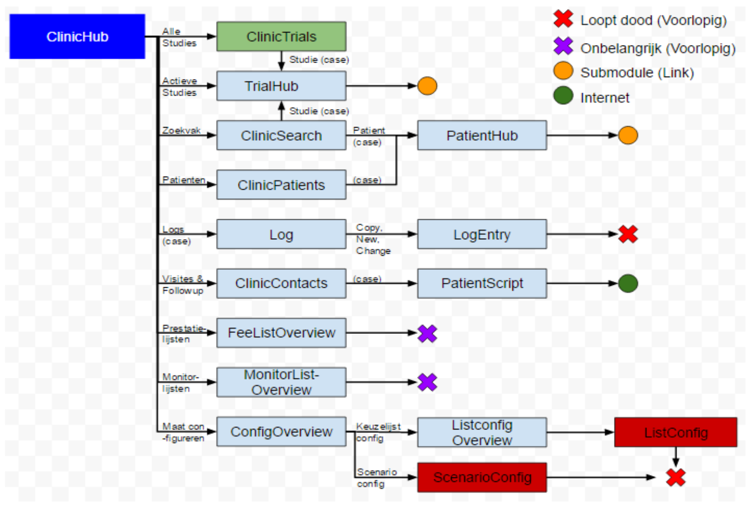
Testing Guideline

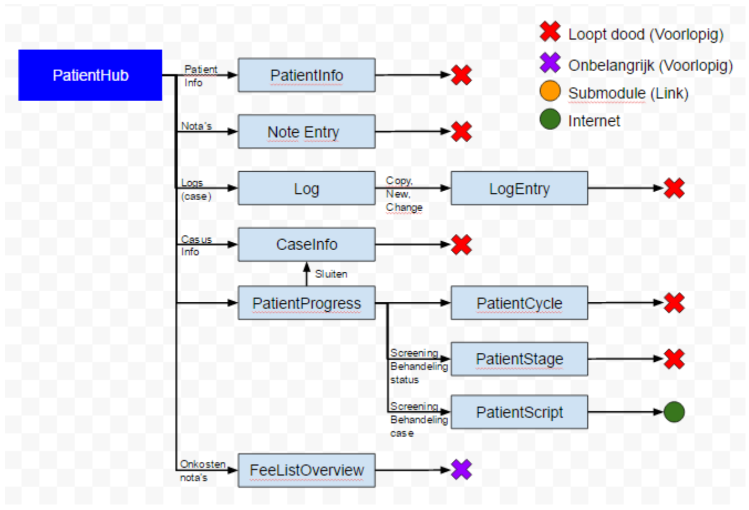
Hierarchical navigation design

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.

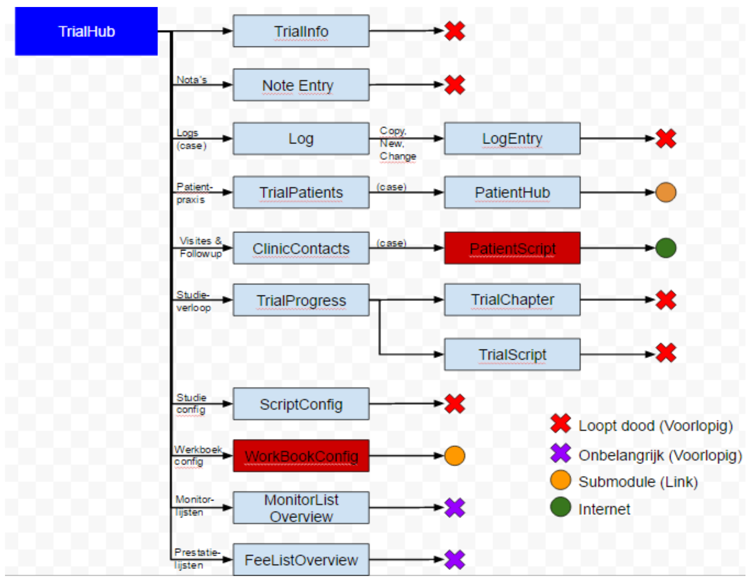
ClinicHub



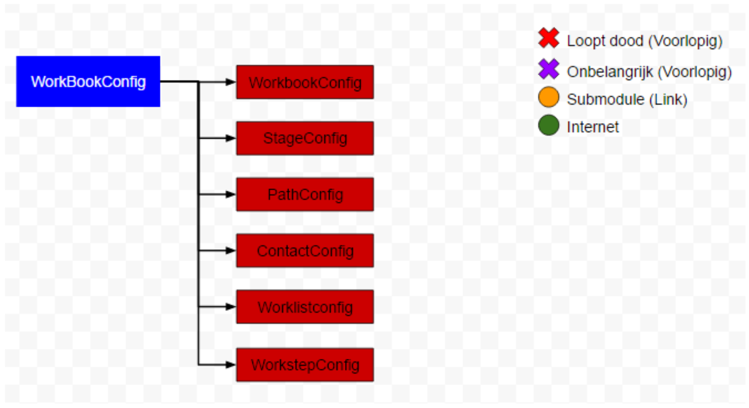
PatientHub



TrialHub



WorkBookConfig



Method Naming

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

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
  - Ex: navigate, zoom, toggle,...
- Context: Studie
  - Indicator of the context in which the control/set of controls you want to 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.

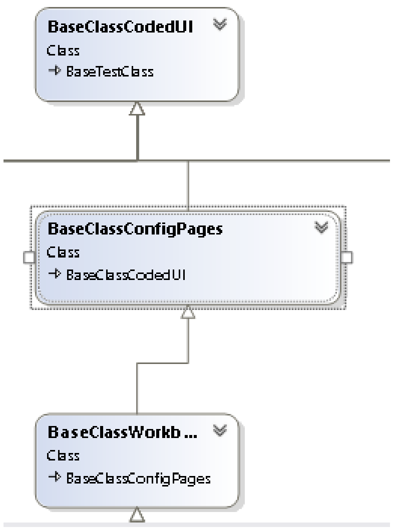
File Structure

Maät follows a specific map structure for all it’s elements. To keep the file structure readable, we follow the same structure inside our test project. For more information on the hiërarchy, refer to the next page.

Calidos.Maat.CodedUITests
- Screens
  - General UIMap
  - BaseClassCodedUI partial classes
  - .csv global scenarios
  - Category (clinic,…)
    - Optional BaseClass per category
    - Optional UIMap per category
    - .csv categorical scenarios
    - Page (contactspage,….)
      - UImap per page
      - BaseClass per page
      - .csv local scenarios
      - .cs files per paradigm (content, navigations,...)

Testproject hierarchy

In the progress of our project, we often found that some functions could write some functions in parent-classes to make them reusable. The end result of this is the following hierarchy:



BaseTestClass is the overall class for the entire test project. This class has been written to enable the possibility of implementing test functions on different test project.

BaseclassCodedUI inherits from BaseTestClass, and contains all the functions for testing "Maät".

BaseClass*Category* (in this case the ConfigPages) is a baseclass containing global functions and variables per screen-category. This is an optional class for those pages that are visually very similar, it inherits from the BaseClassCodedUI.

BaseClass*Page* (in this case WorkBookConfig) inherits from BaseClass*Category* and contains all private functions and variables used to test one page. All test projects per paradigm inherit from this class.

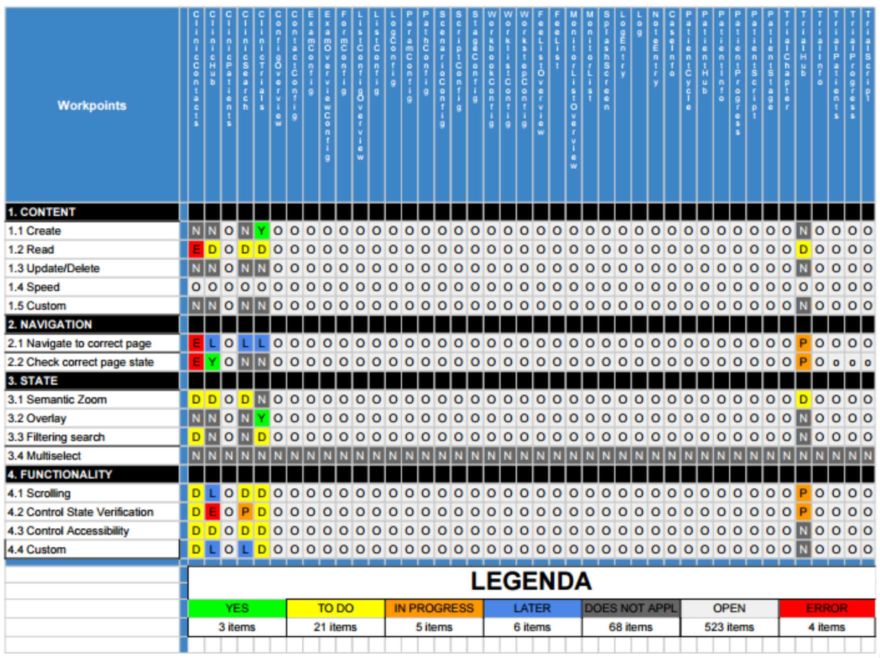
Checklist: Testing

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:



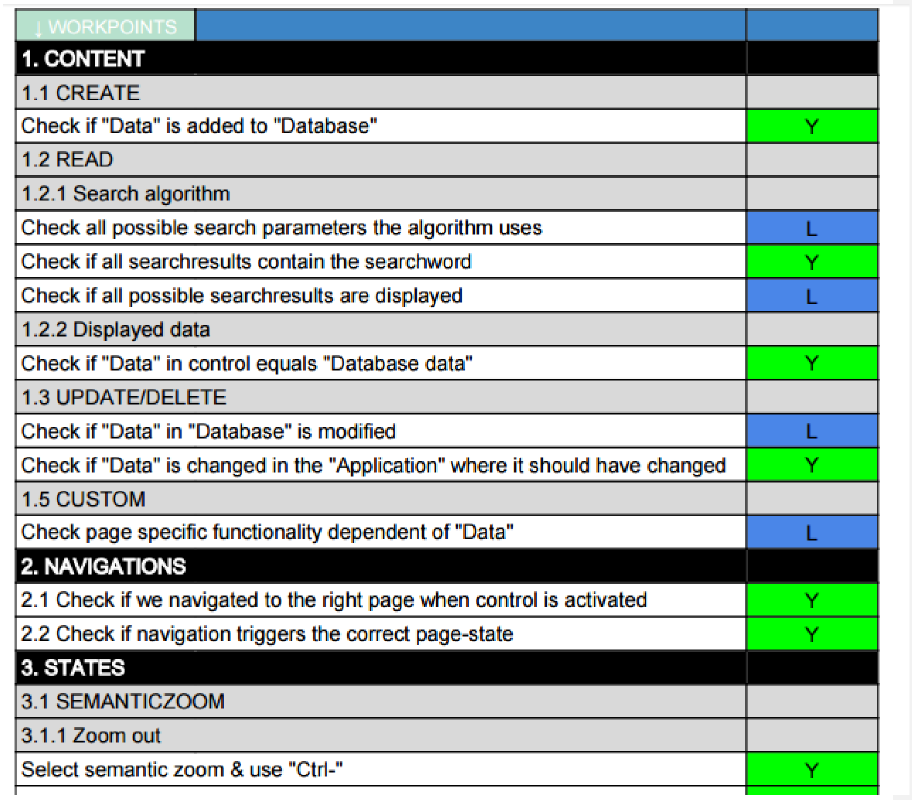
Workpoints	ClinicContacts	ClinicHub	ClinicPatients	ClinicSearch	ClinicTrials	ConfigOverview	ContactConfig	ExamConfig	ExamOverviewConfig	FormConfig	ListConfigOverview	ListConfig	LogConfig	ParamConfig	PathConfig	ScenarioConfig	ScriptConfig	StageConfig	WorkbookConfig	WorklistConfig	WorkstepConfig	FeeListOverview	FeeList	MonitorListOverview	MonitorList	SplashScreen	LogEntry	Log	NoteEntry	CaseInfo	PatientCycle	PatientHub	PatientInfo	PatientProgress	PatientScript	PatientStage	TrialChapter	TrialHub	TrialInfo	TrialPatients	TrialProgress	TrialScript
1. CONTENT																																										
1.1 Create	N	N	O	N	Y	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
1.2 Read	E	D	O	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	D	O	O	O	O
1.3 Update/Delete	N	N	O	N	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
1.4 Speed	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
1.5 Custom	N	N	O	N	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
2. NAVIGATION																																										
2.1 Navigate to correct page	E	L	O	L	L	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O
2.2 Check correct page state	E	Y	O	N	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	o	o	o
3. STATE																																										
3.1 Semantic Zoom	D	D	O	D	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	D	O	O	O	O
3.2 Overlay	N	N	O	N	Y	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
3.3 Filtering search	D	N	O	N	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
3.4 Multiselect	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
4. FUNCTIONALITY																																										
4.1 Scrolling	D	L	O	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O
4.2 Control State Verification	D	E	O	P	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O
4.3 Control Accessibility	D	D	O	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O
4.4 Custom	D	L	O	L	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O

LEGENDA

YES	TO DO	IN PROGRESS	LATER	DOES NOT APPL	OPEN	ERROR
3 items	21 items	5 items	6 items	68 items	523 items	4 items

Paradigm checklist

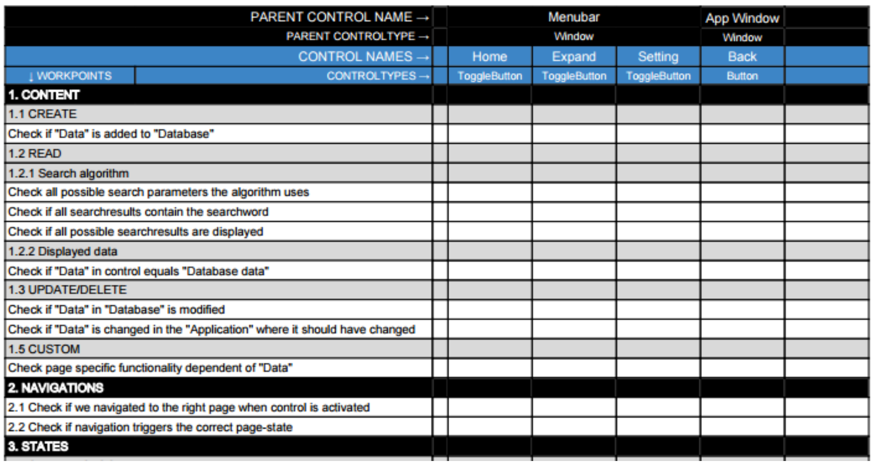
In the Paradigm checklist, we have a more detailed list for the checklist, containing all the subcategories explained in this document. Here, we indicate whether a certain paradigm is already analysed on the “How to test?”-aspect. It looks like this:



↓ WORKPOINTS	
1. CONTENT	
1.1 CREATE	
Check if "Data" is added to "Database"	Y
1.2 READ	
1.2.1 Search algorithm	
Check all possible search parameters the algorithm uses	L
Check if all searchresults contain the searchword	Y
Check if all possible searchresults are displayed	L
1.2.2 Displayed data	
Check if "Data" in control equals "Database data"	Y
1.3 UPDATE/DELETE	
Check if "Data" in "Database" is modified	L
Check if "Data" is changed in the "Application" where it should have changed	Y
1.5 CUSTOM	
Check page specific functionality dependent of "Data"	L
2. NAVIGATIONS	
2.1 Check if we navigated to the right page when control is activated	Y
2.2 Check if navigation triggers the correct page-state	Y
3. STATES	
3.1 SEMANTICZOOM	
3.1.1 Zoom out	
Select semantic zoom & use "Ctrl-"	Y

