



PROGRAMMEERTECHNIEKEN EN TESTEN–UI TESTING

MATTHIAS DRUWÉ
SAM VAN BUGGENHOUT



Wat is GUI-testing?

- Bij **UI-testing** wordt het functioneren van de **grafische gebruikersinterface** (Graphical User Interface) getest
- Hierbij wordt de **status** (kleur, beschikbaarheid, zichtbaarheid, ...) van de verschillende **componenten** (buttons, checkboxes, radiobuttons, textboxes, labels, ...) van de GUI gecontroleerd
- Een GUI-applicatie bevat vaak twee **"soorten" logica**:
 - **Business Logica**: berekeningen, validatie-regels, verwerking van gegevens, ...
 - **GUI-logica**: wanneer wordt een bepaalde component zichtbaar? Wanneer moet een specifieke knop enabled/disabled worden?
- Bij UI-testing, wordt **de applicatie getest zoals een gewone gebruiker dit zou doen**



Wat wordt getest?

- Bij GUI-testen wordt de **status** van de **GUI-controls** getest in specifieke omstandigheden
- Vaak is de **status** van een GUI-control **afhankelijk** van de status van een **andere GUI-control** (bv.: *de knop "bereken" is enkel beschikbaar indien de velden "lengte" en "gewicht" geldige getallen bevatten*)
- Voorbeelden van **"status"** die we kunnen testen:
 - **Aanwezigheid:** zijn alle controls aanwezig?
Bv.: is de knop "afmelden" aanwezig nadat de gebruiker zich heeft aangemeld?
 - **Zichtbaarheid:** bv.: zijn alle labels met foutmeldingen standaard verborgen? Zijn de labels met foutmeldingen zichtbaar wanneer de gebruiker foutieve invoer heeft ingebracht?
 - **Enabled/disabled:** bv.: zijn alle knoppen die enabled moeten zijn ook effectief enabled? Knop mag enkel enabled zijn indien alle velden correcte invoer bevatten → is dit het geval?
 - **Uiterlijke kenmerken:** positie, kleur, grootte, ...



Wat wordt getest?

- **Opgelet!** Zorg voor een duidelijk **onderscheid** tussen **business logica** en **GUI-logica**!
- **Voorbeeld:** *je maakt een BMI-calculator applicatie met twee invulvelden: één voor de lengte van de persoon en één voor het gewicht. Daarnaast bevat de GUI een knop om de berekening uit te voeren. Wanneer de gebruiker op deze knop klikt, wordt het berekende BMI in een label weergegeven. De knop mag echter enkel beschikbaar zijn, indien de gebruiker twee geldige getallen in de twee invulvelden heeft ingevuld.*

Business Logica	GUI-logica
Word het correcte BMI berekend voor de gegeven lengte en gewicht? (m.a.w. werd de formule voor de BMI-berekening correct geïmplementeerd?)	Is de knop “bereken” enkel enabled indien de gebruiker twee geldige getallen heeft ingevoerd in de tekstvelden voor lengte en gewicht?
→ Testen via Unit/Integration-test	→ Testen via UI-test



Wat wordt getest?

- Business Logica en GUI-logica worden op een **andere manier** getest
- Zorg er dus voor dat beide delen **onafhankelijk** van elkaar kunnen getest worden!
- Dit illustreert het belang van een duidelijke **scheiding** tussen **GUI-code** en **business logica** (bv.: via MVVM, MVC, MVP, ...)
- *Voorbeeld BMI-calculator:*
 - Maak een aparte klasse *BMICalculator*, waarin je de berekening van het BMI centraliseert
 - In de *code behind* van je WPF-applicatie gebruik je deze klasse voor de berekening van het BMI
 - test de logica van de klasse *BMICalculator* m.b.v. Unit-tests
 - test de “logica” in de *code behind* van je code m.b.v. GUI-tests



