

Debugging

Programming Basics

1	JE APPLICATIE DEBUGGEN	3
1.1	Breakpoints	3
1.1.1	Breakpoints plaatsen	3
1.1.2	Breakpoints verfijnen	5
1.2	Step Into, Step Over en Step Out	5
1.2.1	Variabelen inspecteren	6
	Met de muiscursor	6
	Het Locals venster	7
	De Watch vensters	7

1 JE APPLICATIE DEBUGGEN

Tijdens het ontwikkelen van programma's zal je tal van problemen moeten oplossen. Eerst en vooral moet je correcte code leren schrijven zodat je geen **compile-time errors** veroorzaakt. Anderzijds moet je **run-time fouten** kunnen oplossen. Het maken van een try/catch blok (zie hfst 8) is niet altijd de oplossing.

Stel dat je een try/catch blok schrijft waarin een netwerkverbinding wordt geopend die een webpagina download. Het wil maar niet lukken; je krijgt steeds een `NullReferenceException`. Zo'n run-time fout wordt meestal veroorzaakt door jezelf en moet door jou worden opgelost zonder nood aan een try/catch blok.

In zo'n geval moet je gaan debuggen.

Statistieken wijzen uit dat een programmeur voor elke 1 000 lijnen code gemiddeld 70 bugs introduceert. Het oplossen van een bug duurt gemiddeld 30 maal langer dan het schrijven van die code. Een programmeur zal gemiddeld meer dan 60% van zijn tijd doorbrengen aan het debuggen van een applicatie. Efficiënt leren debuggen is dus onontbeerlijk!

1.1 BREAKPOINTS

Een breakpoint is een door jou bepaalde locatie in de code waar de uitvoering van de applicatie zal pauzeren. Tijdens zo'n *break* kan je heel wat hulpmiddelen inzetten om fouten op te sporen.

1.1.1 BREAKPOINTS PLAATSEN

Je plaatst een breakpoint door in de donkere kantlijn van je codevenster te klikken zodat er een bordeauxrode bol verschijnt. De uitvoering zal dan stoppen vlak voor die instructie, die eveneens rood wordt ingekleurd.

