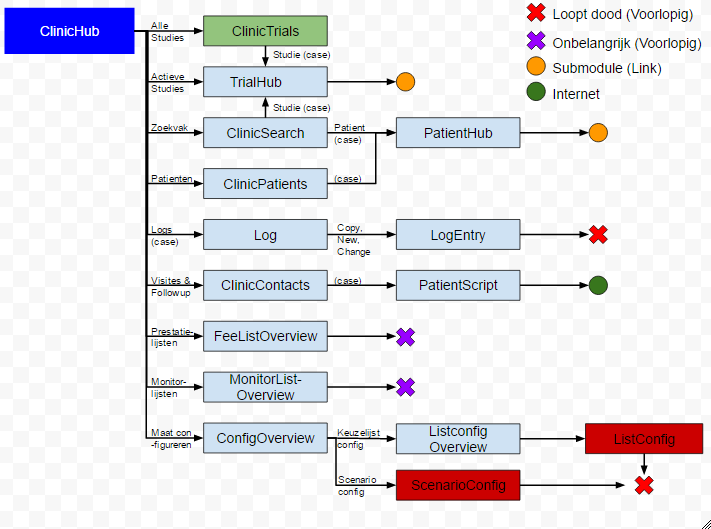
Guidelines

Testing

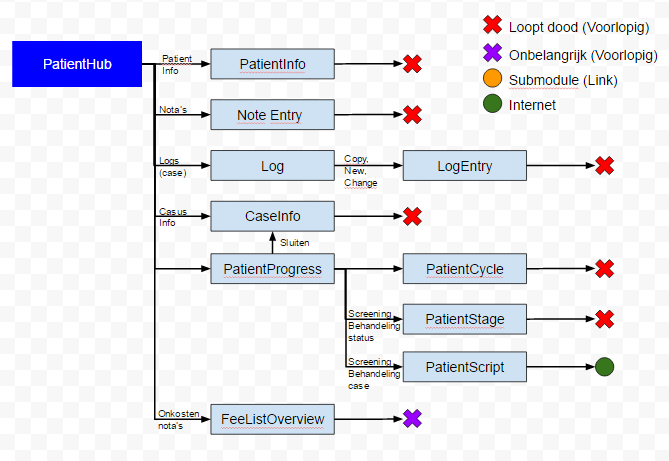
# Hierarchical navigation design

In this section, you can see a hierarchical design of all possible navigations inside Maät, ordered by page. For every page, we have one UI-map and per page, we have a test project for every paradigm on the checklist.

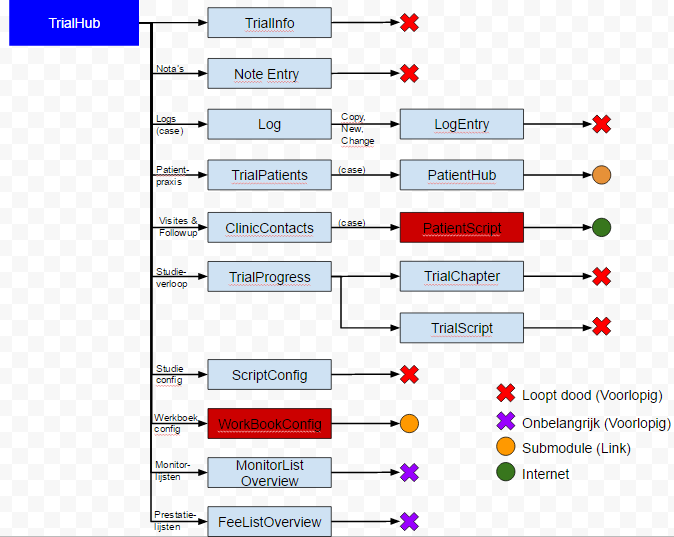
## ClinicHub



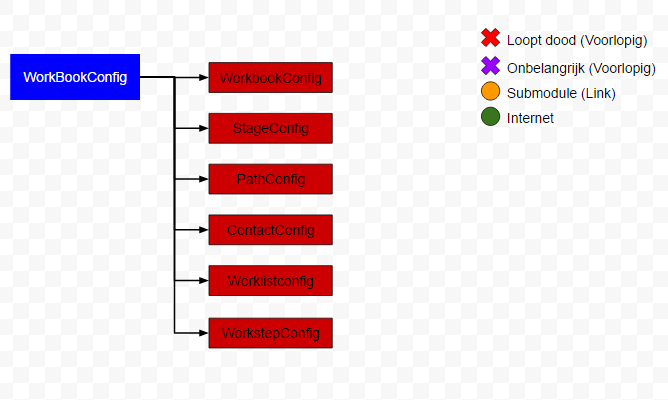
## PatientHub



## TrialHub



## WorkBookConfig



# Method Naming

To maintain the readability of the test project, here are some guidelines on how to name your testmethods

Page name + Paradigm (+ Context) (+ Xaml Name Property) (+ optional info)

Ex: ClinicSearchNavigateStudieListItemState1()

* Page name: ClinicSearch

Page name indicates the page within which your test is situated

* Paradigm: Navigate

Paradigm is the kind of action you are testing for the control/set of controls (bvb navigate, zoom, toggle,...)

* Context: Studie

Indicator of the context in which the control/set of controls you want tot test exist, in this case, the method tests a “Studie-searchresult”

* XamlNameProperty: ListItem

In case of testing one specific control (or set of the same controls), this will indicate the kind of controls tested

* Optional info: State1

this will indicate extra info about the context of your test, in this case, the extra info indicates the stage of the page.

# File Structure

The Clinical Trials application follows a specific map structure for all it’s elements. To keep the file structure readable, we follow the same structure inside our testproject.

Calidos.Maat.CodedUITests

* Screens
  + General UIMap
  + Partial parent .cs file
  + .csv global scenarios
  + Category (clinic,…)
    - .csv categorical scenarios
    - Page (contactspage,….)
      * UImap
      * .csv local scenarios
      * .cs files per category
        + States

All tests regarding the different states of a page

* + - * + Navigation

All tests regarding forward navigations from a page and backwards navigation to the page

* + - * + Functionality

All tests regarding functionality/interfunctionality of controls

* + - * + Content

All tests regarding data in the database

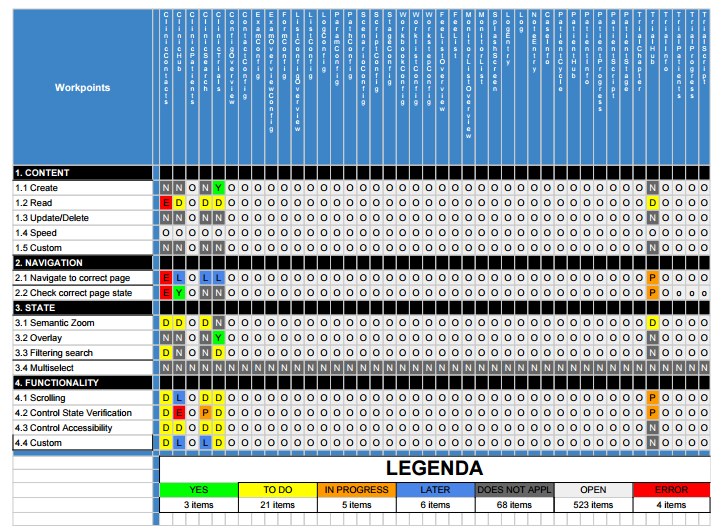
# Checklist: Testing

This is the checklist to review for every page. This means that for every page in the application, we need to test every paradigm documented in the checklist (control per control). If we find new paradigms while analyzing new pages, we will add these to the checklist and review all previously tested pages for the newly found paradigms.