GUI testen in WPF

- Het is mogelijk om deze GUI testen **manueel** uit te voeren, **MAAR**:
 - Dit is zeer tijdrovend → duur !
- Een betere optie is om automatische testen te schrijven die gebruik maken van een **automation tool**
- De automation tool zal **acties** (bv.: kliks, toetsenbord-aanslagen, etc.) **simuleren** in de applicatie
- Via de automation tool, kan je vervolgens de **status** van de GUI-componenten (zichtbaarheid, positie, enabled/disabled, tekst, ...) **opvragen**
- In deze cursus, zullen we gebruikmaken van de automation tool **Appium**

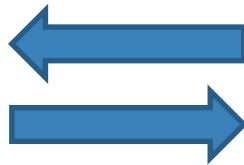


GUI testen in WPF

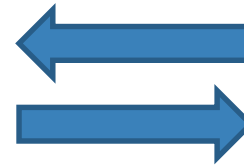
- De testcode “communiceert” met Appium om een actie uit te voeren in de WPF-applicatie
- Appium communiceert met de WPF-applicatie via de *WinAppDriver*, die het mogelijk maakt om de applicatie “remote” te besturen
- Voor het opvragen van gegevens (bv.: de zichtbaarheid van een knop), wordt de omgekeerde weg afgelegd



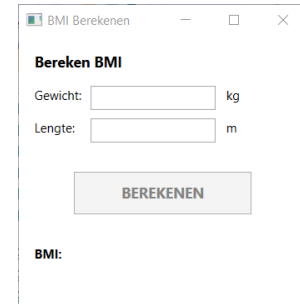
Test Script



Appium-server



WinAppDriver






WPF-applicatie



GUI testen in WPF

- Om gebruik te kunnen maken van Appium, moet je de volgende zaken installeren:
 - WinAppDriver (<https://github.com/microsoft/WinAppDriver/releases>)
 - Zoek de laatste versie en selecteer het **.MSI**-bestand

▼ Assets 3

 WindowsApplicationDriver.msi	3.72 MB
 Source code (zip)	
 Source code (tar.gz)	

(vervolledig de installatie)



GUI testen in WPF

- Schrijf je GUI-testen in een **apart project** (type: *Class Library (.NET Framework)*)
 - Houd je aan de afgesproken conventies: bv.: "*WpfBMI.UITests*"
- Installeer de volgende NuGet-packages in het **testproject**:
 - NUnit
 - NUnitTestAdapter
 - Appium.WebDriver



Appium.WebDriver by Appium Committers
Selenium Webdriver extension for Appium.



NUnit by Charlie Poole, Rob Prouse
NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

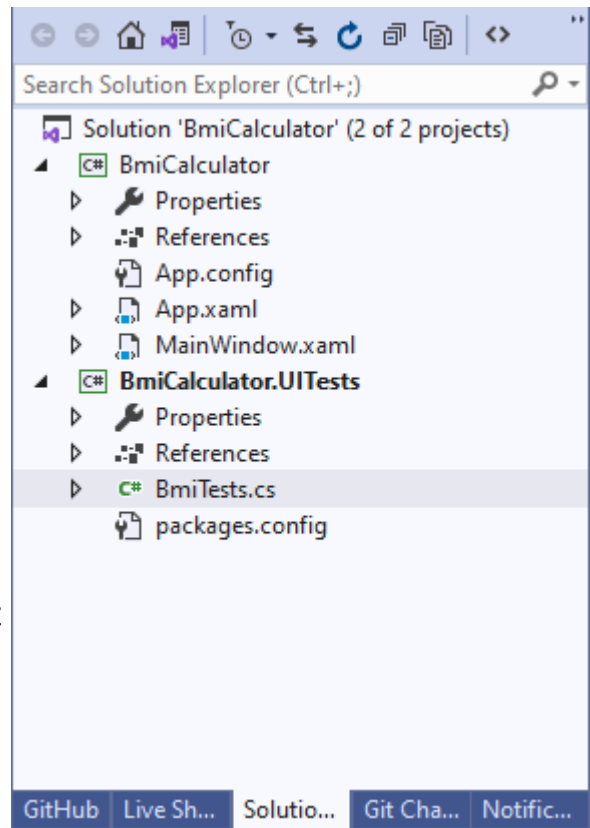


NUnit3TestAdapter by Charlie Poole, Terje Sandstrom
NUnit 3 adapter for running tests in Visual Studio. Works with NUnit 3.x, use the NUnit 2 adapter for 2.x tests.