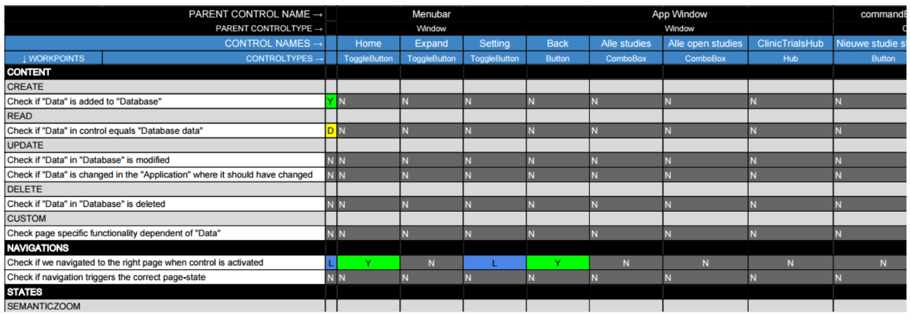
Example page + Page checklists

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 whether a certain paradigm needs to be tested on that control or not. The result looks like this:



PARENT CONTROL NAME →		Menubar			App Window	
PARENT CONTROLTYPE →		Window			Window	
CONTROL NAMES →		Home	Expand	Setting	Back	
↓ WORKPOINTS CONTROLTYPES →		ToggleButton	ToggleButton	ToggleButton	Button	
1. CONTENT						
1.1 CREATE						
Check if "Data" is added to "Database"						
1.2 READ						
1.2.1 Search algorithm						
Check all possible search parameters the algorithm uses						
Check if all searchresults contain the searchword						
Check if all possible searchresults are displayed						
1.2.2 Displayed data						
Check if "Data" in control equals "Database data"						
1.3 UPDATE/DELETE						
Check if "Data" in "Database" is modified						
Check if "Data" is changed in the "Application" where it should have changed						
1.5 CUSTOM						
Check page specific functionality dependent of "Data"						
2. NAVIGATIONS						
2.1 Check if we navigated to the right page when control is activated						
2.2 Check if navigation triggers the correct page-state						
3. STATES						

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



PARENT CONTROL NAME →		Menubar			App Window				commandB
PARENT CONTROLTYPE →		Window			Window				C
CONTROL NAMES →		Home	Expand	Setting	Back	Alle studies	Alle open studies	ClinicTrialsHub	Nieuwe studie s
↓ WORKPOINTS CONTROLTYPES →		ToggleButton	ToggleButton	ToggleButton	Button	ComboBox	ComboBox	Hub	Button
CONTENT									
CREATE									
Check if "Data" is added to "Database"	Y	N	N	N	N	N	N	N	N
READ									
Check if "Data" in control equals "Database data"	D	N	N	N	N	N	N	N	N
UPDATE									
Check if "Data" in "Database" is modified	N	N	N	N	N	N	N	N	N
Check if "Data" is changed in the "Application" where it should have changed	N	N	N	N	N	N	N	N	N
DELETE									
Check if "Data" in "Database" is deleted	N	N	N	N	N	N	N	N	N
CUSTOM									
Check page specific functionality dependent of "Data"	N	N	N	N	N	N	N	N	N
NAVIGATIONS									
Check if we navigated to the right page when control is activated	L	Y	N	L	Y	N	N	N	N
Check if navigation triggers the correct page-state	N	N	N	N	N	N	N	N	N
STATES									
SEMANTICZOOM									

1. Content

This section contains every possible test scenario, which involves the data which is loaded in the application. Note that some content tests are still performed hardcoded (we assert to a hardcoded value).

1.1 CRUD

Crud tests involve every action we can perform on the data in the user interface. Crud is an abbreviation for Create, Read, Update and Delete.

1.1.1 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. If the datafields aren't accessible through normal UI-mapping, add the parent control you can access to the UIMap and use the technique of getting children to reach the control's you want to use.
2. In case of textfields, 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" → "Test Methods" → "The datasource tag".

4. If necessary, click the execute button.
5. 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.

1.1.2 Read

What?

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?

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 data driven tests: go to "How to test" → "Test Methods" → "The datasource tag".
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 data driven tests. If you do not use a data driven test, you can still work with hardcoded expected values.

1.1.3 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" → "Test Methods" → "The datasource tag".

4. If necessary, click the execute button.

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 previous paragraph.

1.2 Custom-CRUD

Under this sub-section, we will place all special Crud-functions.

1.2.1 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.

1.2.2 Search algorithm

What?

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

A. Check all the possible parameters on which you can perform a search (for example, search by name, search by trial, search by trial number,...)
B. Check if all the search results contain the parameter you searched
C. Check if all possible search results available in the database, based on your search, are all displayed

How?

A. to do
B. Use ReadListItems-function.

to do

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 consider 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 their parameters, refer to "How to test" → "BaseClassCodedUI".

Search navigation

- add the SearchBox to the UIMap
- add a searchword to the SearchBox with "TextBox.Text = …"
- Hit enter
- Assert correct navigation by checking the title of the page navigated to

Hyperlink navigation

Use the following function of the BaseClassCodedUI:

"NavigateHyperLink(*hyperlink,titlecontrol*);"

2.2 Correct page state

what?

On some pages, you can use multiple 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

2.3 Load speed

What?

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

How?

1. Add the progressbar to the UIMap, to this by using the “Ctrl-i”-key combination while the page is loading, hovering over the progressbar. You will sometimes have to try this a few times because it disappears quickly.
2. In your code, navigate to the page and start a StopWatch (more info about the stopwatch in “how to test”)
3. Write a while function that keeps running as long as the ProgressBar’s coördinates are not below 0.
4. After the while function, save the StopWatch.Elapsed property in a variable

Report the variable (still to analyse)

Met opmerkingen [1]: still to finish

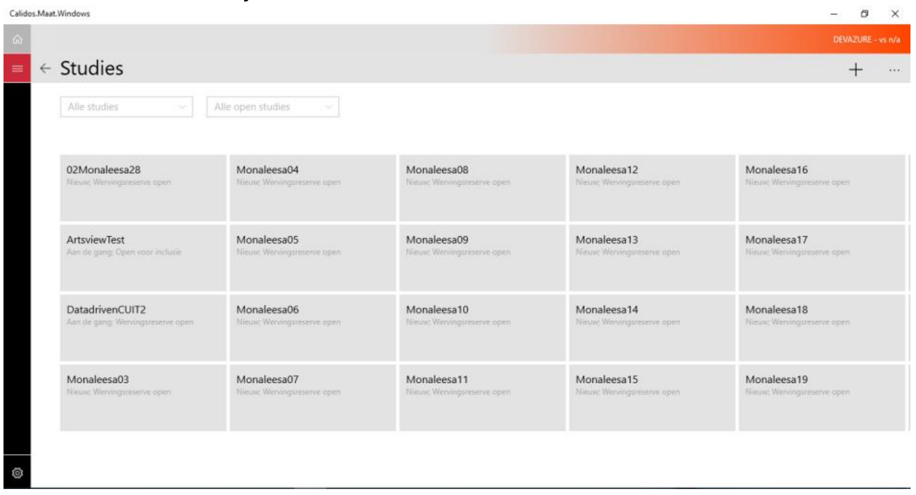
3. States

“States” are all the different ways a page can be ordered and displayed. There are 4 different possible states.

- Overlay

Overlay is a state in which a partial popup window is displayed on top of the normal window of a page. This overlay has data-fields in it, that are intended to add or modify data in the application. An overlay looks like this:

- No overlay:


Calidos.Maat.Windows

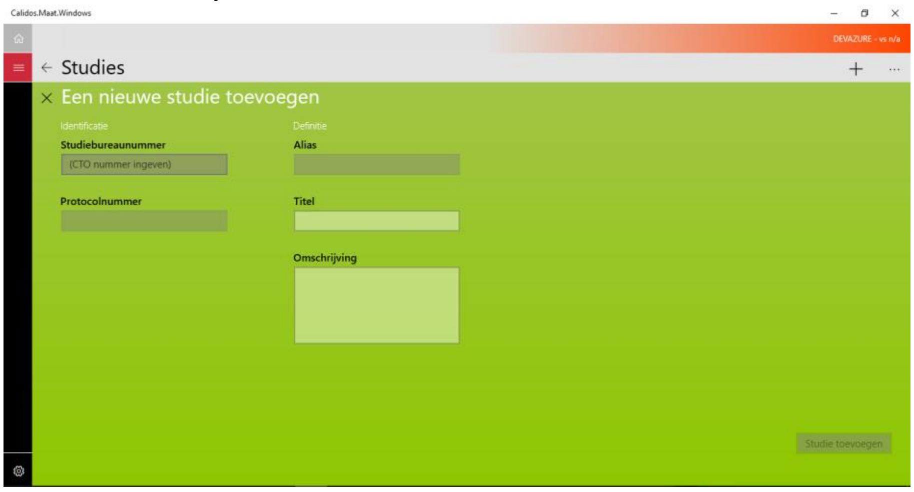
DEVAZURE - vs n/a

Studies

Alle studies | Alle open studies

02Monaleesa28 Nieuw; Wervingsreserve open	Monaleesa04 Nieuw; Wervingsreserve open	Monaleesa08 Nieuw; Wervingsreserve open	Monaleesa12 Nieuw; Wervingsreserve open	Monaleesa16 Nieuw; Wervingsreserve open
ArtsviewTest Aan de gang; Open voor inclusie	Monaleesa05 Nieuw; Wervingsreserve open	Monaleesa09 Nieuw; Wervingsreserve open	Monaleesa13 Nieuw; Wervingsreserve open	Monaleesa17 Nieuw; Wervingsreserve open
DatadrivenCUIT2 Aan de gang; Wervingsreserve open	Monaleesa06 Nieuw; Wervingsreserve open	Monaleesa10 Nieuw; Wervingsreserve open	Monaleesa14 Nieuw; Wervingsreserve open	Monaleesa18 Nieuw; Wervingsreserve open
Monaleesa03 Nieuw; Wervingsreserve open	Monaleesa07 Nieuw; Wervingsreserve open	Monaleesa11 Nieuw; Wervingsreserve open	Monaleesa15 Nieuw; Wervingsreserve open	Monaleesa19 Nieuw; Wervingsreserve open

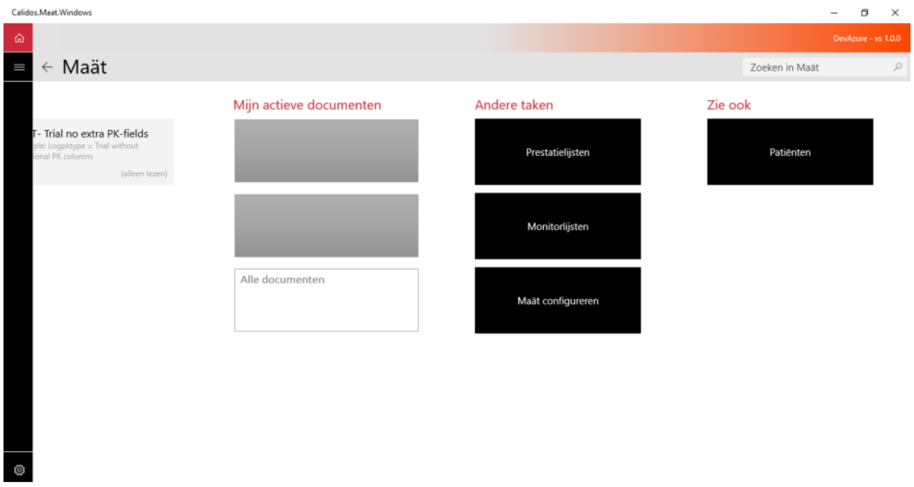
- Overlay:



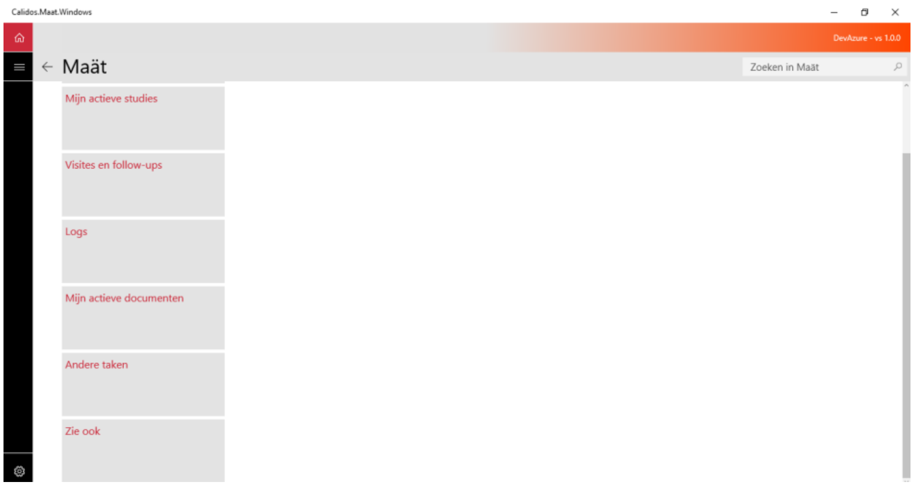
- Zoomed in & zoomed out (semantic zoom)

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:


Calidos.Maat.Windows

DevAzure - vs 1.0.0

Maät

Zoeken in Maät

Mijn actieve studies

Visites en follow-ups

Logs

Mijn actieve documenten

Andere taken

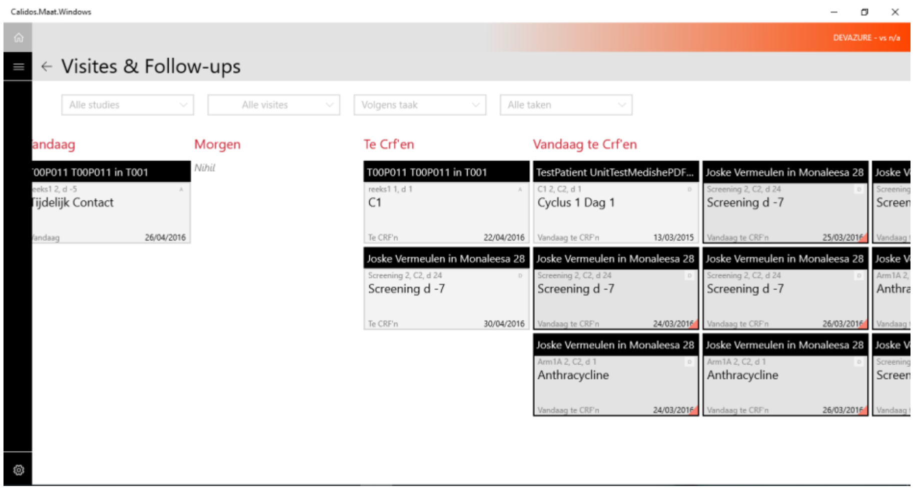
Zie ook

- Page filtering

“Page-filtering” contains all the different states of ordering a page can be in. For example, some elements in the page can be ordered by different factors like date, name,... Every different ordering state is a page filtering state

ex:

- By task filtering


Calidos.Maat.Windows

DEVAZURE - vs n/a

Visites & Follow-ups

Alle studies | Alle visites | Volgens taak | Alle taken

andaag

T00P011 T00P011 in T001
eeks1 2, d -5
Tijdelijk Contact
andaag 26/04/2016

Morgen

Nihil

Te Crf'en

T00P011 T00P011 in T001
reeks1 1, d 1
C1
Te CRF'n 22/04/2016

Joske Vermeulen in Monaleesa 28
Screening 2, C2, d 24
Screening d -7
Te CRF'n 30/04/2016

Vandaag te Crf'en

TestPatient UnitTestMedishePDF...
C1 2, C2, d 1
Cyclus 1 Dag 1
Vandaag te CRF'n 13/03/2015

Joske Vermeulen in Monaleesa 28
Screening 2, C2, d 24
Screening d -7
Vandaag te CRF'n 25/03/2016

Joske V
Screening
Screen
Vandaag

Joske Vermeulen in Monaleesa 28
Screening 2, C2, d 24
Screening d -7
Vandaag te CRF'n 24/03/2016

Joske Vermeulen in Monaleesa 28
Screening 2, C2, d 24
Screening d -7
Vandaag te CRF'n 26/03/2016

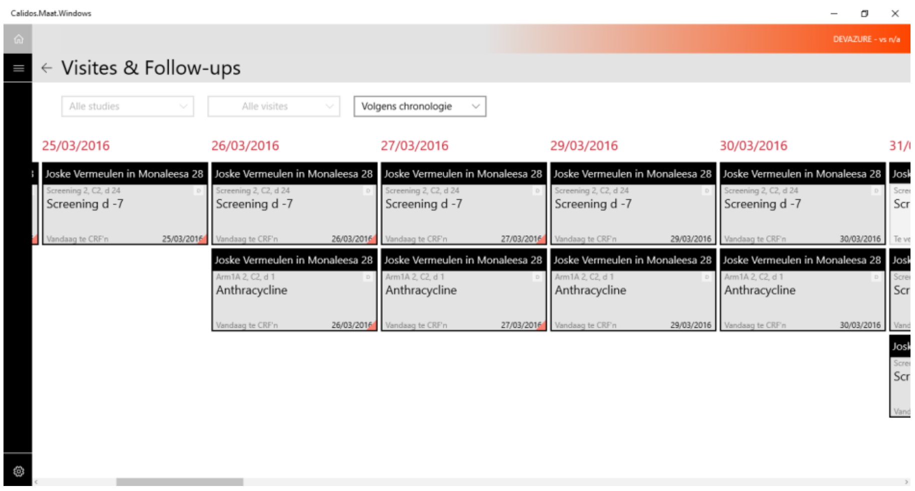
Joske V
Arm1A 2,
Anthra
Vandaag

Joske Vermeulen in Monaleesa 28
Arm1A 2, C2, d 1
Anthracycline
Vandaag te CRF'n 24/03/2016

Joske Vermeulen in Monaleesa 28
Arm1A 2, C2, d 1
Anthracycline
Vandaag te CRF'n 26/03/2016

Joske V
Screening
Screen
Vandaag

- By chronology filtering



- Multiselect

Multiselect is a state in which multiple elements can be selected. This state is not present inside any “Maät”-page, but it’s a state that would be possible inside a Windows 10 UWP-application, so we put it in our checklist.

3.1 No overlay state

3.1.1 Zoomed in state

Zoom out with Ctrl-

what?

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.

How?

1. Add the hubsection you want to zoom out to, to the UIMap
2. Add the name of the hubsection to the UIMap
3. Add the zoomed out listitem, which represents the hubsection, to the UIMap
4. Use following function:
AssertSemanticZoom(hubsection, hubsectionSemanticName, Action.CtrlKey);

Zoom out with Ctrl & mouse scroll

what?

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.

how?

Do the same as the above method, but use “Action.CtrlScroll” instead of “Action.CtrlKey”.

Zoom out with Tab/arrow select

what?

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.

how?

//PROBLEM: Cannot read whether the HubSection-titles are selected or not.

Zoom out with PgUp/PgDn select

what?

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.

how?

//PROBLEM: Cannot read whether the HubSection-titles are selected or not.

Zoom out with Mouseclick

what?

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

how?

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 hubsection content

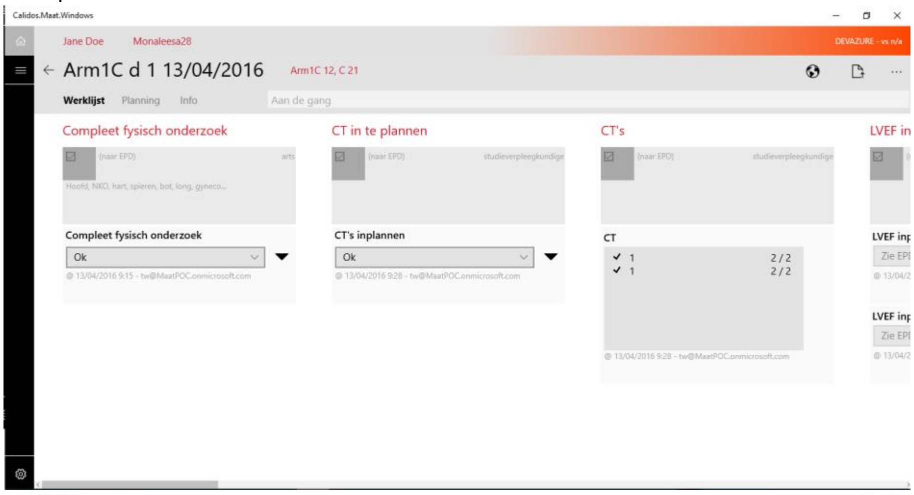
What?

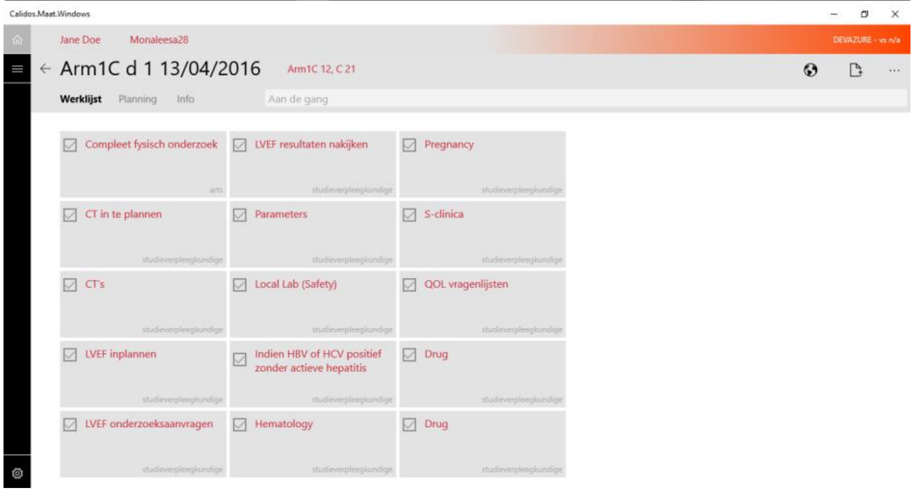
Check if correct data is displayed in hubsections when zoomed out:

- Checkboxes
- hubsection titles

On some pages, every element in the semantic zoom contains a checkbox, when zoomed out, the checkboxes on the zoomed out listItems have to be in the same state as the checkboxes on the zoomed in view.

example:





How?

- Checkboxes:

1. Add both lists (zoomed in and zoomed out view) to the UIMap. Sometimes these lists do not contain a UID, so make sure to change the searchproperties afterwards. To make sure your test method always finds the correct list, add the correct instance to the list (in the above case, both lists have instance “2”).
2. Add the semantic zoom to the UIMap
3. Get all the elements of the list by using following function of the baseclassCodedUI:

UITestControlCollection ZoomedInGroups = getHubSections(List);

4. Make a bool-array to store the content of all checkboxes, give it the length of the UITestControlCollection you just created
5. Store every checkboxes state in the array with a for function that loops over every element in your collection. The checkbox can be found by using the “GetChildren()” function several times. To know which instance to use and how deep you have to search in the hierarchy of your collection, use the CodedUITestBuilder to see the place of the desired checkbox inside the hiërarchy. In the case above, the checkbox can be found by getting the second child of every element in the collection, and getting the first child of that child. To store the state of the checkbox, you have to use the “.Checked”-property.
6. Zoom out using the “ZoomOutSemanticZoom(Action.CtrlScroll, SemanticZoom);” function.
7. Make a new UITestControlCollection for the zoomed out list using the same method as before.
8. Assert the lengths of both collections (should be the same!)
9. Assert the state of the zoomed out checkboxes is equals the bool-array you created earlier, with a for loop, that loops over every zoomed out ListItem. To find these checkboxes, use the same technique as described above (with GetChildren())
10. Zoom in the semantic zoom using following function: ZoomInSemanticZoom(ZoomedOutListItems[0] as XamlControl, Action.CtrlScroll, SemanticZoom);

- Hubsection titles

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()

note: this only works when the SemanticZoom contains a Hub with HubSections, if the elements of the SemanticZoom are different, you have to make a custom-function.

3.1.2 Zoomed out state

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 in with Ctrl+

what?

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.

how?

5. Add the hubsection you want to zoom out to, to the UIMap
6. Add the name of the hubsection to the UIMap
7. Add the zoomed out listitem, which represents the hubsection, to the UIMap
8. Use following function:

AssertSemanticZoom(hubsection, hubsectionSemanticName, Action.CtrlKey);

Zoom in with Ctrl and mouse scroll

what?

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.

how?

Do the same as the above method, but use “Action.CtrlScroll” instead of “Action.CtrlKey”.

Zoom in with Tab/arrow select

what?

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.

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

how?

Zoom in with PgUp/PgDn select

what?
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).

How?

Zoom in with Mouseclick on hubsection-titles

what?

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).

how?

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

3.1.3 Filtering search

- Check if Combobox ordering function orders the page correctly

//NOT YET ANALYSED//

3.1.4 Multi Select

Multiselect is a state in which you can select multiple items at once. The application Maät does not contain this state, but it is a possible state in a windows 10 metro application and needs to be tested if it’s there.

3.2 Overlay state

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 test method, 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 test method will fail to find the overlay in the first place.

Close overlay button

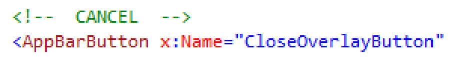
- what?

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).

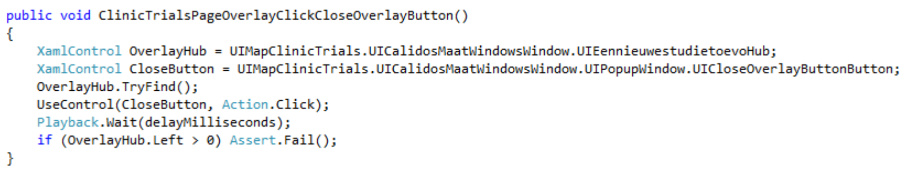
ex.



```
<!--  CANCEL  -->
<AppBarButton x:Name="CloseOverlayButton"
```

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

2. Add the overlay hub to the UIMap
3. Use the XamlControl.TryFind() function to make sure the overlay is opened.
4. Click the button & add delay to make sure the overlay has time enough to close completely
5. 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”.



```
public void ClinicTrialsPageOverlayClickCloseOverlayButton()
{
    XamlControl OverlayHub = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIEennieuwestudietoevoHub;
    XamlControl CloseButton = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIPopupWindow.UICloseOverlayButtonButton;
    OverlayHub.TryFind();
    UseControl(CloseButton, Action.Click);
    Playback.Wait(delayMilliseconds);
    if (OverlayHub.Left > 0) Assert.Fail();
}
```

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}”)

Enable execute button

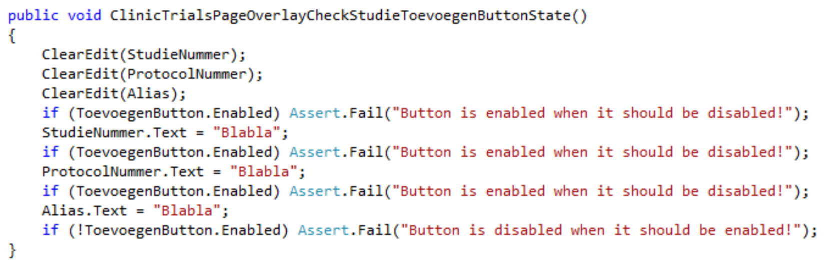
- What?

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

- How?

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

ex.



```
public void ClinicTrialsPageOverlayCheckStudieToevoegenButtonState()
{
    ClearEdit(StudieNummer);
    ClearEdit(ProtocolNummer);
    ClearEdit(Alias);
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    StudieNummer.Text = "Blabla";
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    ProtocolNummer.Text = "Blabla";
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    Alias.Text = "Blabla";
    if (!ToevoegenButton.Enabled) Assert.Fail("Button is disabled when it should be enabled!");
}
```

4. Control state appearance

In this section, we summarize all tests that check the visual appearance or state of a control. This includes things as the color, the fact that it is highlighted, the fact that it is enabled to be used, the fact that the mouse cursor changes when hovering over or clicking the control,...

4.1 Initial

what?

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

how?

color

The displayed color of the control.

//TO ANALYSE

highlighted

When a control is selected (ex through tab) we call it “highlighted”

To test this, assert to the controls “HasFocus”-property, which can be true or false.

enabled

“Enabled” means the control’s function are available to be used, when it’s disabled, the control is displayed but cannot be used.

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

Placeholder text
The default text is the text you sometimes see in a textbox that disappears as soon as you start typing. It's a text indicating the type of data you have to put in the textbox (bvb "*name")*
//TO ANALYSE

default value
The initial value displayed by a control
Assert to the controls "Text" or "Value" property.

4.2 Hovered

What?

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

how?

Use "Mouse.Hover" to hover over the desired control.

Mouse/cursor change

//TO ANALYSE

color

//TO ANALYSE

highlighted

To test this, assert to the controls "HasFocus"-property, which can be true or false.

enabled

assert to the "*Control*.Enabled"-property, which is a boolean.

Placeholdertext

//TO ANALYSE

default value

Assert to the controls "Text" or "Value" property.

4.3 Clicked

What?
Check of the control's state changes correctly when clicking on the control.

How?

Use "Mouse.Click" or the baseclass function "UseControl" to click on the desired control.

Mouse/cursor change

//TO ANALYSE

color

//TO ANALYSE

highlighted

To test this, assert to the controls “HasFocus”-property, which can be true or false.

enabled

assert to the “*Control*.Enabled”-property, which is a boolean.

Placeholdertext

//TO ANALYSE

default value

Assert to the controls “Text” or “Value” property.

4.4 Click & hold

What?

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

How?

Use “Mouse.StartDragging” function to press the left mouse button and holding it down.

Mouse/cursor change

//TO ANALYSE

color

//TO ANALYSE

highlighted

To test this, assert to the controls “HasFocus”-property, which can be true or false.

enabled

assert to the “*Control*.Enabled”-property, which is a boolean.

Placeholdertext

//TO ANALYSE

default value

Assert to the controls “Text” or “Value” property.

Control deepens?

This means that the control will display a subtle animation when clicked, which in this case is that the control moves deeper into the screen.
//TO ANALYSE

4.5 Filled in

What?

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

How?