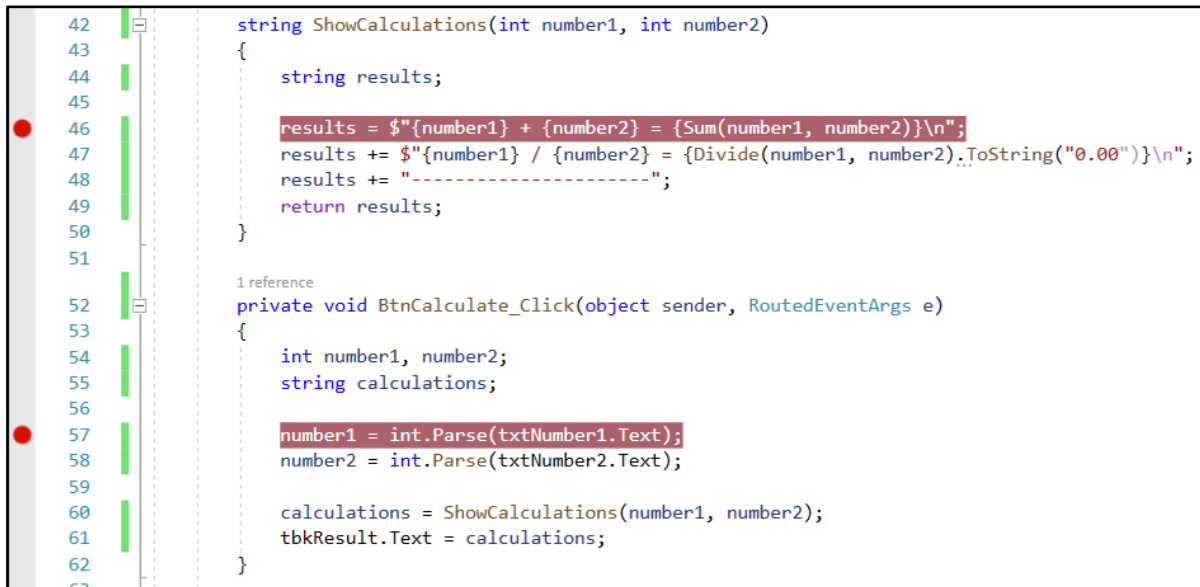


Code repository

Download de oefening Debugging.Wpf via

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

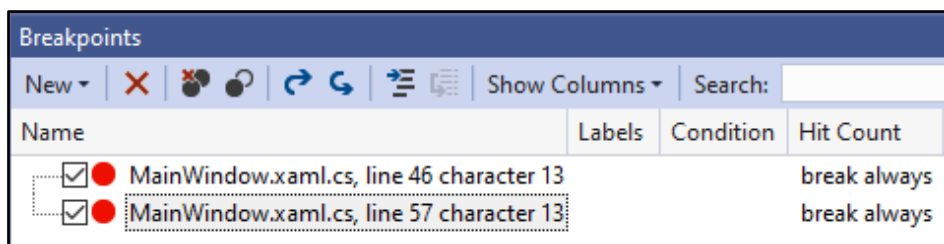
1 Plaats enkele breakpoints, op de volgende plaatsen:



```
42 string ShowCalculations(int number1, int number2)
43 {
44     string results;
45
46     results = $"{number1} + {number2} = {Sum(number1, number2)}\n";
47     results += $"{number1} / {number2} = {Divide(number1, number2).ToString("0.00")}\n";
48     results += "-----";
49     return results;
50 }
51
52 1 reference
53 private void BtnCalculate_Click(object sender, RoutedEventArgs e)
54 {
55     int number1, number2;
56     string calculations;
57     number1 = int.Parse(txtNumber1.Text);
58     number2 = int.Parse(txtNumber2.Text);
59
60     calculations = ShowCalculations(number1, number2);
61     tbkResult.Text = calculations;
62 }
63
```

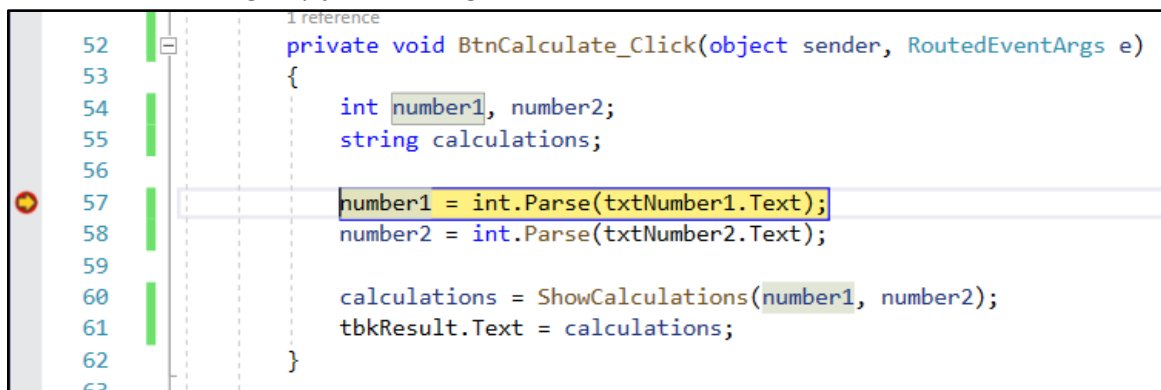
Om een breakpoint te verwijderen die je niet langer nodig hebt klik je nogmaals in de kantlijn, op de bol.

2 Open het Breakpoints venster in Visual Studio, via **Debug → Windows → Breakpoints**.
Je vindt er alle breakpoints in de broncode.