GUI testen in WPF

- We gaan verder met het voorbeeld van de *BMI-calculator*
- Maak in het testproject een nieuwe klasse waarin je de GUI-testen schrijft
- Zorg ervoor dat alle benodigde NuGet-packages geïnstalleerd zijn
- Om de GUI-testen uit te voeren, moet de Appium-service actief zijn. Deze vind je terug in **C:\Program Files (x86)\Windows Application Driver\WinAppDriver.exe**. Dubbelklik op het exe-bestand en laat het console-venster geopend!





GUI testen in WPF

- Om vanuit je testcode te communiceren met Appium, die je een **"sessie"** aan te maken. Alle commando's worden via deze sessie verstuurd.
- Voor het maken van een sessie heb je twee zaken nodig:
 - De **URL** waarop de Appium-service draait
(deze vind je terug in het console-venster van de Appium-service, en is meestal <http://127.0.0.1:4723>)
 - Het pad naar het **.exe-bestand** van de applicatie die je wilt testen
(het .exe-bestand van een project, vind je terug in de map **<projectnaam>/bin/Debug/<projectnaam>.exe**)

```
C:\Program Files (x86)\Windows Application Driver\WinAppDriver.exe
Windows Application Driver listening for requests at: http://127.0.0.1:4723/
Press ENTER to exit.
```



GUI testen in WPF

- Maak voor deze gegevens twee **constanten** in je klasse:

```
1 [TestFixture]
2 public class BmiTests
3 {
4     private const string windowsApplicationDriverUrl = "http://127.0.0.1:4723";
5     private const string wpfAppId = @"D:\Repos\OdiseePTT\Voorbereiding\CodeExamples_UITesting\BmiCalculator\BmiCalculator\bin\Debug\BmiCalculator.exe"; // Zelf aan te passen naar eigen path
6     ...
7 }
```

- Maak vervolgens een tweede member aan voor het bijhouden van het **sessie-object** zelf:

```
1 [TestFixture]
2 public class BmiTests
3 {
4     ...
5     private WindowsDriver<WindowsElement> session;
6 }
```



GUI testen in WPF

- We schrijven vervolgens de code om het **sessie-object** te **instantiëren**
- Het aanmaken van een sessie, dient **vóór** het uitvoeren van **elke test** te gebeuren
- Om ervoor te zorgen dat de sessie één keer aangemaakt wordt voor elke test, schrijven we een **set-up methode** die het sessie-object instantieert. Het attribuut **SetUp**, geeft aan dat deze methode één keer moet uitgevoerd worden vóór elke tests in de klasse wordt uitgevoerd.

```
1 [SetUp]
2 public void SetUp()
3 {
4     if(session == null)
5     {
6         AppiumOptions appiumOptions = new AppiumOptions();
7         appiumOptions.AddAdditionalCapability("app", wpfAppId);
8         appiumOptions.AddAdditionalCapability("deviceName", "WindowsPC");
9         session = new WindowsDriver<WindowsElement>(new Uri(windowsApplicationDriverUrl), appiumOptions);
10    }
11 }
```