Fill the control by changing its "Text" or "Value" property.

color

highlighted

To test this, assert to the controls "HasFocus"-property, which can be true or false.

enabled

assert to the "*Control*.Enabled"-property, which is a boolean.

value

Assert to the controls "Text" or "Value" property.

5. Functionality

In this section, we will discuss how to test all the possible functionalities a control can have, other than navigating to another page.

5.1 General

These are the general functionalities of every control, which include its selectability by using tab or arrows, as well as its ability to execute its function with a click, enter or space action. Some controls will be usable with enter/space while other don't. The same is true for its selectability.

Click selection

What?

Check if the control is selectable with a click action.

how?

1. Click on the control
2. assert on the "*control*.HasFocus" property. If the control is selected, this property should be true.

tab & arrow selection

what?

Check if the control is selectable through tabbing and using the arrow keys

how?

1. Go to the control by using the BaseClass-functions TabToControl() and GoToControl()
2. assert on the "*control*.HasFocus" property. If the control is selected, this property should be true.

Click execution

what?

check if the control can be used with click action.

how?

1. Click on the control
2. Assert to the action the control was supposed to execute.

Enter execution

what?

check if the control can be used with enter action

how?

1. Make sure the control is selected
2. Hit enter
3. Assert to the action the control was supposed to execute.

Space execution

what?

check if the control can be used with space action

how?

1. Make sure the control is selected
2. Hit space

Assert to the action the control was supposed to execute.

5.2 Execution functionality

In this part we will discuss every possible function a control can have. In this section it does not matter whether we use click/enter/space, but it matters what happens when we use one of these actions.

Partpicker data selection with arrow-keys

what?

check if the partpicker section changes the number with up and down arrow keys (when selected)

how?

//PROBLEM: cannot access the popup window, nor the different tabs inside of it through code. Only the checkmark and cross button are accessible.

Partpicker data selection with scroll wheel

what?

check if the partpicker section changes the number with the scroll wheel (when selected)

how?

//PROBLEM: cannot access the popup window, nor the different tabs inside of it through code. Only the checkmark and cross button are accessible.

Open popup window

what?

Check if using the control opens up a pop up window

how?

1. Add the popup-window to the UIMap by using the “Ctrl-i” combination
2. Use the control
3. Assert the popup-window exists with the coördinate properties, these should be set to -1 if the popup-window isn’t visible. Sometimes the test will even fail when it’s trying to find the popup-window when it's not visible.

Close popup window with control

what?

check if using the control closes the popup window

how?

1. Add the control to the UIMap by using the “Ctrl-i” combination
2. Use the control
3. Assert the popup-window is gone with the coördinate properties, these should be set to -1 if the popup-window isn’t visible. Sometimes the test will even fail when it’s trying to find the popup-window when it's not visible.

Close popup window by clicking elsewhere

what?

When a popup-window is opened, and you click elsewhere with the mouse, the popup-window should close automatically.

how?

1. Make sure the popup-window is opened
2. Click on a hard coded coördinate, that is outside of the popup-window
3. Assert the popup-window is gone with the coördinate properties, these should be set to -1 if the popup-window isn’t visible. Sometimes the test will even fail when it’s trying to find the popup-window when it's not visible.

Close popup window with “esc”

what?

When a popup-window is opened, and you click elsewhere with the mouse, the popup-window should close automatically.

how?

4. Make sure the popup-window is opened
5. press the “esc”-key
6. Assert the popup-window is gone with the coördinate properties, these should be set to -1 if the popup-window isn’t visible. Sometimes the test will even fail when it’s trying to find the popup-window when it's not visible.

Check data change in control

what?

Sometimes, using a control involves the input of some data, and after executing, the control will display the data you put in. The test is to check if inputting the data and executing results in the data you put in, displayed in your control.

how?

1. Execute the data change action, which can be typing text, selecting partpicker numbers,...
2. Make sure you save the data you put in as a variable in your test
3. Perform a read-test on the control, looking for your saved variable

Control-visibility changing

what?

check if using the control makes the correct other control’s appear or disappear

how?

1. Use the control
2. Assert to the coördinates of the controls that should appear or disappear

Control reactivity

what?

check if you can use the control before the page is fully loaded

how?

1. Use the control you want to check the reactivity on, as you would do it to check the functionality of the control. Make sure the code is written in such a way that the control is used as soon as possible.
2. Check it the progressbar on top of the screen is still visible after the previous code. If it isn’t, write an Assert.Fail() function.

Control scrolling functions

scroll wheel

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

click and drag scrollbar

what?

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

How?

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

click on scrollbar arrows

what?

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

How?

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

Tooltip

what?

Check if hovering over the control makes a tooltip appear

How?

//NOT YET ANALYSED

Toggle state

what?

Check if using the control changes it’s toggle state

how?

1. Use the control
2. Assert on the *control*.Pressed property

Clipboard functions

Add the clipboard popup menu to the UIMap by using “Ctrl-i”

Cut

what?

Check if choosing “cut” on selected text makes the selected content disappear but still available to paste elsewhere.

how?

1. Select the text inside the control (still to analyse how to do this)
2. Right click on the text and click on the clipboard function: cut
3. Assert the correct actions happened with the control’s content by using the “*control*.Text”-property

Copy

what?

Check if choosing “copy” on selected text makes the selected content available to paste elsewhere.

how?

1. Select the text inside the control (still to analyse how to do this)
2. Right click on the text and click on the clipboard function: copy
3. Assert the correct actions happened with the control’s content by using the “*control*.Text”-property

Paste

what?

Check if choosing “paste” makes content you cut or copied, appear in the control you chose to paste in. Make sure to check if the content that was already in the control stays in the control, unless you selected it first.

how?

1. Right click in the control and click on the clipboard function: paste
2. Assert the correct actions happened with the control's content by using the "*control*.Text"-property
3. Do the same as above but first select a part of the already existing content in the control
4. Assert to the content again

Undo

what?

Check if choosing "undo" makes the content inside the control you are working in disappear.

how?

1. Right click on the control
2. click on the clipboard function: undo
3. Assert the correct actions happened with the control's content by using the "*control*.Text"-property

Select all

what?

Check if choosing "select all" selects all content inside of the control you are working in

how?

1. Right click on the control
2. click on the clipboard function: select all
3. Assert the correct actions happened with the control's content by using the "*control*.Text"-property

Clear control's content

what?

Check if clicking the "x" in the control sets it back into default state

how?

1. Click on the "x"-button inside the control
2. Check if the control is cleared with the "*control*.Text"-property
3. Check if the control is back into default state by executing "Control state appearance" tests

Validation text

what?

Check if inputting the wrong content into the control triggers the appearance of a validation text, which is a red line of text with a warning message.

how?

Perform a “Control visibility changing” test after inputting the wrong content

Default selected control

what?

Sometimes, there is a control selected by default, without performing any further actions. Check if this control is selected when it should be.

how?

1. Create the correct conditions for the control to be selected
2. Assert to the “*control*.HasFocus”-property

Typing content into selected control

what?

Check if typing some content when your control is selected, results in the content you typed appearing inside the control you are testing

How?

1. Select the control
2. Use the “WriteAsUser(*content*)” function
3. Assert to the “*control*.Text”-property

Typing content randomly into control

what?

Sometimes, when typing randomly, the content you are typing will appear in a control by default. Check this functionality

how?

Do the same as above, but without selecting the control

Clear control with “Esc

what?

Some controls can be cleared by pressing the “esc” key. Test this functionality

how?

1. Make sure there is content inside the control you are testing.
2. Hit the “Esc”-key
    3. Assert to the “*control*.Text”-property

Search with magnifying glass

what?

Inside a SearchBox, there is a magnifying glass. By clicking it, you will execute the same search action as if you were hitting enter. Test this functionality.

how?

//PROBLEM: magnifying glass is not accessible through any other way than manual

Select control-item with arrow keys

what?

Check if using the arrow-keys on a selected control (up and down-keys) changes the control's selected item

how?

1. Select the control with tab.
2. Press the arrows up or down.
3. Assert that the content of the control has changed correctly.

Select control-item by clicking

what?

By clicking on the control, you open a dropdown-menu and can select an item inside of it. The drop down menu will disappear and your selected item will appear inside the control. Test this functionality.

how?

//TO ANALYSE

6. Custom

Because in all the config pages, there are much hiërarchic similarities, we decided to make one global config-UIMap, in which we put controls that can be found with similar properties on every config-page. For page-specific properties, we will make a separate UIMap.
note that this section is not yet fully analysed and finished

6.1 Dropdown grid

All Config-pages have 2 dropdown-buttons in the titlebar. Both of them open up a dropdown-window.

The first dropdown shows a list of different cases of the pages context (ex StageConfig --> different stages, PathConfig → different paths,...)

The second dropdown-window shows a list of versions we can use.

6.1.1 Click popup-window item

What?
Click on one of the items inside the popup-window and assert the correct action happened after you clicked.

How?
1. Add the listItem to the UIMap. Note that it is placed under the wrong hiërarchy (wrong list).
2. To make sure you can find the control afterwards, add the SearchProperty “ClassName” to it, with the correct class-name.
3. Click the control
4. perform a read function to assert the correct elements are changed in the UI

6.2 Custom control bar

6.2.1 Change workbook state

Open overlay

What?

You can work in a workbook, and make as many changes as you want. After that you can lock the changes you made, validate them and publish them. All three functions are done with an overlay

how?

Refer to: States → Overlay state → Open overlay button

6.3 OptionsListView
In every config-page, there is an options list on the left of the screen. From there on, you can either choose the Info option or all the other options. The info displays info about this particular config-page. The rest of the options shows a list of other elements, which either modify data or contain objects of data you can modify.

Testing Maät

In this section, we will describe the workflow of the testing of Maät, this includes a general method and a specific description of the process we went through for each page.

Note that +-40% of all WorkbookConfig-tests are written, all other tests (ClinicHub, …) are outdated and should be looked at to make them up to date again.

General work method

Here we will describe the general method to follow to make sure a page is fully tested.
It consists of 4 steps:

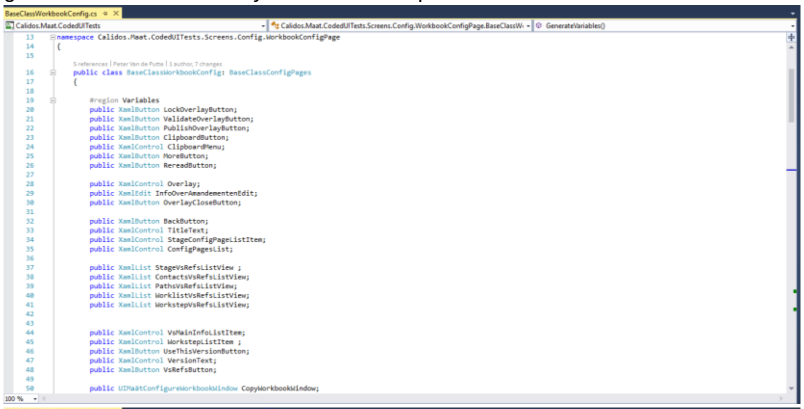
1. Analyse
2. Expand checklist
3. Map
4. Test

Analyse/expand checklist/map

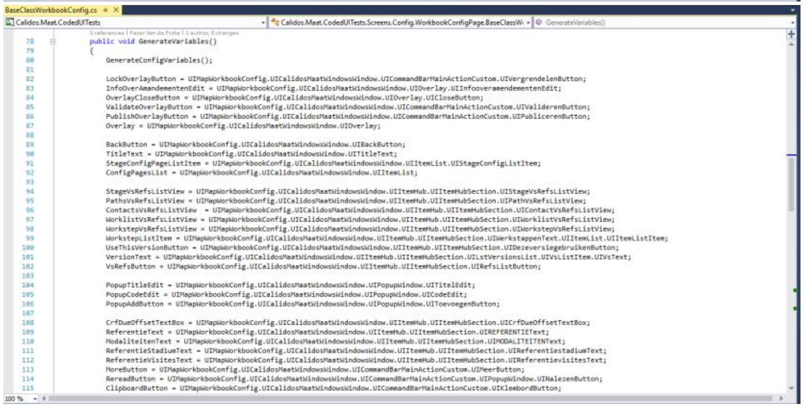
These 3 steps have a lot of influence on each other. To analyse means that you manually search for all possible functionalities on a page by trying out all possibly usable controls in a number of different ways (ex. clicking, tabbing to the control, hovering over it, watching what happens when it's used,...). Simultaneously, you map all these usable controls to the checklist and add a "To do" to the test that has to be performed on this control. When you find that this functionality is not yet documented in the checklist, you add it to the checklist and analyse how you can test it (this is the "Expand checklist" step. Then you map all used controls to the UIMap with the UIMap test builder. How to do this is explained in the "How to test"-chapter. Sometimes however, the controls are badly mapped by the UIMap test builder. When this happens, you can try some of the following techniques:

1. Map the parent control and search for your desired control by using the "GetChildFromList"-function
2. Add a UID to the control in the XAML file (how to do this is explained in "How to test"
3. Change the searchproperties of the mapped control (for example instance, name,...) in such a way that when searching the control, the program will logically end up on the control you desire. This technique can also be used inside a test to customize your tests. For example, if there is a control on several sections of a page that looks visually the same, but is logically different in each section, you can write one tests in which you change the SearchProperty according to the different sections.

In the map-step, you also have to write a BaseClass for every page, in which you generate the controls you are about to use. This maintains the readability of every test project, because you generate each control only once. An example of this:


BaseClassWorkbookConfig.cs
Calidos.Maat.CodedUITests
Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.BaseClassW
GenerateVariables()
5 references | Peter Van de Putte | 1 author, 7 changes
100 %

```csharp
namespace Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage
{

    public class BaseClassWorkbookConfig: BaseClassConfigPages
    {

        #region Variables
        public XamlButton LockOverlayButton;
        public XamlButton ValidateOverlayButton;
        public XamlButton PublishOverlayButton;
        public XamlButton ClipboardButton;
        public XamlControl ClipboardMenu;
        public XamlButton MoreButton;
        public XamlButton RereadButton;

        public XamlControl Overlay;
        public XamlEdit InfoOverAmandementenEdit;
        public XamlButton OverlayCloseButton;

        public XamlButton BackButton;
        public XamlControl TitleText;
        public XamlControl StageConfigPageListItem;
        public XamlControl ConfigPagesList;

        public XamlList StageVsRefsListView ;
        public XamlList ContactsVsRefsListView;
        public XamlList PathsVsRefsListView;
        public XamlList WorklistVsRefsListView;
        public XamlList WorkstepVsRefsListView;


        public XamlControl VsMainInfoListItem;
        public XamlControl WorkstepListItem ;
        public XamlButton UseThisVersionButton;
        public XamlControl VersionText;
        public XamlButton VsRefsButton;

        public UIMaätConfigureWorkbookWindow CopyWorkbookWindow;
```


BaseClassWorkbookConfig.cs
Calidos.Maat.CodedUITests
Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.BaseClassW
GenerateVariables()
5 references | Peter Van de Putte | 1 author, 6 changes
100 %

```csharp
        public void GenerateVariables()
        {
            GenerateConfigVariables();

            LockOverlayButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIVergrendelenButton;
            InfoOverAmandementenEdit = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIOverlay.UIInfooveramendementenEdit;
            OverlayCloseButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIOverlay.UICloseButton;
            ValidateOverlayButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIValiderenButton;
            PublishOverlayButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIPublicerenButton;
            Overlay = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIOverlay;

            BackButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIBackButton;
            TitleText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UITitleText;
            StageConfigPageListItem = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemList.UIStageConfigListItem;
            ConfigPagesList = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemList;

            StageVsRefsListView = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIStageVsRefsListView;
            PathsVsRefsListView = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIPathVsRefsListView;
            ContactsVsRefsListView  = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIContactVsRefsListView;
            WorklistVsRefsListView = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIWorklistVsRefsListView;
            WorkstepVsRefsListView = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIWorkstepVsRefsListView;
            WorkstepListItem = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIWerkstappenText.UIItemList.UIItemListItem;
            UseThisVersionButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIDezeversiegebruikenButton;
            VersionText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UILstVersionsList.UIVsListItem.UIVsText;
            VsRefsButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIRefsListButton;

            PopupTitleEdit = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIPopupWindow.UITitelEdit;
            PopupCodeEdit = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIPopupWindow.UICodeEdit;
            PopupAddButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIPopupWindow.UIToevoegenButton;

            CrfDueOffsetTextBox = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UICrfDueOffsetTextBox;
            ReferentieText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIREFERENTIEText;
            ModaliteitenText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIMODALITEITENText;
            ReferentieStadiumText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIReferentiestadiumText;
            ReferentieVisitesText = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UIItemHub.UIItemHubSection.UIReferentievisitesText;
            MoreButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIMeerButton;
            RereadButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIPopupWindow.UINalezenButton;
            ClipboardButton = UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UICommandBarMainActionCustom.UIKlembordButton;
```

In this project, all the variables we will use once or multiple times, are generated. In every test project we will execute the function “GenerateVariables()” in the initialize-function. Also every private function we use in test projects can be placed in this project.