This document contains the full checklist in which every paradigm that needs to be tested is fully explained together with how to test this paradigm. We also have an Excel document which contains the following sheets:

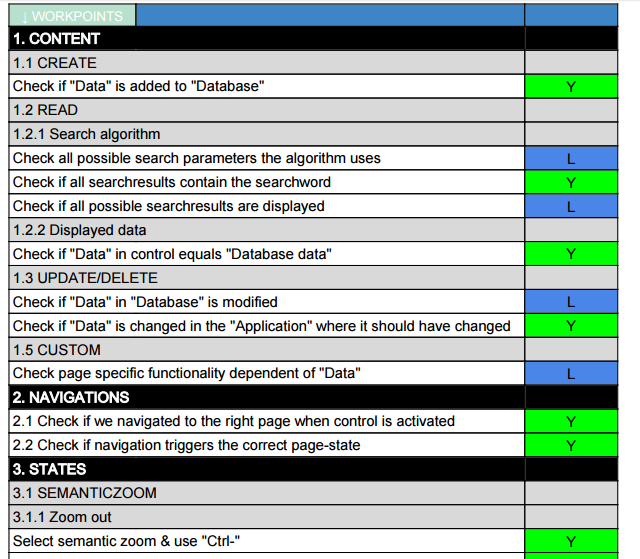
### General checklist

In this sheet, the general paradigms are displayed and we have indicate for every page if this paradigm is already tested or not. It looks like this:



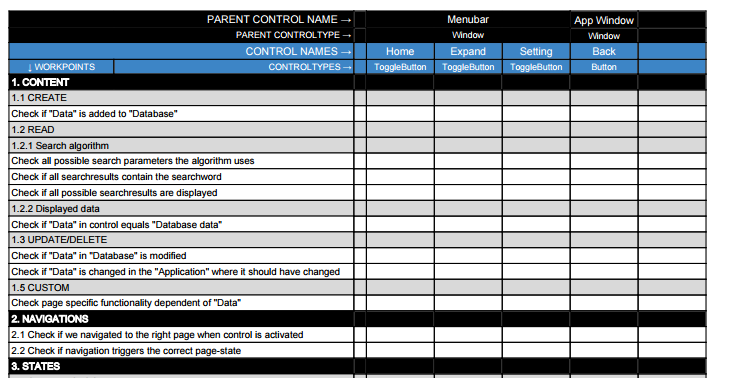
### Paradigm checklist

In the Paradigm checklist, we have a more detailed list for the checklist, containing all the subcategories explained in this document. Here, we indicate wether a certain paradigm is already analysed on the “How to test?”-aspect. It looks like this:

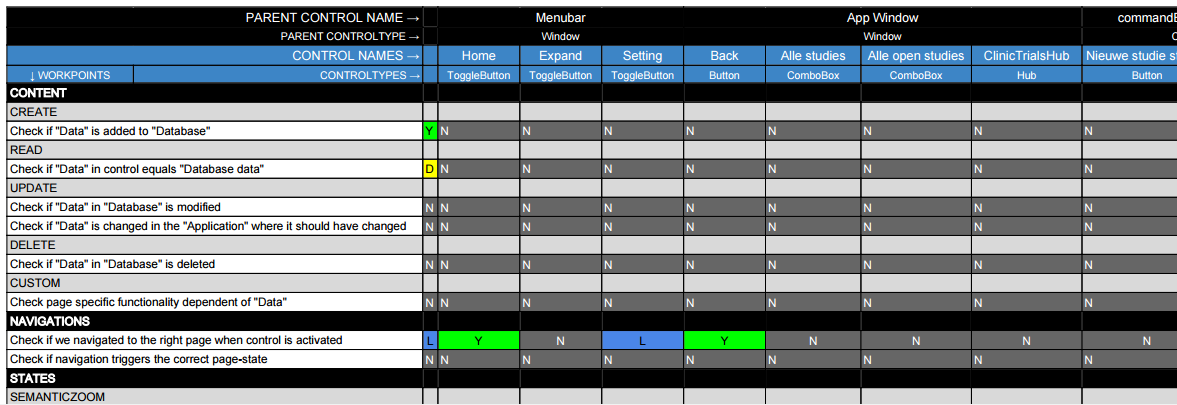


### Example page + Page checklists

In the example page, we have the same list as in the Paradigm checklist, but on top of it, but together with some of the basic controls which are present on every single page. On the Page checklists (one for each page), we use the example page and then search for every possible testable control. For every control we mapped, we indicate wether a certain paradigm needs to be tested on that control or not. The result looks like this:



The first small column is a summary of each row, to indicate wether this paradigm is tested for all controls it needed to be tested for.



## Content

This section contains every possible testscenario, which involves the data which is loaded in the application.

### 1.1. Create

#### what?

Check if adding data (for example adding patient) and clicking the execute button adds the data you added to the database, do this by asserting the elements in the user interface where this data should be added after clicking the execute button.

#### how?

1. Add all textfields (and if necessary, the “execute” button) involved in the “create”- action, to the UIMap.
2. Clear all textfields with the “ClearEdit(*textfield*)” function from the BaseClassCodedUI.
3. Add content to the textboxes by using the following line of Code:

TextBox.Text = *content*;

in which *content* is replaced by the data you want to add. To do this with external data: go to “How to test” → “Datadriven tests”.

1. If necessary, click the execute button.
2. Navigate to the page where your input should have influence and execute a “Read” test on the influenced controls, to make sure the data is added. How to execute a “Read” test is explained in the next paragraph.

### 1.2 Read

What?

##### Search algorithm

When using a searchbox, you want to check 3 things.

1. Check all the possible parameters on wich you can perform a search (for example, search by name, search by trial, search by trialnumber,...)
2. Check if all the searchresults contain the parameter you searched
3. Check if all possible searchresults available in the database, based on your search, are all displayed

##### Displayed data

Check if all controls on the page you are testing, containing data from the database, display the correct data for the parameters by which you got to that page. So for example, if you navigated to a trial with trial-id “4”. All controls in the TrialHub should contain the data of trial “4”, and not the data of for example trial “6” or completely faulty data.

##### How?

General

If you want to make sure a certain word/value is displayed in every single ListItem in the list, you can use the ReadListItems-function:

ReadListItems( *list*, *expectedValue*);

This function loops over every ListItem and checks all datafields inside of it, if the ExpectedValue isn’t displayed in one of the ListItems, the test will fail.

Search algorithm

1. to do
2. Use ReadListItems-function.
3. to do

Displayed data

1. Execute the correct actions to go to the page you are testing, using variables such as for example “Case ID”/”Trial ID”. Ideally, you should get this variable from a datafile in which all other data of this Case/Trial/… is available. For datadriven tests: go to “How to test” → “Datadriven tests”.
2. Add the involved controls on the page to the UIMap.
3. To check if the data is correct, just assert to *control*.Name or *control*.Text, depending on the type of control involved. The expected value should be in your datafile, so to access this, check out datadriven tests.

### 1.3 Update/Delete

#### what?

This paradigm is similar to create, but now you change already existing data instead of creating new data.

#### How?

The “how to” of Update/Delete is similar to the create.

1. Add all textfields (and if necessary, the “execute” button) involved in the “update”- action, to the UIMap.
2. Clear all textfields with the “ClearEdit(*textfield*)” function from the BaseClassCodedUI.
3. Add content to the textboxes by using the following line of Code:

*TextBox*.Text = *content*;

in which *content* is replaced by the data you want to add. To do this with external data: go to “How to test” → “Datadriven tests”.