GUI testen in WPF

- De volledige set-up code wordt:

```
1 [TestFixture]
2 public class BmiTests
3 {
4     private const string windowsApplicationDriverUrl = "http://127.0.0.1:4723";
5     private const string wpfAppId = @"D:\Repos\OdiseePTT\Voorbereiding\CodeExamples_UITesting\BmiCalculator\BmiCalculator\bin\Debug\BmiCalculator.exe"; // Zelf aan te passen naar eigen path
6
7     private WindowsDriver<WindowsElement> session;
8
9     [SetUp]
10    public void Setup()
11    {
12        if(session == null)
13        {
14            AppiumOptions appiumOptions = new AppiumOptions();
15            appiumOptions.AddAdditionalCapability("app", wpfAppId);
16            appiumOptions.AddAdditionalCapability("deviceName", "WindowsPC");
17            session = new WindowsDriver<WindowsElement>(new Uri(windowsApplicationDriverUrl), appiumOptions);
18        }
19    }
20 }
```



GUI testen in WPF

- Nadat een test uitgevoerd is, moet de sessie opnieuw **gesloten** worden
- Aangezien we **voor elke test** een **nieuwe sessie** aanmaken, moet deze ook **na elke test** opnieuw **gesloten** worden
- We schrijven hiervoor een **CleanUp-methode** die de sessie afsluit. Met behulp van het attribuut **TearDown**, gaaf je aan dat deze methode één keer moet uitgevoerd worden, na elke test.

```
1  [TearDown]
2  public void TearDown()
3  {
4      if (session != null)
5      {
6          session.Close();
7          session.Quit();
8          session = null;
9      }
10 }
```



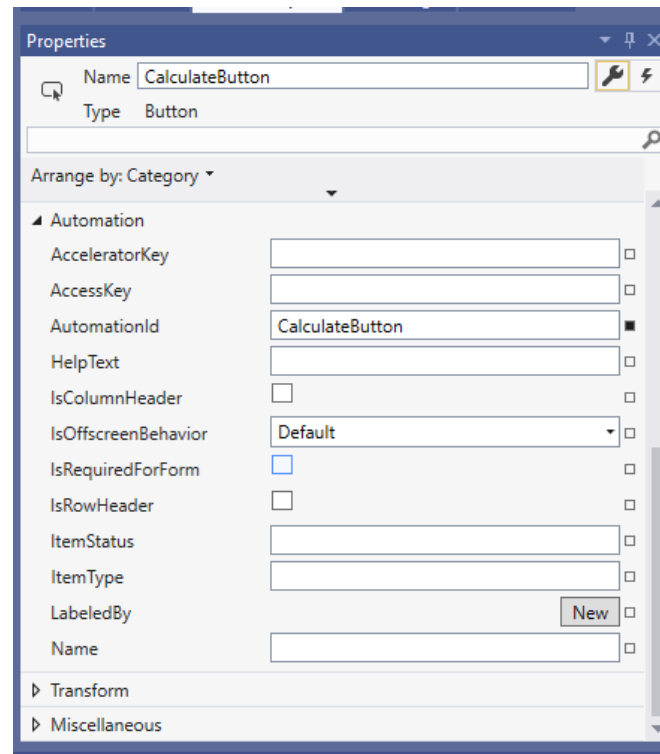
GUI testen in WPF

- De setup en cleanup code is nu volledig, vervolgens kan je testen schrijven om de gebruikersinterface te testen
- Maak voor elke testcase een aparte test!
- De opzet voor een GUI-test is gelijkaardig aan die van een “normale” unit-test
- We schrijven de eerste test voor de BMI-calculator applicatie:
 - *Indien de gebruiker het scherm opstart, verwachten we dat de “BEREKENEN”-knop gedisable is*



GUI testen in WPF

- In de test, willen we de beschikbaarheid (Enabled-property) van de “BEREKENEN”-knop nagaan
- We hebben dus een manier nodig om in de testcode naar deze knop te **verwijzen**
- Geef de knop (via de XAML-designer) van de BMI-calculator een “AutomationId”. Het “AutomationId” is een **unieke** identifier (“naam”) die je aan de knop kunt geven, zodat je vanuit je testcode naar deze control kunt verwijzen met behulp van dit id