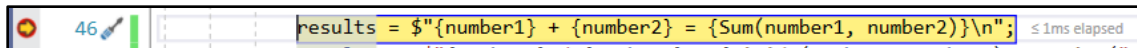


- Om een breakpoint tijdelijk te **deactiveren**, zet je het betreffende vinkje uit.
- Om een breakpoint te **verwijderen**, rechtsklik je erop en kies je voor **Delete**.

3 Voer de applicatie uit in debug mode, via F5, de Start knop of via **Debug → Start Debugging**.
Wanneer je nu klikt op 'Voer bewerkingen uit', merk je dat je terug naar het codescherm wordt geleid. Visual Studio stopt vlak vóór de code van het breakpoint wordt uitgevoerd en geeft de locatie aan met een gele pijl en arcering.



4 Klik op **Continue** of druk op **F5** om de uitvoering verder te zetten.
De gele cursor springt nu naar het volgende breakpoint dat geraakt wordt.



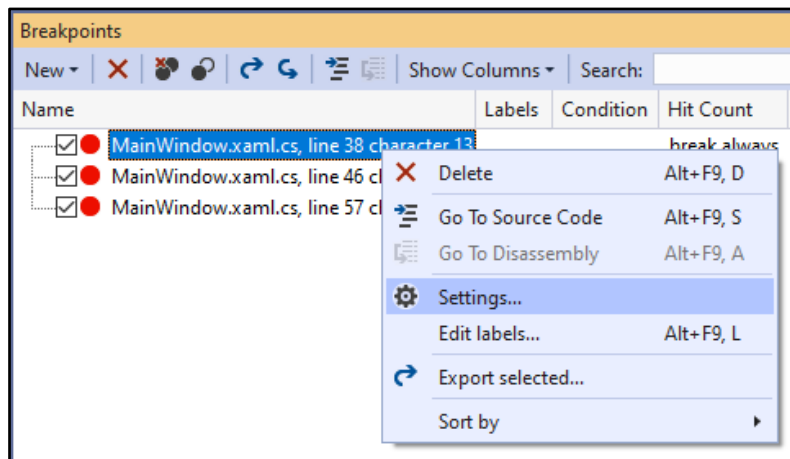
- 5 Druk nogmaals op **Continue** of **F5** om de uitvoering verder te zetten. Er worden geen breakpoints meer geraakt en de applicatie wordt opnieuw actief en bestuurbaar.

1.1.2 BREAKPOINTS VERFIJNEN

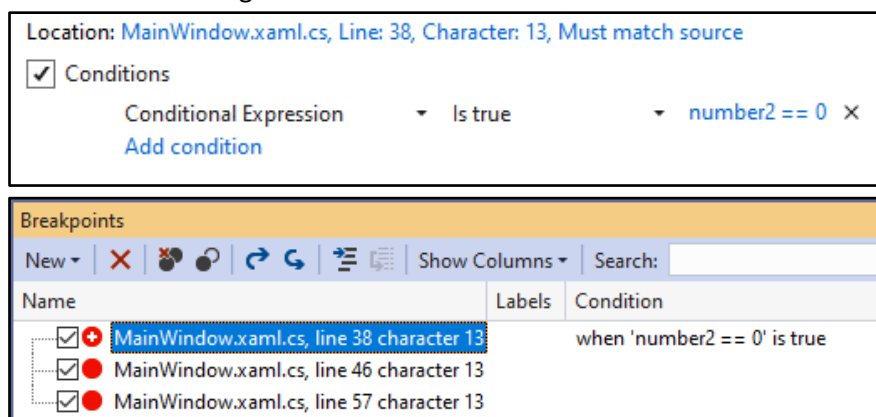
Het kan gebeuren dat de onderbreking van de code enkel wil laten gebeuren in bepaalde gevallen. Stel dat we willen volgen wat er gebeurt als getal 2 gelijk is aan 0 bij het volgende breakpoint:

```
float Divide(int number1, int number2)
{
    float quotient;
    quotient = number1 / (float)number2;
    return quotient;
}
```

In het snelmenu van het Breakpoints window kunnen we kiezen voor 'Settings':



Daarna kun je een voorwaarde ingeven om de code te onderbreken.



