- In XAML worden enkel eigenschappen bewaard die niet de “standaard” (default) waarde hebben.
- Stel steeds best een “Name” in voor elke control (zie ook verder waarom)



Controls en hun eigenschappen (2)

- Via de properties view vind je alle mogelijke eigenschappen terug en kan je ze desgewenst eenvoudig aanpassen.

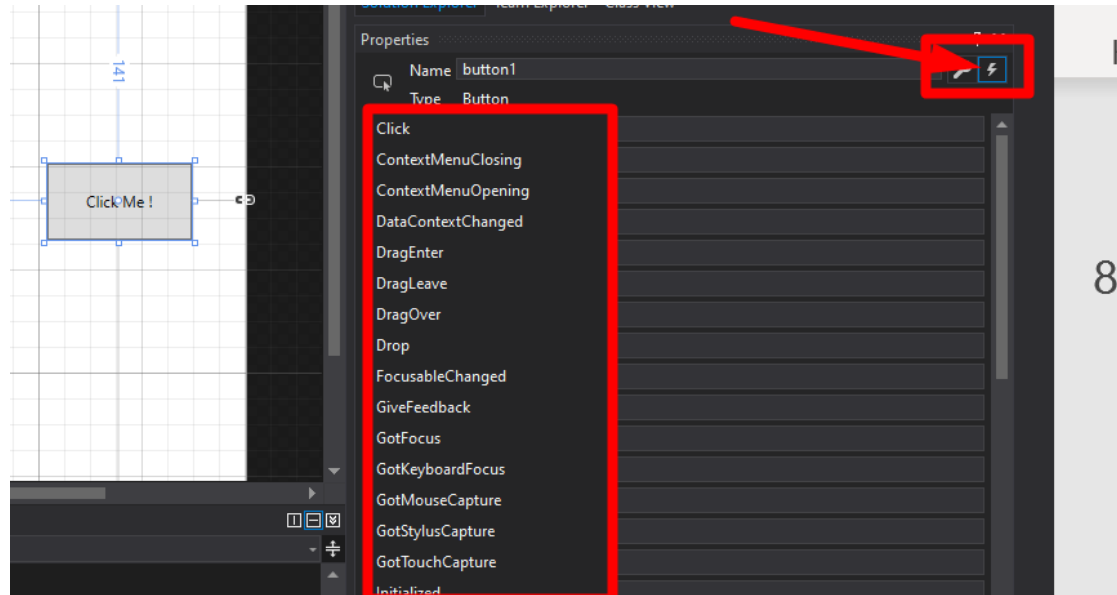


Gebeurtenissen → Event driven programming

- In een console applicatie gebeurt alle invoer(ReadLine) sequentieel (na elkaar) en bepaalt de developer de volgorde. (bv. eerst Geef je naam, je geb. datum, je gewicht, je lengte,...)
- In een WPF applicatie kan de gebruiker verschillende volgordes hanteren en eerder zelf bepalen hoe hij/zij de gegevens ingeeft.
- Als je in een WPF applicatie op de hoogte wil gesteld worden van een gebeurtenis moet je inschrijven op “events”. Men spreekt dan ook over “**Event driven programming**”

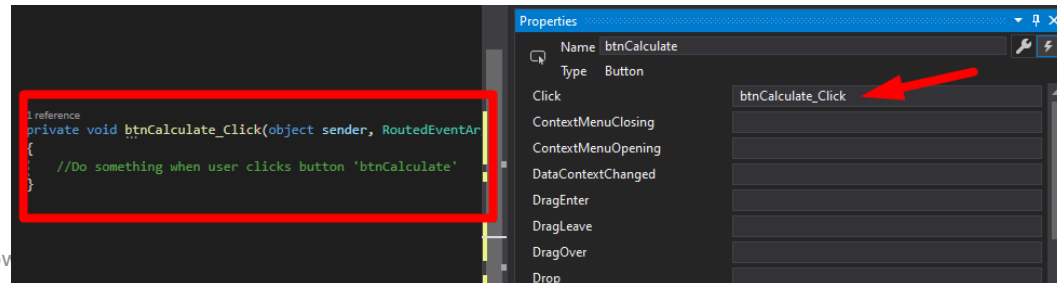
Events (2)

- Via de properties view krijg je een overzicht van alle mogelijke events voor de geselecteerde control.



Events (3)

- “**Inschrijven**” op een event kan door te dubbelklikken in het vakje naast het event.
- Bijvoorbeeld voor een “button” is het meest voor de hand liggende het “**click**” event (de gebruiker drukt op de knop)
- Er wordt vervolgens **automatisch** een “methode” aangemaakt.
- In deze methode kan je de **c# code** schrijven bij dit event.



Console.ReadLine & Console.WriteLine ???