GUI testen in WPF

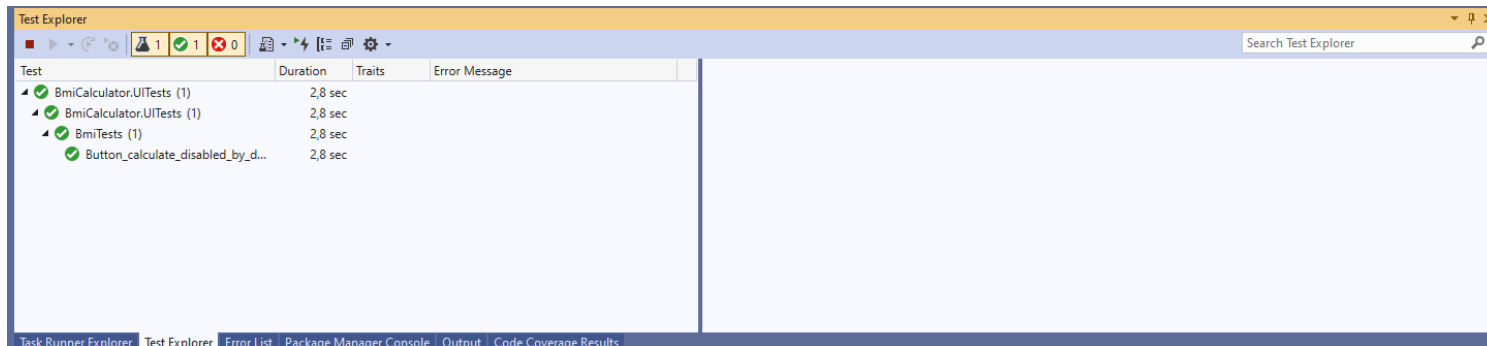
- In de testcode, kan je nu met behulp van de methode ***FindElementByAccessibilityId*** verwijzen naar de knop op het scherm
- **Opgelet:** het return-type is ***WindowsElement***, **NIET *Button*!!!**
- Van het *WindowsElement* (dat de button voorstelt), vragen we de waarde van de *Enabled*-property op, en gaan na of deze *false* is, m.b.v. een assertion

```
1 [Test]
2 public void Button_calculate_disabled_by_default()
3 {
4     WindowsElement btnCalculate = session.FindElementByAccessibilityId("CalculateButton");
5     Assert.IsFalse(btnCalculate.Enabled);
6 }
```



GUI testen in WPF

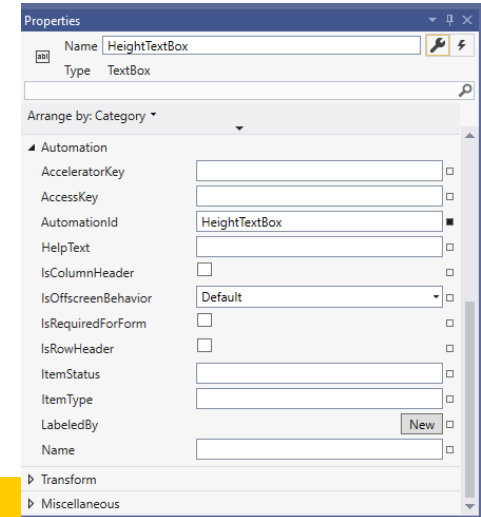
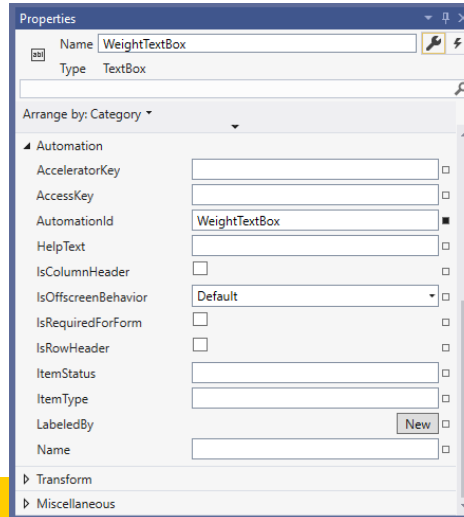
- Voer de test op dezelfde manier uit als bij een “normale” unit-test
- **Zorg ervoor dat de *WinAppDriver*-service loopt!**
- Het resultaat van de test, wordt ook getoond zoals je dat gewend bent