The hiërarchy is as following: *PageParadigm* (ex ClinicHubStates) inherits from BaseClass*Page* (inherits from BaseClass*Category,* sometimes) inherits from BaseClassCodedUI.

Test

When all the previous is done, you can start writing all the tests you just analysed. Note that sometimes, different tests can be embedded in one test, like selecting and using a control, or navigation forward and backward. This is why sometimes more than one square in the checklist will be checked after writing one test.

ConfigPages

General problems

UIMapping

For all the tests (every category), the biggest problem is finding the control's you want to test. Because the UIMap-builder is not yet fully functional with windows 10 UWP applications, and some control's contain uIDs while others don't, many controls are mapped in a hiërarchy that does not resemble the real hiërarchy of the application. For example, some controls are mapped as children of one control, while they are really siblings. Or some controls are mapped directly under the Window, while they are children of for example a popup-window.

General work method

UIMapping

In the config-pages, many controls come back with different values. This is why we made a general UIMap for all the config-pages, in which we put the controls that are available in every config-page. To make them accessible in every page, we sometimes had to change the searchproperties (for example remove the uID or add a certain instance). When doing this, make sure to first test this in every page with the UIMap-builder (you can click "search control" to make the UIMap-builder re-highlight the control you added). We also sometimes changed the SearchProperty "uID" to a part of the ID of the control, and change the property to "contains" instead of "equals"

For all the controls that are unique to one or some (but not all) pages we made a UIMap for every page.

Methods

For certain tests that will come back in every config-page, we wrote a method in which you have to pass some variables to specify in which page the test is executed or which page-specific controls are involved. We tried to write these methods as autonomous as possible. For the config-pages, we have a separate BaseClassCodedUI-partial class.

TestInitialize

Because navigating to the page directly results in a page without data (invalid), we have written the Testinitialize-method in such a way that it navigates to the TrialHub, with trialID 19 (which is Monaleesa28 or the trial with the most complete set of data). This trial is ideal for testing. After navigating to this trial, we wrote to search for the button that navigates to WorkbookConfig and clicking it. Afterwards we make the test wait until the page is fully loaded because much functionality in the ConfigPages only works optimally after the ProgressBar has disappeared.

For every config-page, we will write an extra line in the testinitialize-function to navigate from WorkbookConfig to this config-page.

WorkbookConfig

General problems

Note that the hiërarchy of the controls inside the HubSection varies from section to section although the visual representation is the same.

General work method

In the different option-sections of WorkbookConfig alone, we also found some controls that are always available, with different values. So here, we also added them to the UIMap with custom searchproperties.

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.

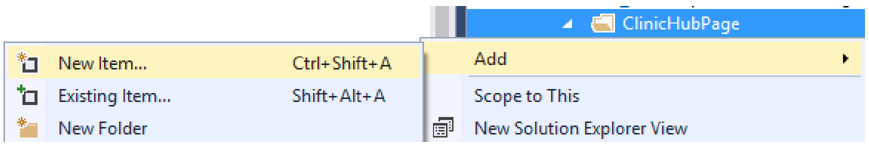
We will also discuss what steps we followed on each page to come to a correct test project.

Basics

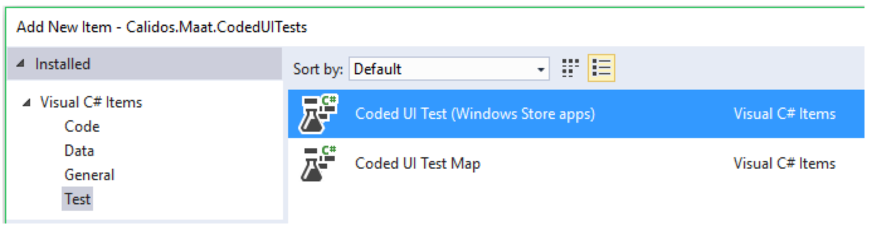
Coded UI Test Builder and UI Map

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)'

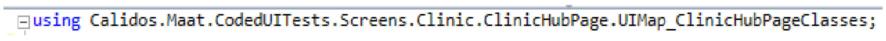

Add New Item - Calidos.Maat.CodedUITests
Installed
Visual C# Items
Code
Data
General
Test
Sort by: Default
Coded UI Test (Windows Store apps) Visual C# Items
Coded UI Test Map Visual C# Items

To add a UI map:

- Do the same
- Select 'Coded UI Test Map' instead of 'Coded UI Test (Windows Store apps)'

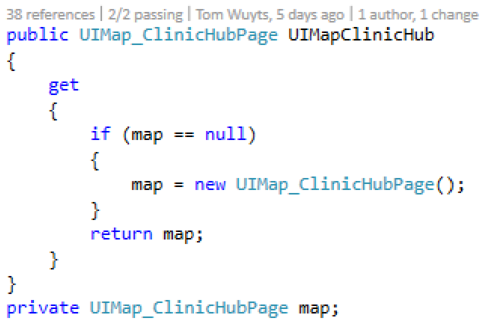
When you added a Coded UI Test project, the first thing you need to do is add the right UI Map as a variable in your test project:

- At the top of your project, add a "using" statement for the UI Map you created. If we named our UI Map "UIMap_ClinicHubPage", this is what we would have to add:



```
using Calidos.Maat.CodedUITests.Screens.Clinic.ClinicHubPage.UIMap_ClinicHubPageClasses;
```

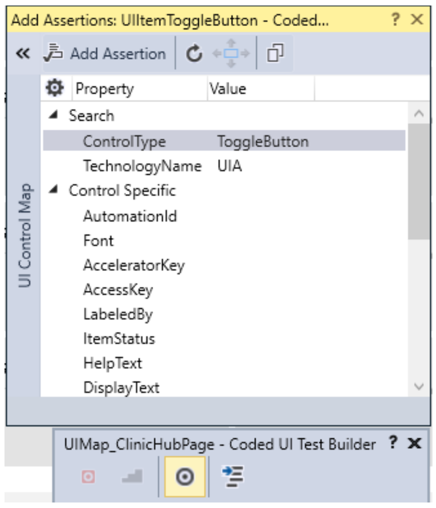
- At the bottom of your project, change the UI Map property to something like this:



```
38 references | 2/2 passing | Tom Wuyts, 5 days ago | 1 author, 1 change
public UIMap_ClinicHubPage UIMapClinicHub
{
    get
    {
        if (map == null)
        {
            map = new UIMap_ClinicHubPage();
        }
        return map;
    }
}
private UIMap_ClinicHubPage map;
```

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. Don't 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.



Add Assertions: UIItemToggleButton - Coded...

Property	Value
Search	
ControlType	ToggleButton
TechnologyName	UIA
Control Specific	
AutomationId	
Font	
AcceleratorKey	
AccessKey	
LabeledBy	
ItemStatus	
HelpText	
DisplayText	

UIMap_ClinicHubPage - Coded UI Test Builder

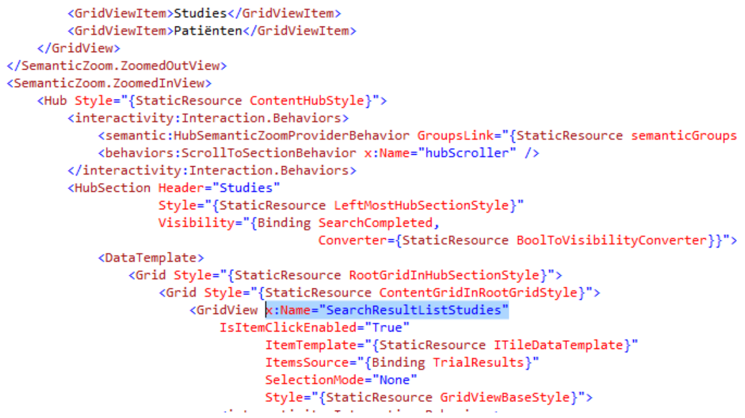
The control you selected will be highlighted within the application with a blue border and on the right side of the new window you can see more detailed information about the currently selected control. If you click the arrow button in the top left hand corner you can see the hiërarchy in which the control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

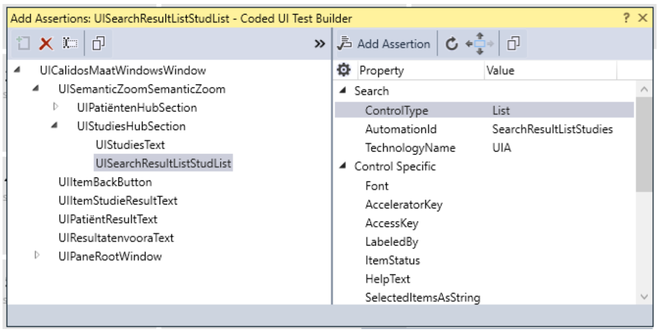
To make sure control's or lists are properly mapped and easy to find by the test program itself, we sometimes gave AutomationIDs to the control's ourselves. To do this, open the XAML file of the page you want to test, and then search for an indication out of which you can derive this section is the section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

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



```
            <GridViewItem>Studies</GridViewItem>
            <GridViewItem>Patiënten</GridViewItem>
        </GridView>
    </SemanticZoom.ZoomedOutView>
    <SemanticZoom.ZoomedInView>
        <Hub Style="{StaticResource ContentHubStyle}">
            <interactivity:Interaction.Behaviors>
                <semantic:HubSemanticZoomProviderBehavior GroupsLink="{StaticResource semanticGroups
                <behaviors:ScrollToSectionBehavior x:Name="hubScroller" />
            </interactivity:Interaction.Behaviors>
            <HubSection Header="Studies"
                        Style="{StaticResource LeftMostHubSectionStyle}"
                        Visibility="{Binding SearchCompleted,
                                             Converter={StaticResource BoolToVisibilityConverter}}">
                <DataTemplate>
                    <Grid Style="{StaticResource RootGridInHubSectionStyle}">
                        <Grid Style="{StaticResource ContentGridInRootGridStyle}">
                            <GridView x:Name="SearchResultListStudies"
                                IsItemClickEnabled="True"
                                      ItemTemplate="{StaticResource ITileDataTemplate}"
                                      ItemsSource="{Binding TrialResults}"
                                      SelectionMode="None"
                                      Style="{StaticResource GridViewBaseStyle}">
```

We wanted to give a unique Id to the list of Studies-search results, so we searched the XAML file for a while, tried naming some different grids and grid views, until we named the right one. Now if we select that list with the Coded UI Test Builder, the name we gave to it will appear as AutomationId, as seen below.



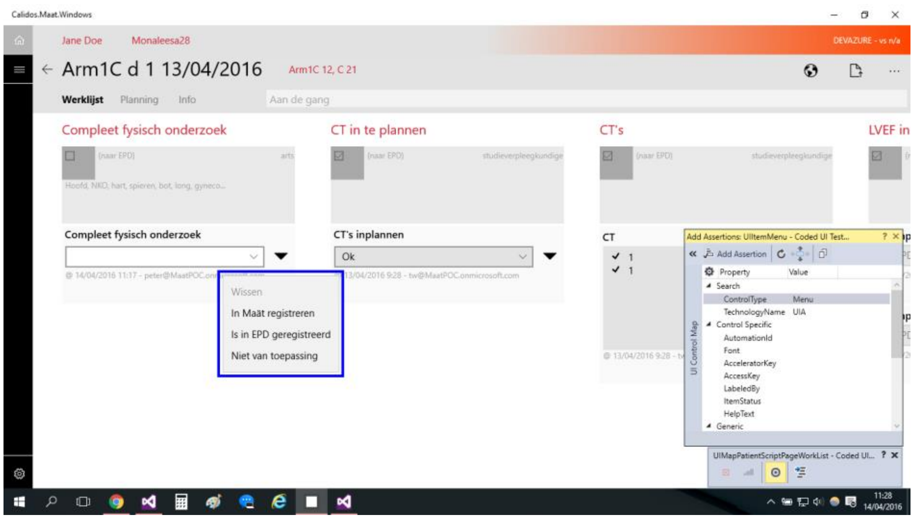
Add Assertions: UISearchResultListStudList - Coded UI Test Builder

```
UICalidosMaatWindowsWindow
    UISemanticZoomSemanticZoom
        UIPatiëntenHubSection
        UIStudiesHubSection
            UIStudiesText
            UISearchResultListStudList
    UIItemBackButton
    UIItemStudieResultText
    UIPatiëntResultText
    UIResultatenvooraText
    UIPaneRootWindow
```

Property	Value
Search	
ControlType	List
AutomationId	SearchResultListStudies
TechnologyName	UIA
Control Specific	
Font	
AcceleratorKey	
AccessKey	
LabeledBy	
ItemStatus	
HelpText	
SelectedItemsAsString	

Adding popup controls to the UIMap

Sometimes, we have to add control's or menu's to the UIMap that are only visible after clicking a certain button/control. If we just drag the crosshair of the CodedUITestBuilder onto the screen, the popup menu will disappear. A solution for this is using the "Ctrl-i"-combination while hovering over the menu, which will directly select the menu.



Test Methods

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 specific, 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.

```
/* Basic TestMethod skeleton:

[TestMethod]
public void METHODNAME()
{

}

*/
```

The ‘[TestInitialize]’ tag

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

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

The ‘[TestCleanup]’ tag

This will be executed **after every** ‘[TestMethod]’. The same as with the TestInitialize, only after each and every test method. 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, because we don't want to reload all the data every few seconds for the UI tests.

The ‘[DataSource()]’-tag

When you want to create a data driven test (which is a test using external data for execution/assertion/…), this tag can be added on top of the test method, 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 data connection (ex. CSV, XML,...)
- path to datafile
- table name
- access method (sequential/random)

If you add a datatable to a test method, the method will be executed for every row in the table. The first row will be ignored and can be used to assign variable names to your data for easy accessibility.