1. If necessary, click the execute button.
2. Navigate to the page where your input should have influence and execute a “Read” test on the influenced controls, to make sure the data is updated. How to execute a “Read” test is explained in the next paragraph.

### 1.4 Custom

* (things we may find in the future)

page specific functionality regarding data in the database

## 2. Navigations

2.1 Correct navigation

#### What?

When navigating to a certain page, from the page you are testing, you have to make sure the application navigates to the correct page according to the navigation-control you used. To keep the testing project easily readable, we will concider each different tab of a page (if there are tabs available) as a different page.

#### How?

##### Variable Controls

A variable Control is a Control which is embedded in a list, in which all cases of a certain object are displayed. For example, you can have a list of trials, patients, logs,... In other words, “variable” in this context means that there is never a fixed amount of Controls in the list, and the Controls displayed, never have a fixed instance inside the list. These kind of Controls all navigate to the same page, but the data displayed on that page is dependent on the instance you navigated to.

To test the navigation of this kind of Control, use the following function:

NavigateVariableControl(*parameters)*;

For more info about this function and its parameters, refer to “How to test” → “BaseClassCodedUI”

##### Fixed Controls

A fixed Control is a Control which is embedded in a HubSection, in which every possible navigation-control has a fixed instance. Also, there is a fixed amount of navigation-control’s inside the hub. These kind of Controls normally navigate to a different page, or to a different state of the same page(for that, refer to the next paragraph).

To test the navigation of this kind of Control, use one of the following functions:

* NavigateFixedControlToPage(*parameters*);
* NavigateFixedControlToTab(*parameters*);

For more info about these function and there parameters, refer to “How to test” → “BaseClassCodedUI”.

Search navigation

### 2.2 Correct page state

#### what?

On some pages, you can use multible controls to navigate to the same page. But every control triggers a different page state on the page you navigated to. So you need a test which can assert if the navigation-control you used, triggers the correct page-state on the page navigated to.

#### How?

##### Filtering search

In this case, the Controls trigger a certain FilterBox-state on the page navigated to. This is how to test this:

1. Add the control you want to navigate with to the UIMap
2. Generate an ExpectedValue, refer to “How to test” to see how to do this.
3. Navigate
4. Get the combobox involved in the filtering, and get its SelectedItem
5. Assert the ExpectedValue to the SelectedItem’s value

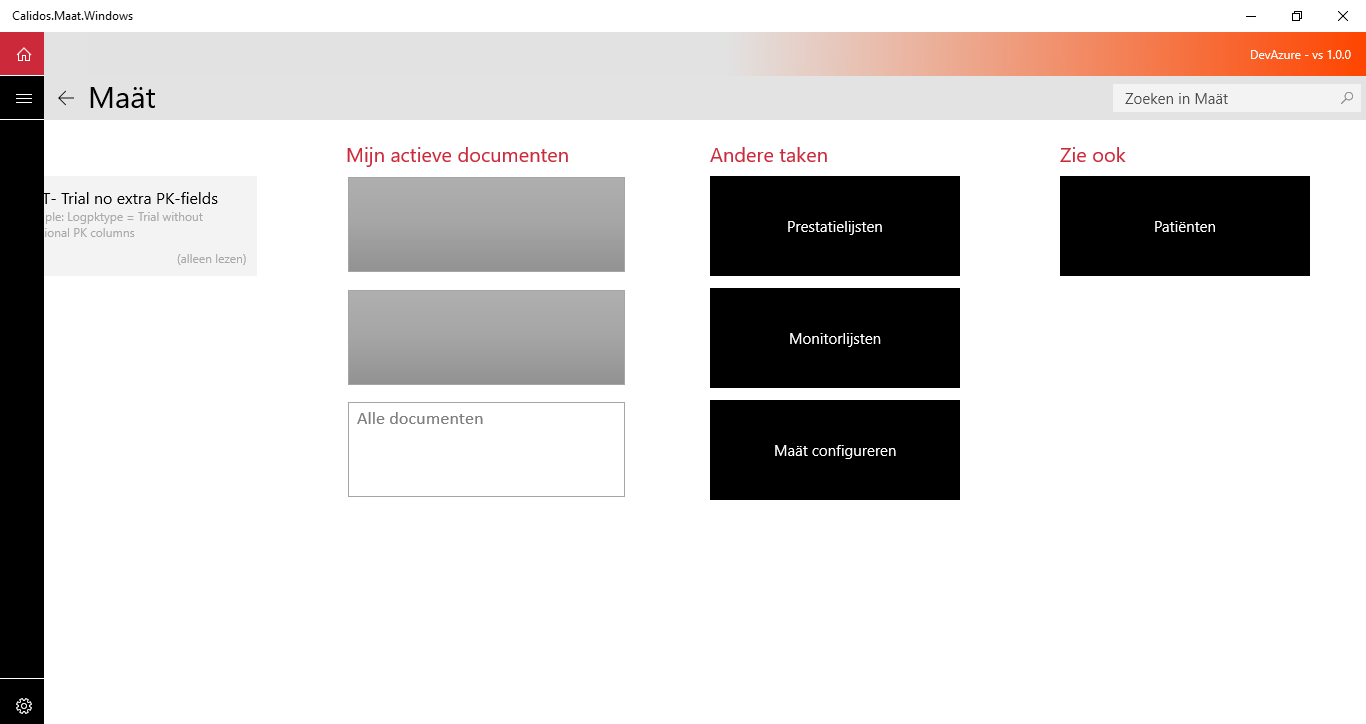
## 3. States

### 3.1 Semantic zoom

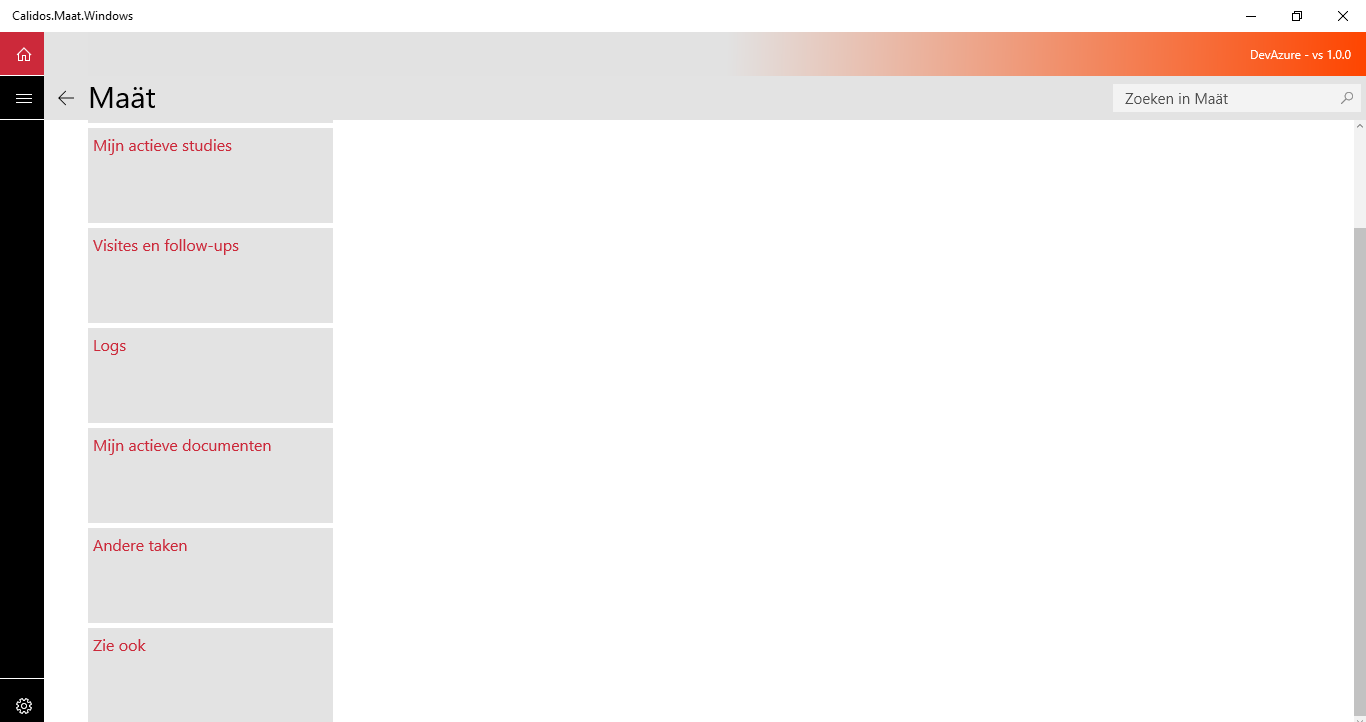
#### What?