GUI testen in WPF

- We schrijven de code voor een tweede testcase:
 - *Wanneer de gebruiker een geldige waarde voor de lengte en het gewicht heeft ingevuld, is de knop "BEREKENEN" beschikbaar*
- We gaan op dezelfde manier tewerk, we moeten de TextBoxes voor *gewicht* en *lengte* invullen, dus moeten we ook naar deze velden kunnen verwijzen m.b.v. een *AutomationId*





GUI testen in WPF

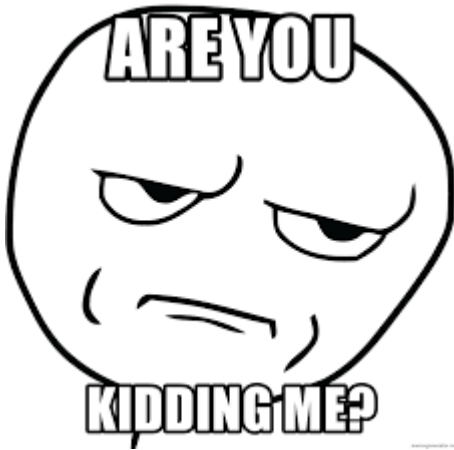
- Met behulp van de ***SendKeys***-methode, kan je een toetsaanslag (of reeks toetsaanslagen) naar de component “versturen” (bv.: een string is een reeks toetsaanslagen van letters, cijfers, ...)

```
1  [Test]
2  public void Button_calculate_enabled_when_filled_in()
3  {
4      //Verzamel referenties naar de GUI-controls => Arrange
5      WindowsElement btnBereken = session.FindElementByAccessibilityId("CalculateButton");
6      WindowsElement txtGewicht = session.FindElementByAccessibilityId("WeightTextBox");
7      WindowsElement txtLengte = session.FindElementByAccessibilityId("HeightTextBox");
8
9      //Vul de TextBox-componenten in => Act
10     txtGewicht.SendKeys("62");
11     txtLengte.SendKeys("1,78");
12
13     //Controleer of BEREKENEN-knop beschikbaar is => Assert
14     Assert.IsTrue(btnBereken.Enabled);
15 }
16
```



GUI testen in WPF

- **Opgelet:** indien je een test hebt die gebruik maakt van het toetsenbord (bv.: de *SendKeys*-methode) moet de keyboard-layout van je toetsenbord **op het moment dat je de test uitvoert** op QWERTY staan!





GUI testen in WPF

- Nieuwe testcase:
 - *als de gebruiker een geldige waarde heeft ingevuld voor het gewicht en de lengte en op de knop "BEREKENEN" klikt, wordt het BMI in een label weergegeven.*
- Met behulp van de **Click**-methode kan je een muisklik op de control uitvoeren
- Om de tekstuele inhoud van een element op te vragen, kan je gebruik maken van de **Text**-property (read-only)
- Om de waarde van het label op te vragen, dien je dit label ook een *AutomationId* te geven



GUI testen in WPF

- De code voor deze test wordt:

```
1  [Test]
2  public void Button_BMI_is_shown_when_calculated()
3  {
4      //Verzamel referenties naar de GUI-controls
5      WindowsElement btnBereken = session.FindElementByAccessibilityId("CalculateButton");
6      WindowsElement txtGewicht = session.FindElementByAccessibilityId("WeightTextBox");
7      WindowsElement txtLengte = session.FindElementByAccessibilityId("HeightTextBox");
8      WindowsElement lblBMI = session.FindElementByAccessibilityId("BMILabel");
9
10     //Vul de TextBox-componenten in
11     txtGewicht.SendKeys("62");
12     txtLengte.SendKeys("1,78");
13
14     //Klik op de knop "BEREKENEN"
15     btnBereken.Click();
16
17     //Controleer dat label niet leeg is
18     Assert.IsNotEmpty(lblBMI.Text);
19 }
```




GUI testen in WPF

- **Opmerking:** merk op dat we hier **NIET** testen of het berekende BMI overeenkomt met het verwachte BMI (m.a.w. we testen **NIET** de correcte werking van de berekening van het BMI), we testen enkel of het **BMI zichtbaar** is in het label
- De **berekening** van het BMI valt onder **Business Logica**, en kan dus getest worden met behulp van **Unit-tests**!