Testcontext

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

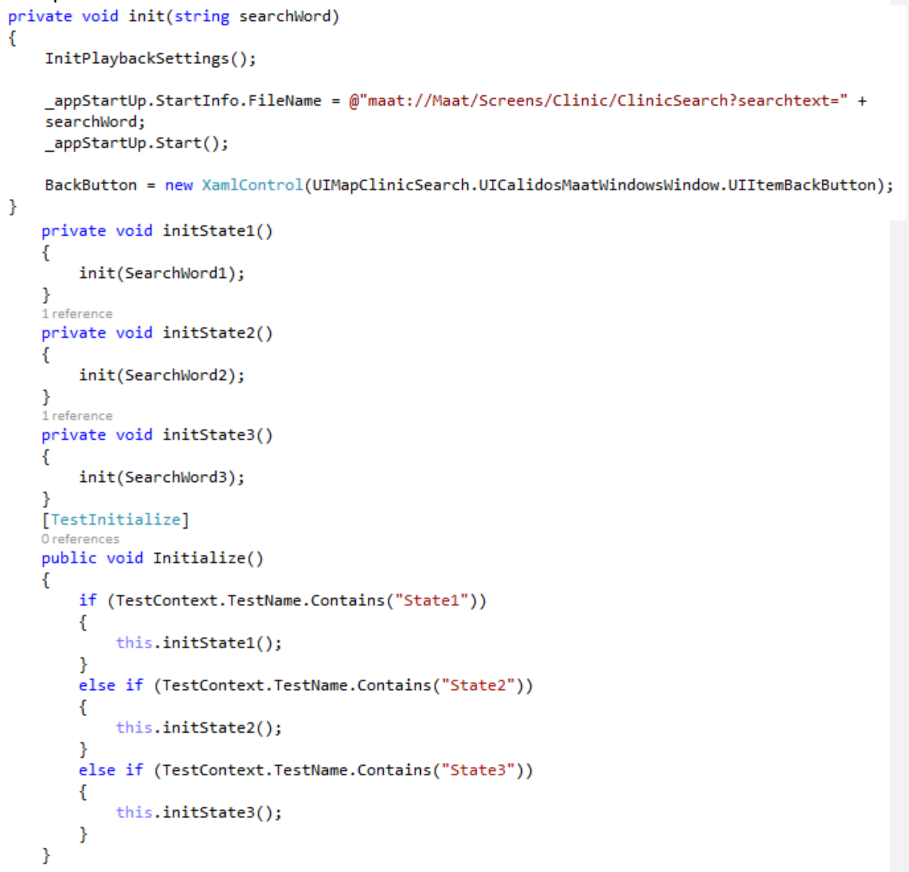
Data

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 & different TestInitialize-tags

In some test project, 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:



```
private void init(string searchWord)
{
    InitPlaybackSettings();

    _appStartUp.StartInfo.FileName = @"maat://Maat/Screens/Clinic/ClinicSearch?searchtext=" +
    searchWord;
    _appStartUp.Start();

    BackButton = new XamlControl(UIMapClinicSearch.UICalidosMaatWindowsWindow.UIItemBackButton);
}
    private void initState1()
    {
        init(SearchWord1);
    }
    1 reference
    private void initState2()
    {
        init(SearchWord2);
    }
    1 reference
    private void initState3()
    {
        init(SearchWord3);
    }
    [TestInitialize]
    0 references
    public void Initialize()
    {
        if (TestContext.TestName.Contains("State1"))
        {
            this.initState1();
        }
        else if (TestContext.TestName.Contains("State2"))
        {
            this.initState2();
        }
        else if (TestContext.TestName.Contains("State3"))
        {
            this.initState3();
        }
    }
```

Generating controls

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

- 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 — Text Font (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 properties (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.

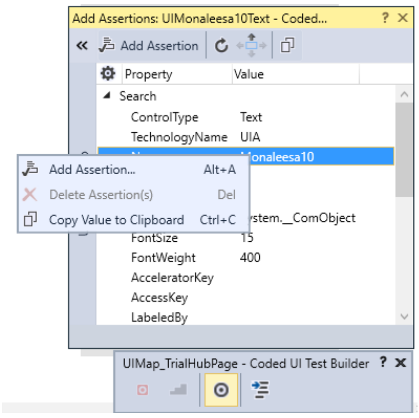
```
XamlText homeText = UIMapClinicHub.UICa
```

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 make an assertion for a specific property, by selecting it and clicking “Add Assertion” in the 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.

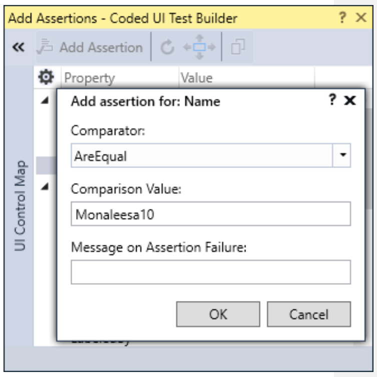

Add Assertions: UIMonaleesa10Text - Coded...
Add Assertion
Add Assertion... Alt+A
Delete Assertion(s) Del
Copy Value to Clipboard Ctrl+C
UIMap_TrialHubPage - Coded UI Test Builder

Property	Value
Search	
ControlType	Text
TechnologyName	UIA
	Monaleesa10
	ystem.__ComObject
FontSize	15
FontWeight	400
AcceleratorKey	
AccessKey	
LabeledBy	

can

top

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


Add Assertions - Coded UI Test Builder

Add assertion for: Name

Comparator: AreEqual

Comparison Value: Monaleesa10

Message on Assertion Failure:

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

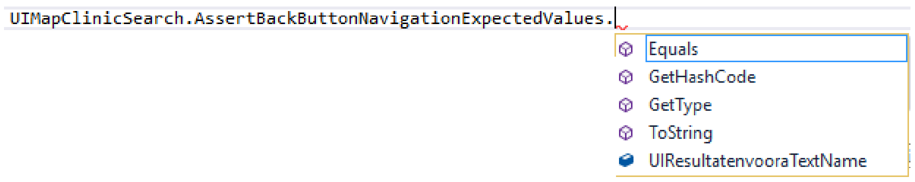
The comparison value is the value you want the actual value to be, in order to let the assertion pass. In other words, the test framework will get the actual value and check it against the comparison value

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

```
UIMapClinicSearch.AssertBackButtonNavigation();
```

The UI Map in the example is the UI Map of the test project, 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 test methods or within the partial UI Map class (The part that doesn't contains auto-generated code). Changing the expected value can be done like below:


```
UIMapClinicSearch.AssertBackButtonNavigationExpectedValues.
```

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 available when making a manual assertion ('Contains' for example is not available when using manual assertions). The comparator requires parameters to create the full assertion.

```
Assert.AreEqual(true, hub.TryFind());
```

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 values come from, is completely under your control. This way of writing assertions is way more intensive than the auto-assertions, but makes room for some more flexibility. Manual assertions are written inside your test methods.

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.

```
public void ClinicTrialsPageOverlayClickCloseOverlayButton()
{
    XamlControl OverlayHub = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIEennieuwestudietoevoHub;
    XamlControl CloseButton = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIPopupWindow.UICloseOverlayButtonButton;
    OverlayHub.TryFind();
    UseControl(CloseButton, Action.Click);
    Playback.Wait(delayMilliseconds);
    if (OverlayHub.Left > 0) Assert.Fail();
}
```

Commonly used variables and methods

These are some variables and methods that already exist, and that we will use to write some of our tests.

StopWatch

This is an object with which you can time how long it takes for a certain amount of code to execute. To use it, add "System.Diagnostics" to your usings. You can then generate a new StopWatch. To start timing, write StopWatch.Start(), and to read how many time has passed, use the StopWatch.ElapsedMilliseconds-property. Write your code between the StopWatch.Start()-function and StopWatch.Stop()-function.

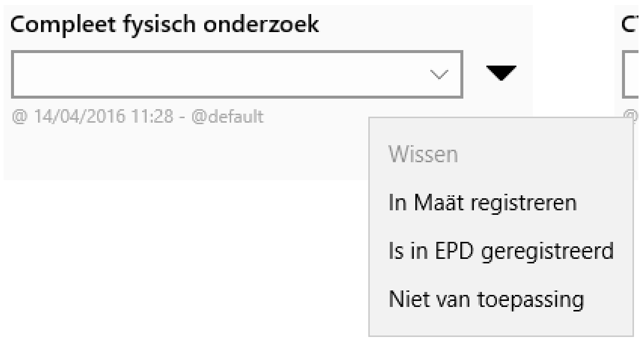
Commonly used controls

These are some control's that we found in the application, for which you have to use certain techniques to use them.

Flyout button

These are buttons that make a popup-menu appear when clicking on it. In this popup-menu, you can choose several options, each resulting in a different change somewhere else in the application.

ex.



The thick black arrow is a flyout button. Clicking one of the options in the popup-menu, has an influence on the combobox below "compleet fysisch onderzoek".

To add the menu appearing after clicking the flyout button to the UIMap, you can't drag the crosshair of the UIMap-builder to the menu, because clicking elsewhere makes the menu disappear. Instead, hover over the menu with your mouse and use the "Ctrl-i" combination.

ProgressBar

Sometimes, a control is already loaded and enabled before a page is fully loaded. However, the desired functionality of the control will only work correctly after the page is fully loaded. In that case, it's necessary to build some kind of waiting function to wait for the page to be fully loaded.

This is where the ProgressBar comes in handy. It's a bar on top of the screen, which displays an animation as long as the page is loading. As soon as the page is loaded, the bar disappears.

To setup a waiting function for the page to be fully loaded, you can add the ProgressBar to the UIMap with the "Ctrl-I" key combination (do this while a page is still loading, before it disappears). Afterwards, you can use one of the coördinates of the ProgressBar to build an empty while loop. The coördinates of a control that has disappeared are set to "-1", so your while-loop will look something like this:

"while (ProgressBar.Top > 0) ;"

Note that no code is executed in the while-loop, it is just there to wait for the condition you desire to occur.

BaseClassCodedUI

This is a base class 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 test scenarios (that don't require page-specific control's but overall control's, available on every page) to this class.

Every test project that is created, has to inherit from this base class.

General

enum Action

This is an enum of actions that we can use for all methods that involve performing some sort of action on a control. This enhances the readability of the method.

```
public enum Action
{
    Enter,
    Space,
    Click,
    CtrlKey,
    CtrlScroll,
    Tab,
    Arrows
};
```

enum Scroll

Scroll is an enum to indicate whether you want to scroll up or down, this enum is also written to enhance the readability of the methods that use scroll functions.

```
public enum Scroll
{
    Up,
    Down
};
```

DelayMilliseconds

To write a hardcoded delay in a test, we use the PlayBack.Wait() function, in which you have to pass an int indicating the amount of milliseconds the delay has to be. We created this variable to pass to the delay's so we can edit the wait-time of every delay in the entire project in one line.

```
            protected int DelayMilliseconds = 2000;
```

Process AppStartUp

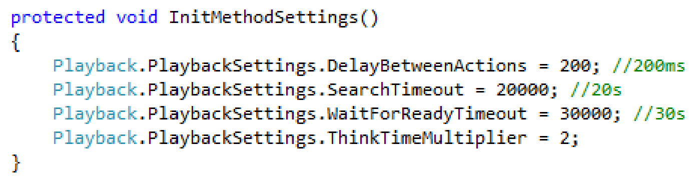
AppStartUp is a variable used to start the application before every test. AppStartUp has a property called StartInfo, in which you can pass the Url of the page you want to navigate to. Afterwards, you can use the function Start() of AppStartUp to start the application on the desired page.

```
            protected Process AppStartUp = new Process();
```

InitMethodSettings

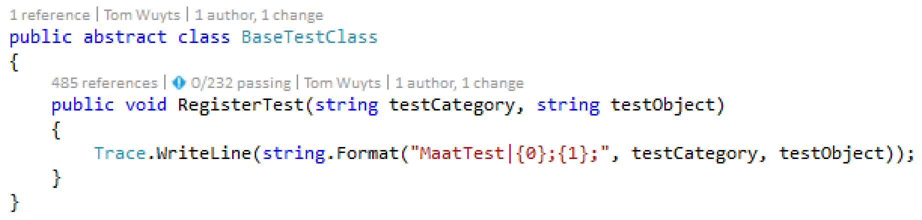
This is a method we use in every TestInitialize-function, to set some parameters regarding the execution of the test.

- DelayBetweenActions: the time the test normally waits between every action it has to execute (ex. between 2 clicks)
- SearchTimeout: the maximum time the tests dedicates on searching a control on the UI, if exceeded, the test fails and the error indicates the control cannot be found
- WaitForReadyTimeout: the maximum time the test dedicates on the total execution of the test, if exceeded, the test will fail no matter what.
- ThinkTimeMultiplier: This multiplies all other delay-values of the playbacksettings to its value



```
protected void InitMethodSettings()
{
    Playback.PlaybackSettings.DelayBetweenActions = 200; //200ms
    Playback.PlaybackSettings.SearchTimeout = 20000; //20s
    Playback.PlaybackSettings.WaitForReadyTimeout = 30000; //30s
    Playback.PlaybackSettings.ThinkTimeMultiplier = 2;
}
```

RegisterTest



```
1 reference | Tom Wuyts | 1 author, 1 change
public abstract class BaseTestClass
{
    485 references | 0/232 passing | Tom Wuyts | 1 author, 1 change
    public void RegisterTest(string testCategory, string testObject)
    {
        Trace.WriteLine(string.Format("MaatTest|{0};{1};", testCategory, testObject));
    }
}
```

For the explanation of this function, refer to “Automation of Coded UI Tests and its results”.

GetChildFromList

This is a function writen to get one specific child from a list or control. We found that sometimes a list or control is already loaded before it’s children are loaded, so we had to build a manual delay that can detect when the children of a list or control are loaded. We then added a variable to already select the instance you want to get.

- list: the list inside which the control you want to get exists
- instance: the instance of the control you want to get inside the list

The function returns the control you asked for.

```
public XamlControl GetChildFromList(XamlControl list, int instance)
{
    list.WaitForControlExist();
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }
    XamlControl testCase = listItems[instance] as XamlControl;
    return testCase;
}
```

WaitForPageLoaded

This is a function that uses the ProgressBar in a delay to wait for the page to be fully loaded. When the progressbar disappears, the page is fully loaded.

```
public void WaitForPageLoaded()
{
    if (UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar.Exists)
        while (UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar.Top > 0)Thread.Sleep(100) ;
}
```

ChangeNameSearchProperty

```
/// <summary>
/// Changes the searchproperty "name" to the desired value
/// </summary>
/// <param name="_control">control to change the searchproperty from</param>
/// <param name="_uID">Sesired "name"-searchproperty</param>
5 references | 0/5 passing | Peter Van de Putte, 8 days ago | 1 author, 1 change
public void ChangeNameSearchProperty(XamlControl _control, string _uID)
{
    _control.SearchProperties.Remove("Name");
    _control.SearchProperties.Add("Name", _uID);
}
```

TabToControl

This is a function that keeps tabbing until the control that is passed to it is highlighted.

```
public void TabToControl(XamlControl control)
{
    while (control.HasFocus == false)
    {
        Keyboard.SendKeys("{TAB}");
        Thread.Sleep(100);
    }
}
```

GoToControl

This function is used for a control in a list. It first gets the first control in the list in which the control you want to highlight exists. It then uses the “Up”-arrow key until the first control in the list is highlighted, and then it uses the “Down”-arrow key until the control you want to highlight is highlighted.

```
public void GoToControl(XamlControl control)
{
    UITestControl parent = control.GetParent();
    UITestControlCollection childs = parent.GetChildren();
    while (childs[0].HasFocus == false)
    {
        if (control.HasFocus == true)
            return;
        else
            Keyboard.SendKeys("{UP}");
        Thread.Sleep(100);
    }
    while (control.HasFocus == false)
    {
        Keyboard.SendKeys("{DOWN}");
        Thread.Sleep(100);
    }
}
```

ScrollToControl

This function accepts 3 parameters. The control to which it has to scroll, a “scrollControl”, which is a control from which it has to start scrolling, and a “Scroll”-enum, which can be Up or Down.