The semantic zoom is a control within a page, which contains a hub. This hub contains different hubsections, each with it’s own title. The semantic zoom is able to zoom out these hubsections so that they are displayed as a list of ListItems, each containing the text of one of the hubsection titles. This is a functionality that needs to be tested.

* Zoomed in view:



* Zoomed out view:



#### How?

temporary for Zoom out & Zoom in: BaseClassCodedUI function:

AssertSemanticZoom(*Hubsection, HubsectionSemanticName, action*);

* Hubsection: Hubsection you want to test
* HubsectionSemanticName: Name of zoomed out listItem representing hubsection
* action: CtrlKey/Scroll

##### Zoom out

###### Ctrl-

Check if using the “ctrl” key in combination with the “-” key makes the semantic zoom zoom out. Note that the semantic zoom needs to be selected before this action is possible.

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

###### Ctrl & mousescroll

Check if using the “ctrl” key in combination with the “mousewheel” makes the semantic zoom zoom out. Note that the mouse needs to be hovered over the semantic zoom before this action is possible.

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

*Tab/arrow select*

Check if using the tab key (or arrow keys) selects the hubsection titles in the semantic zoom and using the -

* space
* enter

-key makes the semantic zoom zoom out.

**//NOT YET WORKING//**

##### *PgUp/PgDn select*

Check if using the PgUp/PgDn keys selects the hubsection titles in the semantic zoom and using the -

* space
* enter

-key makes the semantic zoom zoom out.

**//NOT YET WORKING//**

*Mouseclick*

Temporary for zoom out & zoom in: BaseClassCodedUI function:

AssertSemanticZoom(*Hubsection, HubsectionSemanticName, HubsectionTitle*);

* Hubsection: Hubsection you want to test
* HubsectionSemanticName: Name control of zoomed out listItem representing hubsection
* HubsectionTitle: Title control of hubsection in zoomed in view

Check if clicking on a hubsection-title makes the semantic zoom zoom out.

#### Zoom in

##### *Ctrl+*

For each page, check if the semantic zoom functionality of the page zooms in when using the “Ctrl” key in combination with the “+” key. Check if the selected hubsection is positioned correctly (on the left side of the screen). To select a different hubsection, use the tab key.

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

##### *Ctrl and mousescroll*

For each page, check if the semantic zoom functionality of the page zooms in when using the “Ctrl” key in combination with the “mousewheel”. Check if the selected hubsection is positioned correctly (on the left side of the screen). To select a different hubsection, use the tab key.

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

*Tab/arrow select*

Check if using the tab key and arrow keys selects the hubsection titles in the semantic zoom and using the -

* space
* enter

-key makes the semantic zoom zoom in.

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

Check if the selected hubsection is positioned correctly (on the left side of the screen).

*PgUp/PgDn select*

Check if using the PgUp/PgDn keys selects the hubsection titles in the semantic zoom and using the -

* space
* enter

-key makes the semantic zoom zoom in.

Check if the selected hubsection is positioned correctly (on the left side of the screen).

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

##### *Mouseclick on hubsection-titles*

For every page, check if the semantic zoom functionality of the page zooms in when clicking on the titles of different hubsections. Check if the clicked hubsection is positioned correctly (on the left side of the screen).

**//CODE STILL NEEDS TO BE WRITEN MORE EFFICIENTLY, AFTERWARDS, FINISH “How?”//**

#### Content

Check if correct data is displayed in hubsections when zoomed out

**//NOT YET ANALYSED//**

#### Functionality

To check if the titles displayed in the zoomed out list of the hubsections, are the same as the titles in the zoomed in state (and the same amount), use the following function of the BaseClassCodedUI:

CheckSemanticZoomTitles()

### 3.2 Overlay

#### 3.2.1 Content

##### CRUD (Create/read/update/delete)

Look to: Content

#### 3.2.2 Navigations

Look to: Navigations

3.2.3 Functionality

##### Speed

//NOT YET ANALYSED//

##### Control functionality

###### Open overlay button

* What?

Check if using the button intended to open the overlay, triggers the overlay appearance.

* How?

1. Add button to the UIMap
2. Add the full overlay to the UIMap as a control (this will be a Hub)
3. Click the button
4. Assert on the coördinates of the hub, if the hub is visible, these should have a value above “0”. Otherwise, all values should be “-1”.

note: The overlay needs to be opened and used by your testmethod, before it can see the coördinates being set to -1 after closing it. If you don’t open it initially and try to search it’s coördinates, the testmethod will fail to find the overlay in the first place.

###### Close overlay button

* what?

Check if using the button intended to close the overlay, makes the overlay disappear.

* How?

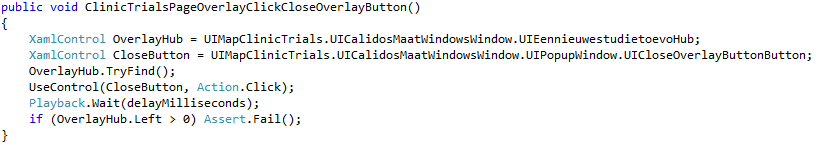
1. Add “x” button to the UIMap (for some overlays, you may need to give an automationId to the button).

ex.



This line is located in the ClinicTrials.NewTrial - page, in which NewTrial is the overlay of ClinicTrials.

1. Add the overlay hub to the UIMap
2. Use the XamlControl.TryFind() function to make sure the overlay is opened.
3. Click the button & add delay to make sure the overlay has time enough to close completely
4. Check if one of the coördinates of the OverlayHub is still larger than zero, normally when a control is not visible on a page, it’s coördinates will all be set to “-1”.



###### Close overlay with “esc”

* What?

Check if overlay closes when pressing the “esc”-key

* How?

Same method as above, but instead of clicking a button, use Keyboard.SendKeys(“{Escape}”)

###### Control functionality

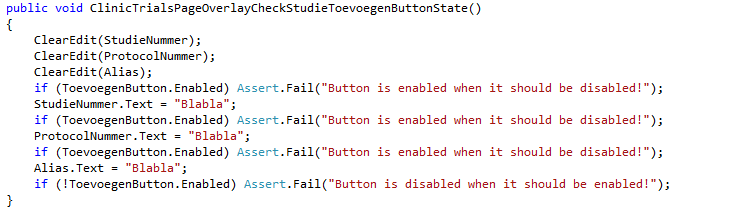
* What?

Check if “execute” button is only enabled when correct datafields are filled in

* How?

1. Make sure the datafields involved are all empty before you begin checking the button’s state.
2. Check button’s state after filling in datafields

ex.