De code wordt nu inderdaad enkel onderbroken als *number2* gelijk is aan 0.

1.2 STEP INTO, STEP OVER EN STEP OUT

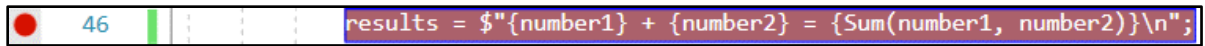
Tijdens de break van een applicatie kan je bepalen waarheen de gele cursor moet springen. Er zijn verschillende opties:

- Doorgaan tot aan het volgende breakpoint: **Continue** of **F5**
- De huidige instructie uitvoeren en wachten: **Step Into**, **Step Over** en **Step Out**.

Je vindt deze opties bovenaan het hoofdvenster van Visual Studio in debug modus:



- 6 Plaats een breakpoint in lijn 46



- 7 Klik op de knop en wacht tot het breakpoint geraakt wordt.
- 8 Druk op **Step Over** of **F10**. Dit is de meeste gebruikte optie.
De gele cursor voert de huidige instructie uit en pauzeert op de instructie eronder.
Druk op F5 om de code verder te laten uitvoeren.
- 9 Klik nogmaals op de knop en wacht tot breakpoint geraakt wordt.
- 10 Druk op **Step Into** of **F11**.
De gele cursor springt naar de methode 'Sum' en pauzeert op de eerste accolade.
Met F10 of F11 kun je nu de code binnen deze methode *Sum* stap voor stap laten uitvoeren.
- 11 Klik nu op de **Step Out** knop of **Shift + F11**. De cursor springt terug naar het statement dat de call naar de methode heeft uitgevoerd.

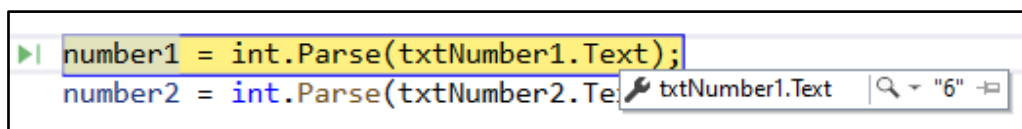
1.2.1 VARIABELEN INSPECTEREN

Tijdens het navigeren van een gepauzeerde applicatie kan je de waarden van variabelen inspecteren. Dit is een bijzonder krachtig hulpmiddel voor het opsporen van fouten. Er zijn twee manieren om dit te doen; door er met de muiscursor over te zweven, of met een hulpvenster.

- 12 Plaats een breakpoint op regel 57.
- 13 Voer een som uit en wacht tot het breakpoint geraakt wordt.

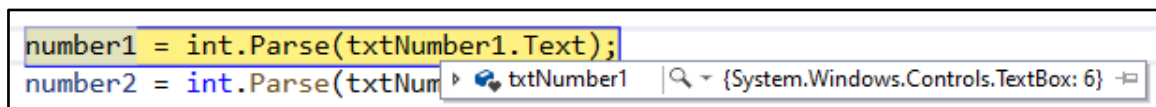
MET DE MUISCURSOR

- 14 Zweef met de muiscursor over de Text property van de Textbox instanties.
Je merkt telkens een tooltip die de waarde van de variabele weergeeft.