The function makes the mouse hover over the “scrollControl”, and then starts scrolling in the direction of “scroll”, until “control” is visible.

```
public void ScrollToControl(XamlControl control, XamlControl scrollControl, Scroll scroll)
{
    Mouse.Hover(GetMiddleOfControl(scrollControl));
    while (control.Left < 0)
    {
        if (scroll == Scroll.Up)
            Mouse.MoveScrollWheel(1);
        else
            Mouse.MoveScrollWheel(-1);
        Thread.Sleep(100);
    }
}
```

WriteAsUser

This function converts the string “msg” to char’s and then inputs every char as if a user was typing the keyboard.

```
public void WriteAsUser(string msg)
{
    char[] msgInChars = msg.ToCharArray();
    for (int i = 0; i < msgInChars.Length; i++)
    {
        Keyboard.SendKeys(msgInChars[i].ToString());
    }
}
```

GetMiddleOfControl

This function returns a Point, which is the middle of the “control” passed to it.

```
public Point GetMiddleOfControl(XamlControl control)
{
    int x = control.Left + (control.Width / 2);
    int y = control.Top + (control.Height / 2);

    Point middle = new Point(x, y);

    return middle;
}
```

GetExpectedValue

This function returns a string, which is the expected value to assert to in certain tests. You give 2 parameters to the function, the first is the control in which the value you expect exists, and the second is the instance of the value inside the control.

```
public string GetExpectedValue(XamlControl testCase, int instance)
{
    string expectedValue = GetChildFromList(testCase, instance).Name;
    return expectedValue;
}
```

RandomListInstance

This function accepts a list, and then returns a random int between 1 and the lists count -2 (so a random instance which is not the first and not the last instance of the list.

```
public int RandomListInstance(XamlControl list)
{
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }

    Random rdm = new Random();
    int random = rdm.Next(1, listItems.Count - 2);
    return random;
}
```

LastListInstance

The same as above, but this time the function returns the last instance inside the list as an int.

```
public int LastListInstance(XamlControl list)
{
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }

    int last = listItems.Count - 1;
    return last;
}
```

Content

ReadListItems

This function reads all the content of all the items inside a list, and compares it to an expected value string. If the string is present somewhere inside the list, the tests passes, if the string is not in the list, the test fails.

```
public void ReadListItems(XamlControl list, string expectedValue)
{
    XamlControl tile = GetChildFromList(list, 0);
    string text;
    for (int i = 0; i < list.GetChildren().Count; i++)
    {
        bool correct = false;
        tile = GetChildFromList(list, i);
        for (int j = 0; j < tile.GetChildren().Count; j++)
        {
            text = GetChildFromList(tile, j).Name;
            if (text.ToLower().Contains(expectedValue.ToLower())) correct = true;
        }
        if (correct == false) Assert.Fail("The search algorithm went wrong");
    }
}
```

Navigations

NavigateVariableControl

This is a function which tests the navigation for variable control’s. Variable controls are controls that exist inside a list but are dependent on the data existing inside the database. The amount of control’s inside the list is not fixed and neither is the data in the controls.

The function accepts a high amount of variables necessary for the testing:

- **list**: List inside which the case you want to test exists
- **assertControl**: Control you want to assert to after forward navigation
- **scrollStartControl:** Control from which you want to start the scroll function
- **backButton**: Button to navigate backwards after forward navigation
- **pageTitle**: Title of the page from which you are navigating
- **expectedValue**: Value you expect in the title after navigating forwards
- **instance**: instance of the ListItem you want to use inside the list
- **action**: Execution method (Click/Enter/Space)

```
public void NavigateVariableControl(XamlControl scrollStartControl, XamlContro
{

    string pageTitle = pageTitleControl.Name;
    XamlControl testCase = GetChildFromList(list, instance);
        if (forwardAction == Action.Click)
    {
        ScrollToControl(testCase, scrollStartControl, Scroll.Down);
            UseControl(testCase, Action.Click);
    }
        else
    {
            ListItemKey(list, instance, forwardAction);
    }

    string value = assertControl.Name;
    while (value.Contains(pageTitle) || value.Equals(""))
    {
        value = assertControl.Name;
        Thread.Sleep(100);
    }
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(expectedValue, value);

    UseControl(backButton, backwardAction);

    string backValue = assertControl.Name;
    while (backValue.Contains(value) || backValue.Equals(""))
    {
        backValue = assertControl.Name;
        Thread.Sleep(100);
    }
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(pageTitle, backValue);
}
```

NavigateHyperLink

This function tests the navigation of a hyperlink-control. The parameters you give to it are the hyperlink, the title-control of the pages and the action you want to use (click/enter/space).

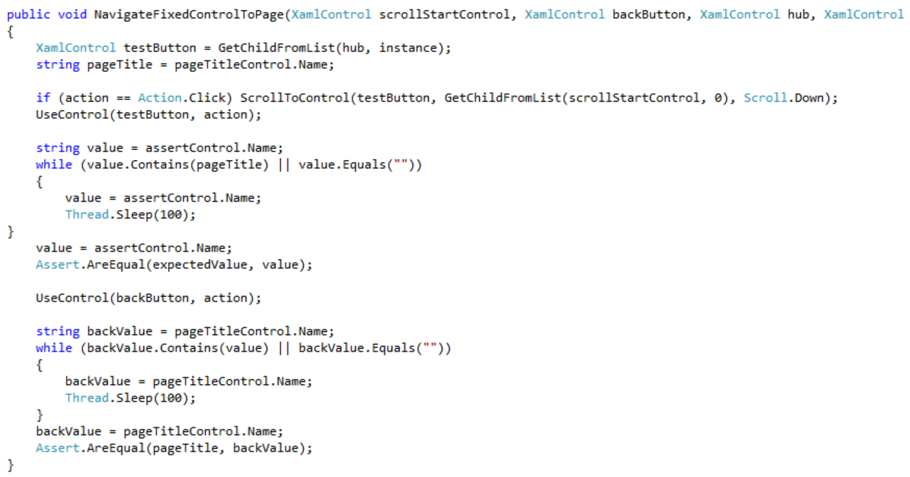
```
public void NavigateHyperlink(XamlHyperlink _hyperLink, XamlControl _titleControl, Action _action)
{
    XamlControl _backButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIItemBackButton;
    string _startValue = _titleControl.Name;
    string _trialLinkName = _hyperLink.Name;
    UseControl(_hyperLink, _action);
    while (_titleControl.Name.Contains(_startValue) || _titleControl.Name.Equals("")) ;

    if (_trialLinkName != _titleControl.Name) Assert.Fail("The forward navigation went wrong!");

    UseControl(_backButton, _action);

    while (_titleControl.Name.Contains(_trialLinkName) || _titleControl.Name.Equals("")) ;

    if (_startValue != _titleControl.Name) Assert.Fail("The backward navigation went wrong!");

}
```

NavigateFixedControlToPage

This function tests the navigation of a fixed control to a unique page. Fixed controls are controls that always exist on a page and in this case, each control navigates to a unique page in relation to the other controls in the same collection.

These are the variables you need for the function:

- **hub**: Hub in which the ListItem you want to click exists
- **backButton**: Backbutton to navigate back to the page you are testing
- **assertControl**: The titlecontrol of the page you are navigating to
- **pageTitleControl**: The titlecontrol of the page from which you are navigating
- **scrollStartControl**: Control from which you want to start scrolling
- **expectedValue**: Value to expect in the title after navigating forwards
- **instance**: Instance of the button you want to click inside the hub
- **action**: Execution method (click/enter/space)



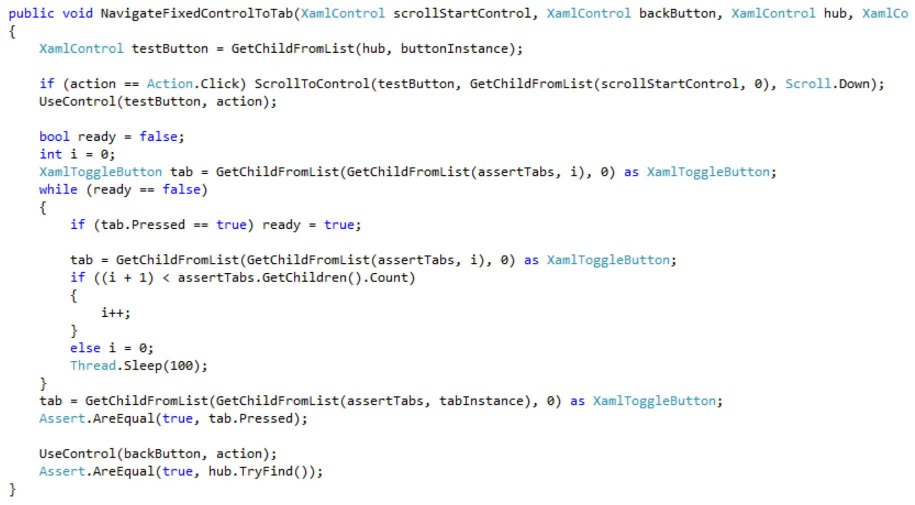
```
public void NavigateFixedControlToPage(XamlControl scrollStartControl, XamlControl backButton, XamlControl hub, XamlControl
{
    XamlControl testButton = GetChildFromList(hub, instance);
    string pageTitle = pageTitleControl.Name;

    if (action == Action.Click) ScrollToControl(testButton, GetChildFromList(scrollStartControl, 0), Scroll.Down);
    UseControl(testButton, action);

    string value = assertControl.Name;
    while (value.Contains(pageTitle) || value.Equals(""))
    {
        value = assertControl.Name;
        Thread.Sleep(100);
}
    value = assertControl.Name;
    Assert.AreEqual(expectedValue, value);

    UseControl(backButton, action);

    string backValue = pageTitleControl.Name;
    while (backValue.Contains(value) || backValue.Equals(""))
    {
        backValue = pageTitleControl.Name;
        Thread.Sleep(100);
    }
    backValue = pageTitleControl.Name;
    Assert.AreEqual(pageTitle, backValue);
}
```

NavigateFixedControlToTab

Performs the same test as NavigateFixedControlToPage, but the difference is that these controls, in their own collection, all navigate to the same page but trigger a different state of that page. In this case, they all trigger a different tab to open on the same page.

Variables:

- **hub**: Hub in which the Control you want to click exists
- **backButton**: Backbutton to navigate back to the page you are testing
- **assertTabs**: The tab-bar in which you can go to the different tabs, on the page you are navigating to
- **scrollStartControl**: Control from which you want to start scrolling
- **buttonInstance**: Instance of the button you want to click inside the hub
- **tabInstance**: Instance of the tab inside "assertTabs" the button should navigate to
- **action**: Execution method (click/enter/space)

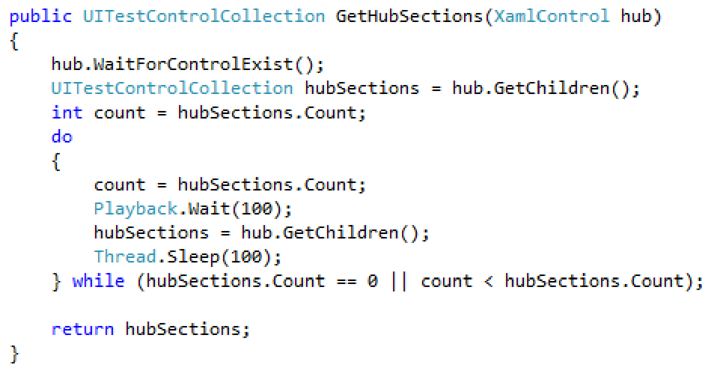


```
public void NavigateFixedControlToTab(XamlControl scrollStartControl, XamlControl backButton, XamlControl hub, XamlCo
{
    XamlControl testButton = GetChildFromList(hub, buttonInstance);

    if (action == Action.Click) ScrollToControl(testButton, GetChildFromList(scrollStartControl, 0), Scroll.Down);
    UseControl(testButton, action);

    bool ready = false;
    int i = 0;
    XamlToggleButton tab = GetChildFromList(GetChildFromList(assertTabs, i), 0) as XamlToggleButton;
    while (ready == false)
    {
        if (tab.Pressed == true) ready = true;

        tab = GetChildFromList(GetChildFromList(assertTabs, i), 0) as XamlToggleButton;
        if ((i + 1) < assertTabs.GetChildren().Count)
        {
            i++;
        }
        else i = 0;
        Thread.Sleep(100);
    }
    tab = GetChildFromList(GetChildFromList(assertTabs, tabInstance), 0) as XamlToggleButton;
    Assert.AreEqual(true, tab.Pressed);

    UseControl(backButton, action);
    Assert.AreEqual(true, hub.TryFind());
}
```

States

GetHubSections

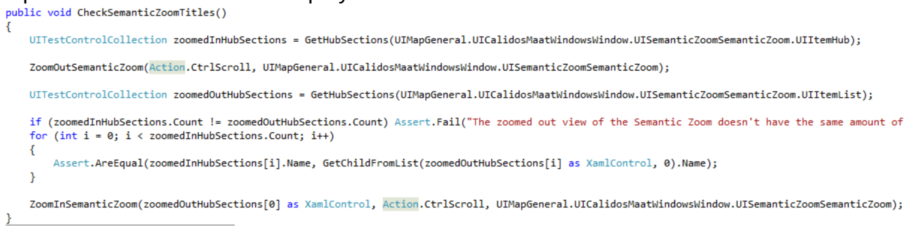
This is a function that returns all the hubsections that exist inside a hub as a Testcontrolcollection.



```
public UITestControlCollection GetHubSections(XamlControl hub)
{
    hub.WaitForControlExist();
    UITestControlCollection hubSections = hub.GetChildren();
    int count = hubSections.Count;
    do
    {
        count = hubSections.Count;
        Playback.Wait(100);
        hubSections = hub.GetChildren();
        Thread.Sleep(100);
    } while (hubSections.Count == 0 || count < hubSections.Count);

    return hubSections;
}
```

CheckSemanticZoomTitles

This function zooms out the semantic zoom and checks if the titles displayed in zoomed out view represent the same titles as displayed in the zoomed in view.



```
public void CheckSemanticZoomTitles()
{
    UITestControlCollection zoomedInHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemHub);

    ZoomOutSemanticZoom(Action.CtrlScroll, UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom);

    UITestControlCollection zoomedOutHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemList);

    if (zoomedInHubSections.Count != zoomedOutHubSections.Count) Assert.Fail("The zoomed out view of the Semantic Zoom doesn't have the same amount of
    for (int i = 0; i < zoomedInHubSections.Count; i++)
    {
        Assert.AreEqual(zoomedInHubSections[i].Name, GetChildFromList(zoomedOutHubSections[i] as XamlControl, 0).Name);
    }

    ZoomInSemanticZoom(zoomedOutHubSections[0] as XamlControl, Action.CtrlScroll, UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom);
}
```