### 3.3 Filtering search

* Check if Combobox ordering function orders the page correctly

**//NOT YET ANALYSED//**

## 4. Functionality

### 4.1 Speed

#### Reaction speed

##### What?

Check how fast a page is loaded when navigated to. Save this loading time and report.

##### How?

// try using streamwriter to txt file

**//NOT YET ANALYSED//**

#### Reactivity

##### What?

Check if you can already use loaded functionality before the page is fully loaded.

##### how?

**//NOT YET ANALYSED//**

### 4.2 Scrolling

#### 4.2.1 Check “Mousewheel” scroll functionality

##### What?

Check if the scroll function works with the “mousewheel”-action.

##### how?

Use the following function of the BaseClassCodedUI:

ScrollToControl(*control,scrollControl, scrolldirection*)

* control: The control you want to scroll to
* scrollControl: The control you want to start scrolling from
* scrolldirection: Up or Down

4.2.2 Check “Click scrollbar & drag” functionality

##### What?

Check if the scroll function works when clicking and holding the scrollbar and than dragging it across.

##### how?

PROBLEM: Can not access the scrollbar in any way other than manual

4.2.3 Check “Click scrollbar arrows” functionality

##### What?

Check if the scroll function works with when clicking the arrows of the scrollbar.

##### how?

PROBLEM: Can not access the scrollbar-arrows in any way other than manual

4.2.4 Check scrollbar state

##### What?

Check if hovering over the scrollbar makes the scrollbar highlight

##### how?

PROBLEM: Can not access the scrollbar in any way other than manual

4.2.5 Check scrollbar-arrows state

##### What?

Check if hovering over the scrollbar arrows makes the scrollbar arrows highlight

##### how?

PROBLEM: Can not access the scrollbar-arrows in any way other than manual

### 4.3 Control state verification

#### 4.3.1 Initial

##### what?

Check if the control’s initial state (appearance) is as it should be.

##### how?

* color
* highlighted
* enabled

Create the correct conditions for the control to be enabled/disabled and then assert to the “*Control*.Enabled”-property, which is a boolean.

* default text
* default value

(check xaml page for “anchor”?)

#### 4.3.2 Hovered

##### What?

Check if the control’s state changes correctly when hovering over the control.

##### how?

* Mouse/cursor change
* color
* highlighted
* enabled

Use the “Mouse.Hover”-function on the control you want to test and then assert to the “*Control*.Enabled”-property, which is a boolean.

* default text
* default value

#### 4.3.3 Clicked

##### What?

Check of the control’s state changes correctly when clicking on the control.

##### How?

* Mouse/cursor change
* color
* highlighted
* enabled

Use the “Mouse.Click”-function on the control you want to test and then assert to the “*Control*.Enabled”-property, which is a boolean.

* default text
* default value

#### 4.3.4 Click & hold

##### What?

Check if the control’s state changes correctly when clicking on it and holding your click.

//StartDragging?

##### How?

* Mouse/cursor change
* color
* highlighted
* enabled
* default text
* default value
* Control deepens?

#### 4.3.5 Filled in

##### What?

Check if the control’s state changes correctly when filling it in.

##### How?

* color
* highlighted
* enabled
* value

### 4.4 Control accessibility

#### what?

Check if the control is accessible through every action it should be accesible through. This means that the control will be selected when this action is executed.

#### How?

##### Click

1. Add the control to the UIMap
2. Click the control
3. Assert to the control’s “HasFocus” property (should be true)

##### Tab

1. Add the control to the UIMap
2. Tab to the control with the TabToControl function in the BaseClassCodedUI, in case the control you want to access is embedded in a list, use the ListItemEnter-function.
3. Assert to the control’s “HasFocus” property (should be true)

### 4.5 Custom

#### 4.5.1 Buttons

##### 4.5.1.1 General

1. Check if hovering over "Button" shows tooltip
2. Check if Button executes correct action when clicked
3. Check if Button executes correct action when hitting enter (if selected)
4. Check if Button executes correct action when hitting space (if selected)

4.5.1.2 Toggle Button

1. Check if Toggle Button toggle-state is changed when clicked
2. Check if Toggle Button responds to ‘space’ key

#### 4.5.2 ListItems

1. Check if ListItem executes correct action when clicked
2. Check if ListItem executes correct action when hitting enter (if selected)
3. Check if ListItem executes correct action when hitting space (if selected)

#### 4.5.3 Edits

##### 4.5.3.1 General

1. Check clipboard functions with right mouse click:Cut
2. Check clipboard functions with right mouse click: Copy
3. Check clipboard functions with right mouse click: Paste
4. Check clipboard functions with right mouse click: Undo
5. Check clipboard functions with right mouse click: Select all

#### 

#### 4.5.3.2 TextBoxes

1. Verify correct data
2. Check if clicking the “x” in the Textbox sets it back into default state
3. Check the by default selected textbox
4. Check typing when selected --> text in selected textbox

#### 4.5.3.3 Searchbox

1. Check typing when selected --> text in searchbox
2. Check typing randomly --> text in searchbox
3. Check "esc" --> searchbox empty
4. Check the searchfunction for pressing “Enter”
5. Check the searchfunction for clicking the magnifying glass

### 4.5.4 Combobox

1. Check if ComboBox responds to ‘arrow’ key: immediately change selected item
2. Check all selected item highlight functions
3. Check combobox scrolling functions
4. Select combobox Item: Check selected item is correct
5. Select combobox Item: check correct action following
6. Check if combobox opens with: Click
7. Check if combobox opens with: Enter (when selected)
8. Check if combobox opens with: Space (when selected)

### 

# How to test

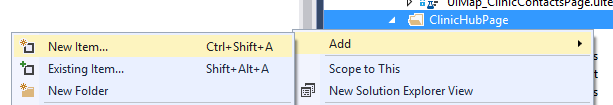
In this section, we will describe how to write tests for every item on the checklist. This is the actual implementation of the tests. In here, we will also describe some guidelines in the test code to keep the code readable and easily adaptable.

## Basics

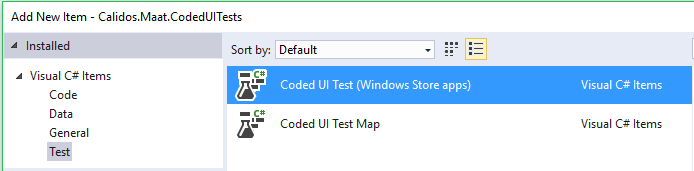
### Coded UI Test Builder and UI Map

To add a Coded UI Test project:

* Right click on the map you want to add the project
* Select ‘Add’
* Select ‘New Item’



* Select ‘Test’
* Select ‘Coded UI Test (Windows Store apps)’



To add a UI map:

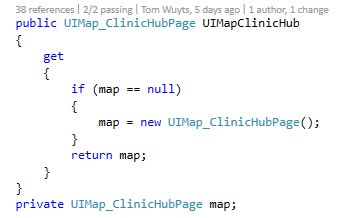
* Do the same
* Select ‘Coded UI Test Map’ instead of ‘Coded UI Test (Windows Store apps)’

When you added a Coded UI Test project, the first thing you need to do is add the right UI Map as a variable in your test project:

* At the top of your project, add a “using” statement for the UI Map you created. If I named my UI Map “UIMap\_ClinicHubPage”, this is what I would have to add:

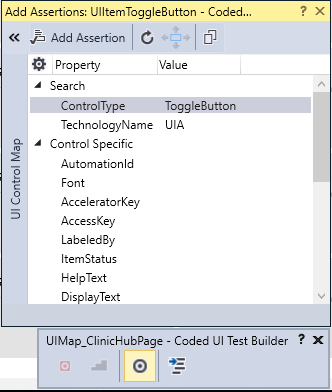


* At the bottom of your project, change the UI Map property to something like this:



After doing this you can start writing tests. To write a test you first have to add the controls you want to use to the UI Map you created. To do this, right click on your ‘.uitest’ file and select “Edit with Coded UI Test Builder”.