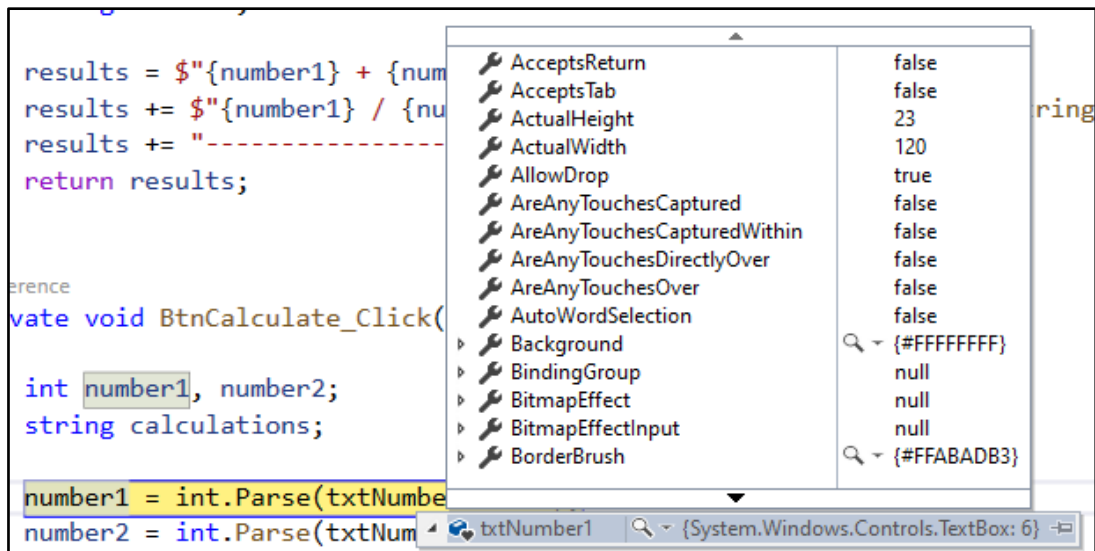


Bij een variabele die een complex datatype heeft wordt de volledige naam van het type getoond. Dat is het geval voor bijvoorbeeld de variabele `txtNumber1`, dat van het type `TextBox` is.

- 15 Zweef met de muiscursor over de variabele `txtNumber1` en bekijk de tooltip.



- 16 Bekijk de Properties van deze variabele door het pijltje in de tooltip open te klappen.



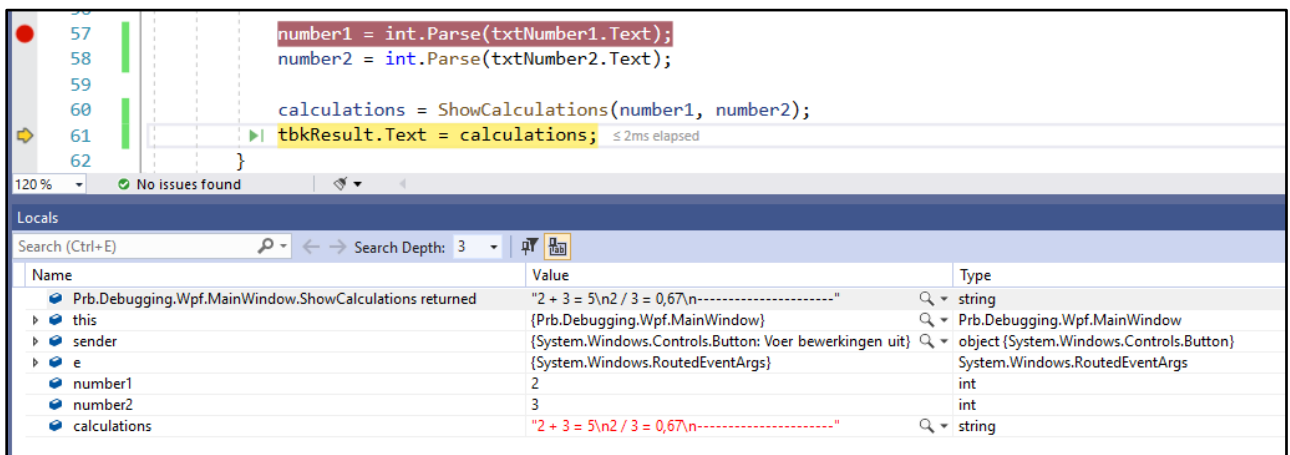
HET LOCALS VENSTER

In het Locals venster worden alle variabelen in de huidige scope (van de methode) getoond. Je kan het venster zichtbaar maken via **Debug → Windows → Locals**.