Geavanceerde interacties

- Met behulp van Appium, kunnen we ook **"geavanceerde" interacties** testen, zoals selecties in ListBox-controls
- **Voorbeeld:** *we maken een applicatie met een lijst van talen. Wanneer de gebruiker een taal in de lijst selecteert, wordt deze taal weergegeven in een label. Als de gebruiker zijn/haar selectie ongedaan maakt (control + klik op item), wordt het label opnieuw leeggemaakt.*



Geavanceerde interacties

- Om een item in de ListBox te selecteren, voer je onderstaande stappen uit:
 - Selecteer de ListBox-control (via een AutomationId)
 - Binnen de ListBox, zoek je naar alle *ListBoxItems* (met behulp van de methode ***FindElementsByClassName***)
 - Selecteer het gewenste item uit de *IEnumerable* met *ListBoxItems*, op basis van de index van het element dat je wilt selecteren
 - Zet het object om naar een *WindowsElement* (via casting)
 - Je kunt nu op het element "klikken" m.b.v. de *Click*-methode

```
1 //Selecteer de ListBox op basis van AccessibilityId
2 WindowsElement languageListBox = session.FindElementByAccessibilityId("LanguageListBox");
3
4 //Selecteer alle ListBoxItems binnen deze ListBox
5 ReadOnlyCollection<AppiumWebElement> listItems = languageListBox.FindElementsByClassName("ListBoxItem");
6
7 //Selecteer het eerste element (index: 0) en zet het om naar een WindowsElement
8 WindowsElement eerste = (WindowsElement)listItems[0];
```



Geavanceerde interacties

- De volledige test wordt:

```
1 [Test]
2 public void Selection_visible_in_label()
3 {
4
5     WindowsElement languageListBox = session.FindElementByAccessibilityId("LanguageListBox");
6     WindowsElement firstItem = (WindowsElement)languageListBox.FindElementsByClassName("ListBoxItem")[0];
7
8     firstItem.Click();
9
10    WindowsElement lblSelectie = session.FindElementByAccessibilityId("ChosenLanguageLabel");
11
12    Assert.AreEqual(firstItem.Text, lblSelectie.Text);
13 }
```



Geavanceerde interacties

- We schrijven vervolgens de testcase voor het ongedaan maken van de selectie
- Om een selectie ongedaan te maken, voer je de volgende stappen uit:
 - Houd de control-toets ingedrukt
 - Klik op het geselecteerde item
 - Laat de control-toets opnieuw los
- Om deze drie acties na elkaar uit te kunnen voeren, maken we gebruik van de klasse ***Actions***

```
1 //Deselecteer het eerste item opnieuw
2 Actions actions = new Actions(session);
3 actions.KeyDown(Keys.Control).Click(firstItem).KeyUp(Keys.Control);
4 actions.Perform();
```



Geavanceerde interacties

- De volledige testcode wordt:

```
1  [Test]
2  public void Label_empty_when_selection_cleared()
3  {
4      WindowsElement lstTalen = session.FindElementByAccessibilityId("LanguageListBox");
5      WindowsElement firstItem = (WindowsElement)lstTalen.FindElementsByClassName("ListBoxItem")[0];
6
7      //Selecteer eerste item
8      firstItem.Click();
9
10     //Deselecteer het eerste item opnieuw
11     Actions actions = new Actions(session);
12     actions.KeyDown(Keys.Control).Click(firstItem).KeyUp(Keys.Control);
13     actions.Perform();
14
15     WindowsElement lblSelectie = session.FindElementByAccessibilityId("ChosenLanguageLabel");
16
17     Assert.IsEmpty(lblSelectie.Text);
18 }
```