The Coded UI Test Builder will launch itself. Dont worry if it minimizes your Visual Studio, the Builder just wants to indicate that you can start up your application you want to be mapping. By dragging the circular marker (‘Add Assertions’) onto the control you want to map you can add it to the UI Map. When you release the marker it will open another window, seen below.



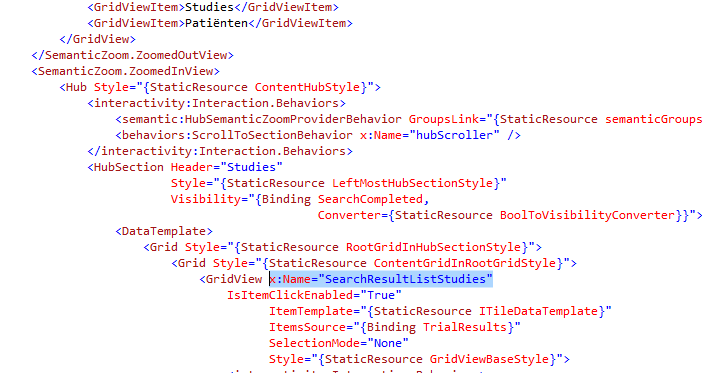
The control you selected will be higlighted within the application with a blue borded and on the right side of the new window you can see more detailed information about the currently selected control. If you click the arrow button in the top left hand corner you can see the hiërarchy in which the control is embedded as well.

Note:

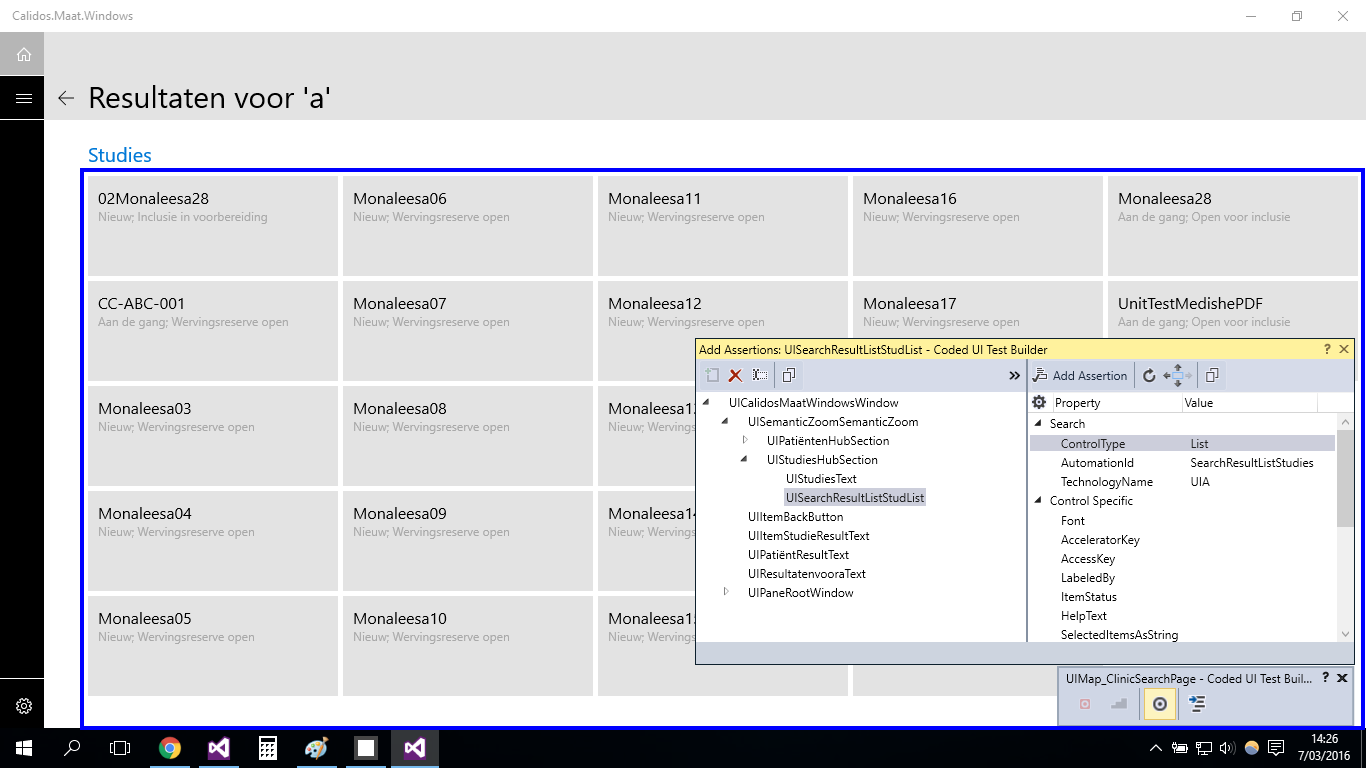
*Because the Clinical Trials application ‘Maät’ is created as a Windows 10 Metro App, the Coded UI technology, mainly the Coded UI Test Builder, is not yet fully adapted for optimal hiërarchy detection. To properly recognize the correct hiërarchy, every control has to have a unique AutomationId. However, almost no control in the application has this Id. For example, to add ListItems we needed to figure out special techniques and workarounds in the test methods, which we will discuss later.*

*To make sure control’s or lists are properly mapped and easy to find by the testprogram itself, we sometimes gave AutomationId’s to the control’s ourselves. To do this, open the XAML file of the page you want to test, and then search for an indication out of which you can derive this section is the section you want to give an AutomationId.*

*Example: I gave a unique Id to a list, so I can later easily access the childs of that list*



*I wanted to give a unique Id to the list of Studies-searchresults, so I searched the XAML file for a while, tried naming some different grid’s and gridviews, untill I named the right one. Now if I select that list with the Coded UI Test Builder, the name I gave to it will appear as AutomationId, as seen below.*



### Testmethods

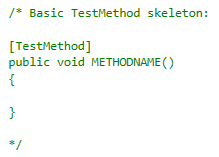
#### Test Classes

When you create a Coded UI Test class, the class must be marked as being for testing purposes. This is done by Visual Studio itself by adding the “[CodedUITest]” tag in front of the namespace declaration. More sepcific, for the application we are working on at the moment, is the “[CodedUITest(CodedUITestType.WindowsStore)]” tag that was auto-generated for us.

#### The ‘[TestMethod]’ tag

Every Coded UI Test needs to be preceded by a “[TestMethod]” tag, else the tests will not be picked up by Visual Studio’s test framework.

A basic testmethod skeleton is depicted below.



#### The ‘[TestInitialize]’ tag

This method will be executed **before every** ‘[TestMethod]’. Its important to note the ‘every’ in bold. Whenever a testmethod is ran, The testing framework loops over the whole test project to check for initialization methods. This method wil always be ran first. Which could make it a usefull method at certain points.

In our project, we will use this tag to navigate to the desired page in the application before every TestMethod using Process() and a URI string that every page in the application posesses

#### The ‘[TestCleanup]’ tag

This will be executed **after** **every** ‘[TestMethod]’. The same as with the TestInitialize, only after each and every testmethod. Usually this method contains a shutdown or close command to close the application after every test.

In our project this tag has not (yet) been used, bacause we dont want to reload all the data every few seconds for the UI tests.