- Vermits er geen “**Readline**” bestaat in WPF, hoe komen we dan wel aan de ingevoerde gegevens ?
 - ✓ Door de “**Text**” **property** van de **Textbox** uit te lezen
- Vermits er geen “**Writeline**” bestaat, hoe tonen we dan het resultaat van onze berekening enz.. ?
 - ✓ Door de “**Text**” **property** van het **TextBlock** in te stellen.

Uitgewerkt voorbeeld: Wachtwoord generator

The image displays a WPF application for a password generator. On the left, the UI is shown with a grid layout containing a text input labeled 'Geef lengte:', a button labeled 'Berekenen', and a text area for the password. Red arrows point from the UI elements to the code: one from the 'Geef lengte:' label to the `tbLengte` TextBox in the XAML, another from the 'Berekenen' button to the `btnCalculate_Click` method, and a third from the password display area to the `txtWachtwoord` TextBlock in the XAML.

UI Elements:

- Geef lengte:
- Berekenen
- Hier komt het wachtwoord

XAML Code (TextBlock):

```
<Grid x:Name="WPFDemo" Height="450" Width="800">
    <Grid>
        <Button x:Name="btnCalculate" Content="Berekenen" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="139,24,0,0" TextWrapping="Wrap" />
        <TextBox x:Name="tbLengte" HorizontalAlignment="Left" Margin="139,24,0,0" TextWrapping="Wrap" />
        <TextBlock x:Name="txtWachtwoord" HorizontalAlignment="Left" Margin="139,61,0,0" />
    </Grid>
</Grid>
```

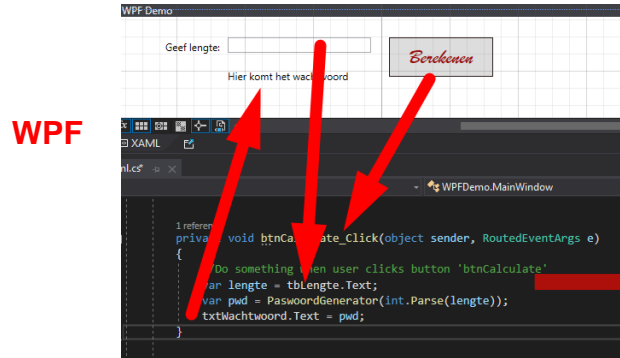
C# Code (btnCalculate_Click):

```
private void btnCalculate_Click(object sender, RoutedEventArgs e)
{
    //Do something when user clicks button 'btnCalculate'
    var lengte = tbLengte.Text;
    var pwd = PaswoordGenerator(int.Parse(lengte));
    txtWachtwoord.Text = pwd;
}

/// <summary>
/// Paswoord generator uit de oefeningen van "Zie Scherp", H7
/// </summary>
/// <param name="lengte"></param>
/// <returns></returns>
1 reference
static string PaswoordGenerator(int lengte)
{
    string resultaat = "";
    Random r = new Random();
    for (int i = 0; i < lengte; i++)
    {
        switch (r.Next(0, 3))
        {
            case 0: //cijfer
                resultaat += r.Next(0, 10);
                break;
            case 1: //kleine letters
                resultaat += (char)r.Next('a', 'z' + 1);
                break;
            case 2: //hoofdletters
                resultaat += (char)r.Next('A', 'Z' + 1);
                break;
        }
    }
    return resultaat;
}
```

Presentatie (invoer/uitvoer) vs. Berekeningen

- Tracht steeds de invoer/uitvoer volledig te scheiden van de eigenlijke berekeningslogica.
- Invoer/uitvoer (specifieke code voor WPF of console) versus berekeningslogica (100% herbruikbaar)
- De berekeningslogica mag dus nooit bv. `Console.ReadLine`, `Console.WriteLine`,...of WPF zaken bevatten !



Console

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Welkom bij de Wachtwoord generator. geef een lengte in voor de lengte van een nieuw wachtwoord");
        int lengte = int.Parse(Console.ReadLine());

        string wachtwoord = PaswoordGenerator(lengte);

        Console.WriteLine($"Het wachtwoord is {wachtwoord}");
    }
}
```

Berekeningslogica

```
/// <summary>
/// Paswoord generator uit de oefeningen van "Zie Scherp", H7
/// </summary>
/// <param name="lengte"></param>
/// <returns></returns>
1 reference
static string PaswoordGenerator(int lengte)
{
    string resultaat = "";
    Random r = new Random();
    for (int i = 0; i < lengte; i++)
    {
        switch (r.Next(0, 3))
        {
            case 0: //cijfer
                resultaat += r.Next(0, 10);
                break;
            case 1: //kleine letters
                resultaat += (char)r.Next('a', 'z' + 1);
                break;
            case 2: //hoofdletters
                resultaat += (char)r.Next('A', 'Z' + 1);
                break;
        }
    }
    return resultaat;
}
```