Geavanceerde interacties

- De selectie van een **ComboBox**, ziet net iets anders uit
- Met behulp van de **pijltoetsen** boven en beneden, kan je door de items in de ComboBox navigeren

```
1 WindowsElement cboTalen = session.FindElementByAccessibilityId("LanguageCombobox");  
2  
3 //Selecteer eerste item  
4 cboTalen.SendKeys(Keys.Down);  
5  
6 //Selecteer tweede item  
7 cboTalen.SendKeys(Keys.Down);
```

- Om de tekstuele inhoud van het geselecteerde item op te vragen, kan je gebruik maken van de *Text*-property



Geavanceerde interacties

● Een compleet voorbeeld van een ComboBox:

```
1 [Test]
2 public void Label_filled_on_combo_selection()
3 {
4
5     WindowsElement cboTalen = session.FindElementByAccessibilityId("LanguageCombobox");
6
7     //Selecteer eerste item
8     cboTalen.SendKeys(Keys.Down);
9
10    //Selecteer tweede item
11    cboTalen.SendKeys(Keys.Down);
12
13    WindowsElement lblSelectie = session.FindElementByAccessibilityId("ChosenLanguageLabel");
14
15    Assert.AreEqual(cboTalen.Text, lblSelectie.Text);
16 }
17
```




Geavanceerde interacties

- **CheckBox** aanvinken en controleren of deze aangevinkt is:

```
1  [Test]
2  public void Checkbox_selection_test()
3  {
4      WindowsElement readCheckBox = session.FindElementByAccessibilityId("ReadCheckBox");
5      readCheckBox.DisableCache();
6
7      //Checkbox aanvinken
8      readCheckBox.Click();
9
10     Assert.IsTrue(readCheckBox.Selected);
11
12     //Checkbox afvinken
13     readCheckBox.Click();
14
15     Assert.IsFalse(readCheckBox.Selected);
16 }
```



Geavanceerde interacties

- **Opmerking:** de ***DisableCache***-methode zorgt ervoor dat de status van de component opnieuw opgehaald wordt, om foutieve waarden te vermijden
- Een andere oplossing is het element **opnieuw opvragen** via de sessie:

```
1  [Test]
2  public void Checkbox_selection_test()
3  {
4      WindowsElement readCheckBox = session.FindElementByAccessibilityId("ReadCheckBox");
5
6      //Checkbox aanvinken
7      readCheckBox.Click();
8
9      readCheckBox = session.FindElementByAccessibilityId("ReadCheckBox");
10
11     Assert.IsTrue(readCheckBox.Selected);
12
13 }
```



Geavanceerde interacties

- Op een gelijkaardige manier, kan je ook **RadioButtons** selecteren en nagaan of een RadioButton al dan niet geselecteerd is:

```
1 [Test]
2 public void RadioButton_selection_test()
3 {
4     WindowsElement maleRadioButton = session.FindElementByAccessibilityId("MaleRadioButton");
5
6     maleRadioButton.Click();
7
8     Assert.IsTrue(maleRadioButton.Selected);
9 }
```



Opmerkingen

- Om de inhoud van een TextBox leeg te maken, kan je gebruik maken van de methode ***Clear***:

```
1 WindowsElement weightTextBox = session.FindElementByAccessibilityId("WeightTextBox");  
2  
3 weightTextBox.SendKeys("62");  
4 weightTextBox.Clear(); //Maak de TextBox leeg
```

- Om na te gaan of een element zichtbaar is op het scherm, kan je van dit element de waarde van property ***Displayed*** opvragen:

```
1 WindowsElement errorLabel = session.FindElementByAccessibilityId("ErrorLabel");  
2  
3 Assert.IsTrue(errorLabel.Displayed);
```