#### The ‘[DataSource()]’-tag

When you want to create a datadriven test (which is a test using external data for execution/assertion/…), this tag can be added on top of the testmethod, together with the [TestMethod]-tag.

ex. [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", @"C:\dev\CTO\Src\Dev\D.Peter.0\Client\Calidos.Maat\Calidos.Maat.CodedUITests\Screens\Clinic\ClinicTrailsPage\NieuweStudieToevoegenData.csv", "NieuweStudieToevoegenData#csv", DataAccessMethod.Sequential), TestMethod]

Datasource uses 4 variables to connect to a datasource:

* type of dataconnection (ex. CSV, XML,...)
* path to datafile
* table name
* access method (sequential/random)

If you add a datatable to a testmethod, the method will be executed for every row in the table.

The first row will be ignored and can be used to asign variable names to your data for easy accesability.

#### Testcontext

“TestContext”, which is a property declared at the bottom of each testproject, contains information about each test.

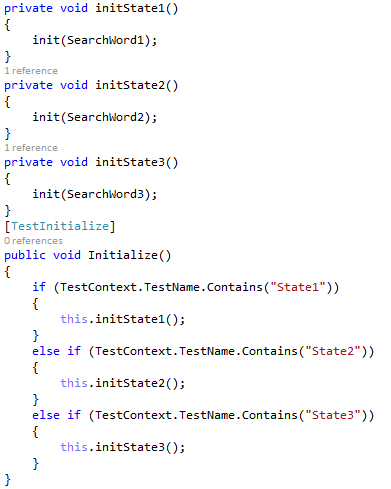
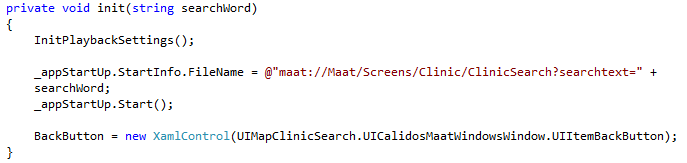
##### Data

when the test contains a datasource, data can be read through TestContext. Do do this, you can use the property “TestContext.DataRow[*row*]” in which row can be an index number or a title of the row you gave in your datatable (in string format)

##### TestName

In some testproject, you will want to be able to use different TestInitialize methods. However the TestInitialize tag can only be used once. To solve this problem, you can write private functions with the different initialization methods, and in the TestInitialize method you can write an if-statement using the TestContext.TestName property. This is a string representation of the name you gave to your testmethod.

full example:



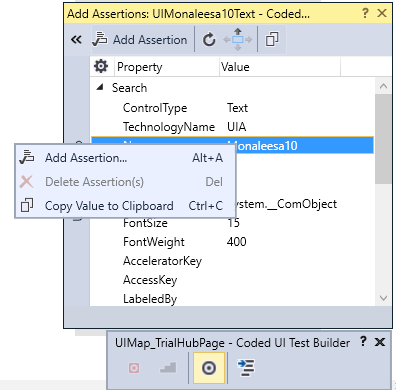
### Generating controls

There are a few phases in order to generate a control in your testmethod:

* Identify the type of control you want to generate (XamlButton, XamlEdit,...).
  + XamlControl is a general name you can use for every type of Xaml control. The downside of doing so, is that you might not be able to use some control-specific properties.
* After you identified the control, give it a unique, yet obvious name.
* Finally, assign a path to your control referring to the control you added to the UIMap.
* After generating a control, you can use its properties to create different kinds of tests. Some example of control properties often used in tests:
  + Name The name of the control
  + Font Textfont (handy when asserting if a tab is selected or not (Bold text)
  + Exists Bool that indicates if the control exist on the current page or not
  + ...

Sometimes you want to use control-specific proporties (like for example with togglebuttons,...). Then you have to give a specific Xaml-type to the control (ex. XamlText, XamlEdit,...)

You will end up getting something like below image.

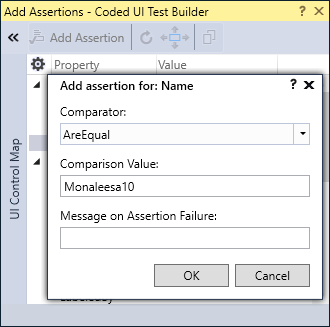


### Writing assertions

#### (Semi) Automatic asserts

The Coded UI Test Builder enables you to write semi-automatic asserts.

After selecting a control with the marker, you can make an assertion for a specific property, by selecting it and clicking “Add Assertion” in the top left corner of the window, or by simply right clicking on the property you wish to assert and choosing “Add Assertion” as illustrated in the snapshot on the right.



A new window will pop up. Here you can choose the comparator, comparison value and message you will receive when the assertion fails.

In the example we go for the ‘AreEqual’ comparator, but there are some more things you could choose from, like ‘Contains’ or ‘StartsWith’.

The comparison value is the value you want the actual value to be, in order to let the assertion pass. In other words, the test framework will get the actual value and check it against the comparison value

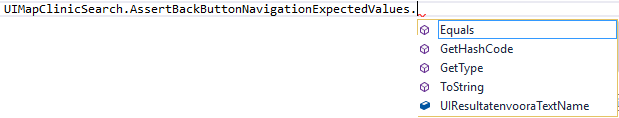
Additionally you can add a Message on assertion failure to get a better understanding what went wrong if an assertion failed. When done, click the ‘OK’ button.

Click the “Generate” button at the Coded UI Test Builder and your assertion a name. You can also add a description to make sure everyone knows what the assertion is all about.

Your assertion is now ready. In order to use it, you have to call your UI Map first. Remember that everything that is generated with the Coded UI Test Builder is added to this map. After calling the UI Map you will see that your assertion is in the list of suggested code



The UI Map in the example is the UI Map of the testproject, and the highlighted area is the name you gave to your assertion method. The assertion contains the expected value you assigned on creation. However, on certain moments we will want to have a different expected value. The value can still be changed within your testmethods or within the partial UI Map class (The part that doesnt contains auto-generated code). Changing the expected value can be done like below:



#### Manual assertions

The advantage of manual asserts is that you have much more control over the control you want to assert to as well as the expected values to make the test as autonomous as possible.

A manual assertion always starts with the ‘Assert’ keyword followed by the comparator. Note that there are different comparators availbable when making a manual assertion (‘Contains’ for example is not available when using manual assertions). The comparator requires parameters to create the full assertion.

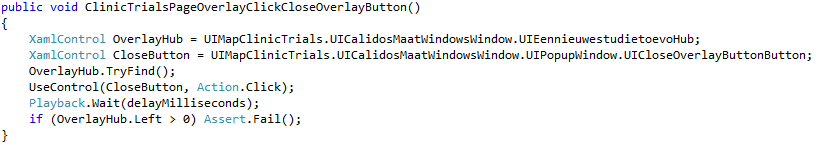


In the example, where we make an ‘AreEqual’ assertion, the first parameter is the expected value. The second parameter is the actual value. Where these 2 value’s come from, is completely under your control. This way of writing assertions is way more intensive then the auto-assertions, but makes room for some more flexibility. Manual assertions are written inside your testmethods.

#### Assert.Fail()

This is a function, when executed, which automatically makes the test fail. You can give a string to the function in which you describe the error occuring if the function is executed. Assert.Fail() is an easy way of making a test fail when a certain condition is true (for example a control is visible when it should not be visible). In other words, for some tests we will write “if” statements in which faulty conditions are specified. Inside these “if” statements we will write an Assert.Fail().

ex.