17 Maak een som met correcte numerieke waarden en gebruik **Step Over** tot je op de regel terecht komt waar de Text property van de TextBox wordt opgevuld met de waarde uit de variabele calculations.

18 Open het **Locals** venster en bekijk de waarden.

Je vindt er drie kolommen: de naam van de variabele, de waarde ervan en het type. De meest recent aangepaste variabele heeft een rode kleur gekregen. Complexe typen zijn ook hier uitklapbaar zodat je de properties ervan kan inspecteren.



DE WATCH VENSTERS

Er zijn meerdere **Watch** vensters beschikbaar waarin je zelf variabelen kan plaatsen om die in het oog te houden. Dit zijn veruit de meest gebruikte Debug vensters die er zijn. Er zijn 4 watch vensters, respectievelijk Watch 1, Watch 2, Watch 3 en Watch 4 genaamd. Tijdens het debuggen zou er al eentje zichtbaar moeten zijn.

Om een Watch venster te openen, ga je naar **Debug → Windows → Watch → Watch 1 ... 4**

19 Maak een som met correcte numerieke waarden en wacht tot het breakpoint geraakt wordt.

20 Open het **Watch 1** venster. Deze is standaard leeg.

Je kan een te inspecteren variabele toevoegen op twee manieren:

- Tik de naam van de variabele in de Name kolom en druk op enter.
- Rechtsklik op de variabele in het codevenster en kies **Add Watch**.

21 Door tijdens het debuggen met de rechtmuisknop op een variabele of object te klikken, kun je in het snelmenu kiezen voor Add Watch. Je kunt ook gewoon in de Name-kolom tikken wat je wil 'watchen'.

22 Voeg de volgende variabelen toe aan het Watch 1 venster:

- `tbkResult.Text`
- `calculations`
- `number2`

Name	Value	Type
<code>tbkResult.Text</code>	""	string
<code>calculations</code>	null	string
<code>number2</code>	0	int

23 Gebruik **Step Over** tot je op de regel terechtkomt waar de Text property van de TextBox wordt opgevuld. Merk op hoe de waarden wijzigen tijdens de stapsgewijze uitvoering van je code. Hier kan je zeer veel uit leren om de oorzaak van fouten te achterhalen.

The screenshot shows the Visual Studio IDE with a breakpoint set at line 61 of the code. The code window displays the following code:

```
57 number1 = int.Parse(txtNumber1.Text);
58 number2 = int.Parse(txtNumber2.Text);
59
60 calculations = ShowCalculations(number1, number2);
61 tbkResult.Text = calculations;
62 }
```

The Watch 1 window is open, showing the following variables and their values:

Name	Value	Type
<code>tbkResult.Text</code>	""	string
<code>calculations</code>	"2 + 3 = 5\n2 / 3 = 0,67\n-----"	string
<code>number2</code>	3	int

Je kunt de waarden in de kolom 'Value' ook overtikken. Dit kan interessant zijn om de effecten na te gaan bij een andere waarde dan de actuele.

Debuggen is een taak die evenwaardig is aan het schrijven van correcte code. Maak er een goede gewoonte van tijdens de ontwikkeling van je applicatie zodat je efficiënt met de hulpmiddelen leert werken. Op deze manier moet je niet gissen wat de oorzaak van een fout is, maar kan je dit zonder meer achterhalen.



Meer informatie

[MS: Visual Studio - Getting started with the debugger](#)