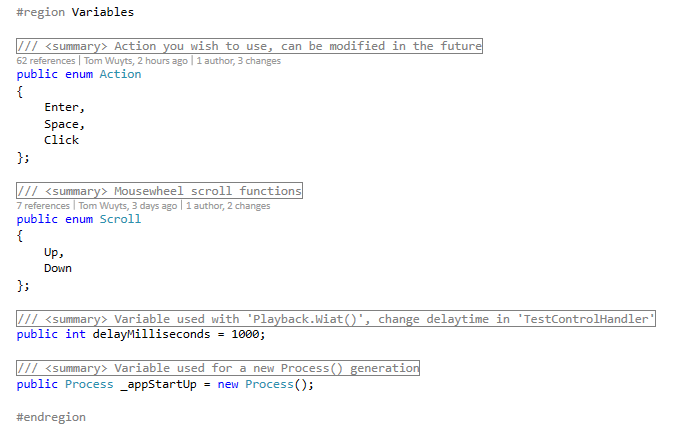


## BaseClassCodedUI

This is a baseclass we wrote ourselves, in which we write functions for actions we have to use very often. Like this, we can very easily perform the action multiple times in just one line of code. We will also add commonly used variables and global testscenario’s (that don’t require page-specific control’s but overal control’s, available on every page) to this class.

Every testproject that is created, has to inherit from this baseclass.

### Variables



* enum Action
  + Action to be used
* enum Scroll
  + Mousewheel scroll functions
  + Created purely for readability reasons
    - More readable then bool although only used for 2 states, like bool itself
* public int delayMilliseconds
  + The Playback.Wait() delaytime between actions
  + Adjustable variable in 1 place
* Process \_appStartUp
  + Variable that is used to start the application on the right page using a URI

### Functions



* InitPlaybackSettings
  + Initialize the settings used during testing
  + Changed some default value’s to more appropriate value’s for our tests
* UseControl
  + Control that will be used as a clickable object through click or key interaction
    - control XamlControl that will be used
    - action enum Action that is needed
* ComboBoxSelectItem
  + Changing the ComboBox control’s SelectedIndex
    - box XamlComboBox that will be used
    - number Index number of the wished SelectedIndex
* ListItemEnter
  + Use the wanted instance inside a list by using tab & enter
    - list List inside which your control exists
    - instance Instance of the control inside the list
    - action action used (may change in future)
* GetListChild
  + Will return a XamlControl, which is the control you want to click
    - list List you specified
    - instance The instance number of the control you want to click

inside the list

* TabToControl
  + Tabbing towards a certain XamlControl that may or may not yet be visible
    - control XamlControl that will be used
* ScrollToControl
  + Scrolling towards a certain XamlControl that may or may not yet be visible
    - control XamlControl that will be used
    - scrollControl XamlControl used to hover over, so we can scroll
    - scroll enum Scroll to indicate the direction to scroll towards
* WriteAsUser
  + Converts string to char’s and simulates as if a user enters the key’s as input
    - msg string that is converted to char’s as user input
* UseSearchbox
  + Method for using the searchbox
    - searchbox Xamlcontrol used to refference where to click
      * Button of searchbox is untargetable by UIMap
    - msg string that is converted to char’s as user input
      * WriteAsUser is used within this method

scroll enum Scroll to indicate the direction to scroll towards

## Navigation

### Click

#### Tile list navigation

Find a control by adding a list to the UIMap with the Coded UI Test Builder

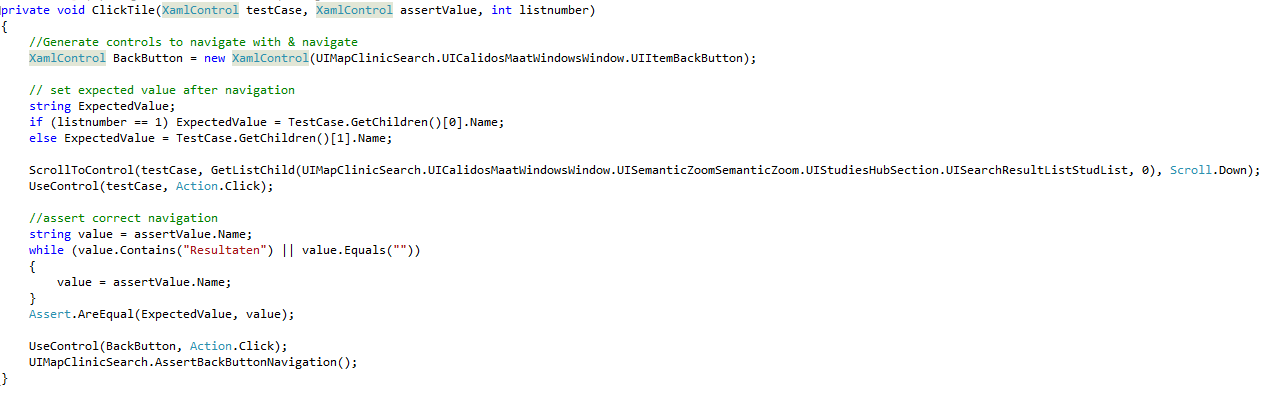
Because most control’s don’t have an AutomationId in the clincical trials application, the Coded UI Test Builder will probably wrongly interpret the hierarchie of controls. Lists of tiles give back wrong hierarchies almost every time. We came up with a solution for this. In the XAML page of the app, there are regions specified for the list in which you want to click a control. You can add a name to it by adding < *x:name=”listname” >*. Be sure to check if you added the correct name by using the Coded UI Test Builder. If correctly implemented, you will be able to always find the correct list you want to use. Because you want to click an element inside this list, you can’t just hardcode the control to your UIMap, even if AutomationId’s would be present. This is because you have to be able to change the instance of the control you want to click later, so te generate a control out of the list, do the following:

* Generate Control for the tile out of one of the list’s elements
  + After adding the list to the UImap, you can use the function GetListChild (out of BaseClassCodedUI) to access the list and get it’s items.

Add the control you want to assert if the click triggered the correct navigation to the UImap (Usually this will be the title of the navigated page)

* + Generate this control in your testmethod
* Generate an expected value by getting the children of your control and selecting the instance in which the desired value is embedded (mostly this will be a name). Later you can assert to the name of this expected value
* Click control
  + In the BaseClassCodedUI, there is a function called ClickControl
* Assert correct navigation
  + Write a while loop in which you make sure the correct title is already loaded before you execute your assert (manual delay function)
  + Assert expected value to the name of your previously added value to assert to

***Full example:***



*This is the basic function used by all click-tests in the ClinicSearchPage.*

* *The back button is generated the same for every test.*
* *testCase is the instance we want to have clicked (found by applying the GetListChild function on the XamlList in which the control is embedded).*
* *assertValue is the title we want to assert to*
* *listnumber is a number to indicate some page specific properties, regarding the expected assert value*

***// to finish***

#### Back button navigation

* Add back button to UImap
* Add back button to every testmethod in your navigation-testproject
* Add title of your page to UI-map and make assert for name of it
* Write click-action for back button at the very end of every testmethod
* Write assert below click action
* example: as above

### Enter

#### Tile list navigation

* Same as click, but use ListItemEnter function from UIBaseClass instead of ControlClick → give list to function (instead of control), the instance of the tile you want to click, and the action you want to execute (enter,...) :

ListItemEnter(list, Instance, action);

#### Back button navigation

* Same principle but instead of ControlClick(), use ControlEnter() function of baseclass

# Adding Coded UI Tests to Build process

Some information considering this part can be found on below URL. This MSDN link has not yet been updated to Visual Studio 2015, but will generally be the same process to set up.

URL: <https://msdn.microsoft.com/en-us/library/ms182465%28v=vs.110%29.aspx#Anchor_3>