ZoomOutSemanticZoom

To go from zoomed-in to zoomed-out state, you can use this function. The action you give to the function is the action you want to zoom out with, this can be Ctrl+Scroll, Ctrl+key,...

```
public void ZoomOutSemanticZoom(Action action, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (semanticZoom.ZoomedIn)
    {
        Playback.Wait(2000);
        if (action == Action.CtrlScroll)
        {
            Mouse.Hover(GetMiddleOfControl(semanticZoom));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Mouse.MoveScrollWheel(-1);
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.CtrlKey)
        {
            Mouse.Click(new Point(semanticZoom.Left, semanticZoom.Top));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Keyboard.SendKeys("{SUBTRACT}");
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

    }
    if (semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed in after trying to zoom out!");
}
```

The function below is the same, but for the click-execution

```
public void ZoomOutSemanticZoom(XamlControl control, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (semanticZoom.ZoomedIn)
    {
        Mouse.Click(GetMiddleOfControl(control));
        Assert.AreEqual(false, semanticZoom.ZoomedIn);
    }

    if (semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed in after trying to zoom out!");
}
```

ZoomInSemanticZoom

This function does the same as the above function, except this one goes from the zoomed-out state back to the zoomed-in state instead of the other way around.

```
public void ZoomInSemanticZoom(XamlControl control, Action action, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (!semanticZoom.ZoomedIn)
    {
        if (action == Action.CtrlScroll)
        {
            Mouse.Hover(GetMiddleOfControl(control));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Mouse.MoveScrollWheel(1);
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.CtrlKey)
        {
            GoToControl(control);
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Keyboard.SendKeys("{ADD}");
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.Click)
        {
            Mouse.Click(GetMiddleOfControl(control));
        }
    }

    if (!semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed out after trying to zoom in!");

}
```

AssertSemanticZoom

This function checks if the semantic zoom zoom functions work correctly. There are 2 overloads of this function. One overload for the key-executions and one for the click-execution.
Each ask for the hubsection and the hubsection-name control.

```
public void AssertSemanticZoom(XamlControl hubsection, XamlControl hubsectionName, Action action)
{
    XamlSemanticZoom semanticZoom = UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom;
    semanticZoom.WaitForControlExist();
    UITestControlCollection zoomedInHubSections;
    UITestControlCollection zoomedOutHubSections;

    if (semanticZoom.ZoomedIn) zoomedInHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemHub);
    else zoomedOutHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemList);

    ZoomOutSemanticZoom(action, semanticZoom);

    ZoomInSemanticZoom(hubsectionName, action, semanticZoom);
    hubsection.WaitForControlExist();
    Assert.IsTrue(hubsection.Left >= semanticZoom.Left);
}
```

```
public void AssertSemanticZoom(XamlControl hubsection, XamlControl hubsectionName, XamlControl control)
{
    XamlSemanticZoom semanticZoom = UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom;
    ZoomOutSemanticZoom(control, semanticZoom);
    ZoomInSemanticZoom(hubsectionName, Action.Click, semanticZoom);
    hubsection.WaitForControlExist();
    Assert.IsTrue(hubsection.Left >= semanticZoom.Left);
}
```

Functionality

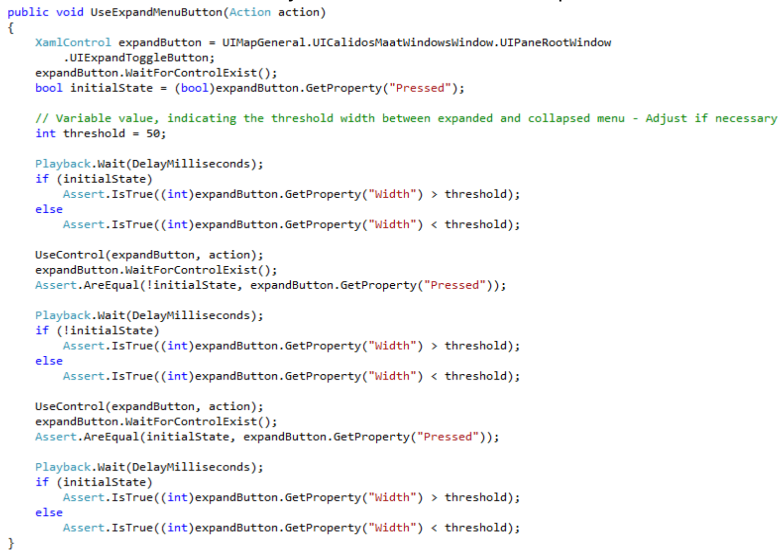
UseHomeButton

This function uses the HomeButton with the action you give to it. Afterwards, it checks if the correct action happened after using the button.

```
public void UseHomeButton(Action action)
{
    XamlControl homeButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow.UIHomeToggleButton;
    XamlText homePageText = UIMapGeneral.UICalidosMaatWindowsWindow.UIHomeTitleText;

    homeButton.TryFind();
    bool initialState = (bool)homeButton.GetProperty("Pressed");

    UseControl(homeButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(!initialState, homeButton.GetProperty("Pressed"));
    Assert.AreEqual(true, homePageText.GetProperty("Exists"));

    UseControl(homeButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(initialState, homeButton.GetProperty("Pressed"));
}
```

UseExpandMenuButton

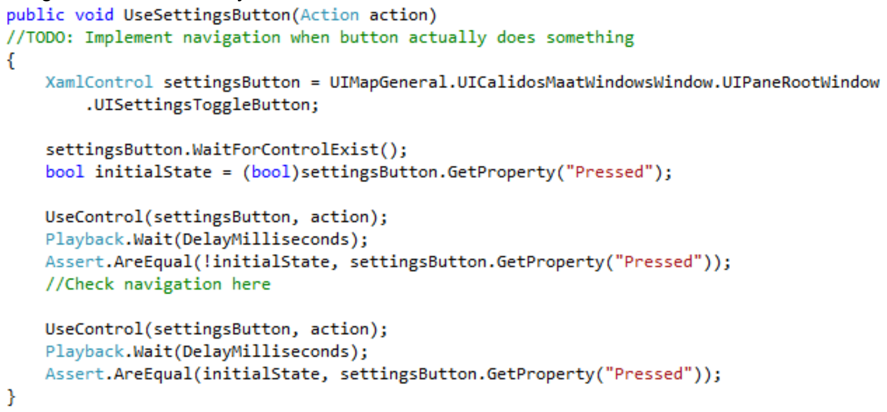
This function tests the functionality of the “expand”-button, below the home-button. You can choose the action with which you want to execute the “expand” action.



```
public void UseExpandMenuButton(Action action)
{
    XamlControl expandButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow
        .UIExpandToggleButton;
    expandButton.WaitForControlExist();
    bool initialState = (bool)expandButton.GetProperty("Pressed");

    // Variable value, indicating the threshold width between expanded and collapsed menu - Adjust if necessary
    int threshold = 50;

    Playback.Wait(DelayMilliseconds);
    if (initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);

    UseControl(expandButton, action);
    expandButton.WaitForControlExist();
    Assert.AreEqual(!initialState, expandButton.GetProperty("Pressed"));

    Playback.Wait(DelayMilliseconds);
    if (!initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);

    UseControl(expandButton, action);
    expandButton.WaitForControlExist();
    Assert.AreEqual(initialState, expandButton.GetProperty("Pressed"));

    Playback.Wait(DelayMilliseconds);
    if (initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);
}
```

UseSettingsButton

This function tests the settings-button. You can choose the action to execute with, note that the navigation doesn’t exist yet.



```
public void UseSettingsButton(Action action)
//TODO: Implement navigation when button actually does something
{
    XamlControl settingsButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow
        .UISettingsToggleButton;

    settingsButton.WaitForControlExist();
    bool initialState = (bool)settingsButton.GetProperty("Pressed");

    UseControl(settingsButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(!initialState, settingsButton.GetProperty("Pressed"));
    //Check navigation here

    UseControl(settingsButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(initialState, settingsButton.GetProperty("Pressed"));
}
```

UseControl

This is a function designed to use a control with the action given to it. If the action is “click”, the control will be clicked. If the action is “enter”, the control will be entered,...

The reason we wrote this function is to implement the “WaitForControlExist()” function so we do not have to write this line of code every time we want to use a control.

```
public void UseControl(XamlControl control, Action action)
{
    if (action == Action.Enter || action == Action.Space)
    {
        TabToControl(control);
        Keyboard.SendKeys("{" + action.ToString().ToUpper() + "}");
    }
    if (action == Action.Click)
    {
        control.WaitForControlExist();
        Mouse.Click(GetMiddleOfControl(control));
    }
}
```

CheckReactionSpeed

This function returns a TimeSpan, which contains the amount of time it took for the page you are testing to fully load.

```
public TimeSpan CheckReactionSpeed()
{
    XamlProgressBar _progressBar = UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar;
    Stopwatch _watch = new Stopwatch();
    _watch.Start();
    while (_progressBar.Top > 0) ;
    _watch.Stop();

    return _watch.Elapsed;
}
```

ComboBoxSelectItem

This function is designed to select an Item from a combobox. You give the Item you want to select to it, and the function opens the combobox and clicks that Item.

```
public void ComboBoxSelectItem(XamlControl control, int number)
{
    control.WaitForControlExist();
    Mouse.Click(GetMiddleOfControl(control.GetParent() as XamlControl));
    UseControl(control, Action.Click);
}
```

ListItemKey

This is a method with which you can use a ListItem inside a list, using the keyboard. The parameters are:

- **list**: the list inside which your ListItem exists.
- **instance**: the instance of the your ListItem inside the list
- **action**: the execution method

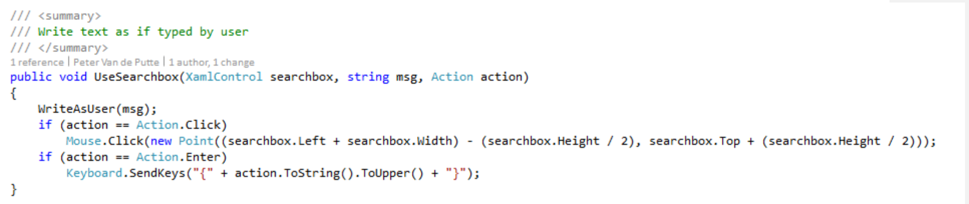
```
public void ListItemKey(XamlControl list, int instance, Action action)
{
    XamlControl testCase = this.GetChildFromList(list, instance);

    if (instance == 0)
    {
        TabToControl(testCase);
    }

    else if (instance != 0)
    {
        while (list.GetChildren()[0].HasFocus == false)
        {
            Keyboard.SendKeys("{TAB}");
            Thread.Sleep(100);
        }
        GoToControl(testCase);
    }
    Keyboard.SendKeys("{" + action.ToString() + "}");
}
```

ClearEdit

This method is designed to clear an edit-control of all it’s content, before putting new content in it.

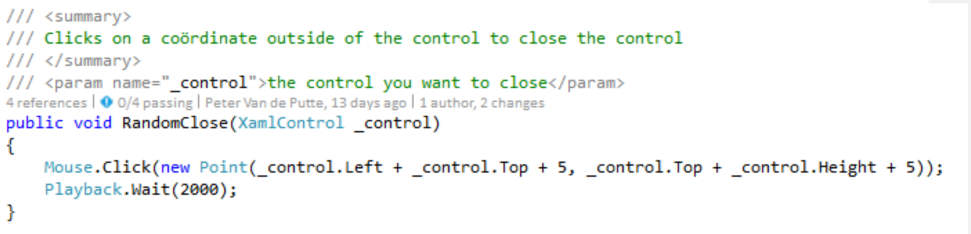
```
public void ClearEdit(XamlEdit edit)
{
    int i = 0;
    while ((edit.Text != "") && (i < 20))
    {
        i++;
        Keyboard.SendKeys("{Back}");
        Thread.Sleep(100);
    }

}
```

UseSearchBox



```
/// <summary>
/// Write text as if typed by user
/// </summary>
1 reference | Peter Van de Putte | 1 author, 1 change
public void UseSearchbox(XamlControl searchbox, string msg, Action action)
{
    WriteAsUser(msg);
    if (action == Action.Click)
        Mouse.Click(new Point((searchbox.Left + searchbox.Width) - (searchbox.Height / 2), searchbox.Top + (searchbox.Height / 2)));
    if (action == Action.Enter)
        Keyboard.SendKeys("{" + action.ToString().ToUpper() + "}");
}
```

RandomClose



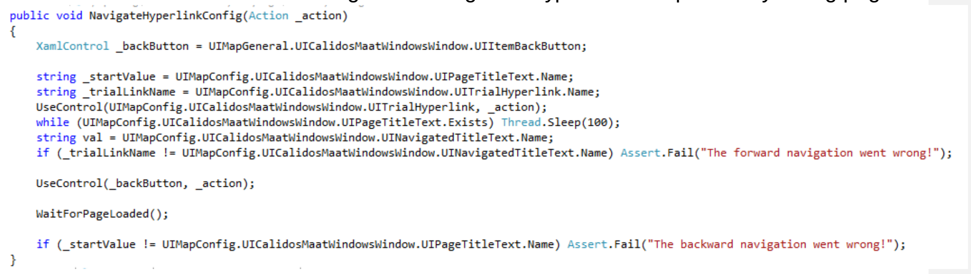
```
/// <summary>
/// Clicks on a coördinate outside of the control to close the control
/// </summary>
/// <param name="_control">the control you want to close</param>
4 references | 0/4 passing | Peter Van de Putte, 13 days ago | 1 author, 2 changes
public void RandomClose(XamlControl _control)
{
    Mouse.Click(new Point(_control.Left + _control.Top + 5, _control.Top + _control.Height + 5));
    Playback.Wait(2000);
}
```

Config

For all the config-pages, we made a separate BaseClassCodedUI-partial class, because there are many tests that can be copy-pasted inside these pages.

NavigateHyperlinkConfig

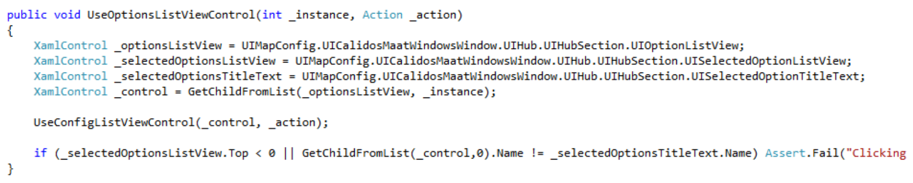
This is a function that tests the navigation through the hyperlink on top of every config page.



```
public void NavigateHyperlinkConfig(Action _action)
{
    XamlControl _backButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIItemBackButton;

    string _startValue = UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Name;
    string _trialLinkName = UIMapConfig.UICalidosMaatWindowsWindow.UITrialHyperlink.Name;
    UseControl(UIMapConfig.UICalidosMaatWindowsWindow.UITrialHyperlink, _action);
    while (UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Exists) Thread.Sleep(100);
    string val = UIMapConfig.UICalidosMaatWindowsWindow.UINavigatedTitleText.Name;
    if (_trialLinkName != UIMapConfig.UICalidosMaatWindowsWindow.UINavigatedTitleText.Name) Assert.Fail("The forward navigation went wrong!");

    UseControl(_backButton, _action);

    WaitForPageLoaded();

    if (_startValue != UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Name) Assert.Fail("The backward navigation went wrong!");
}
```

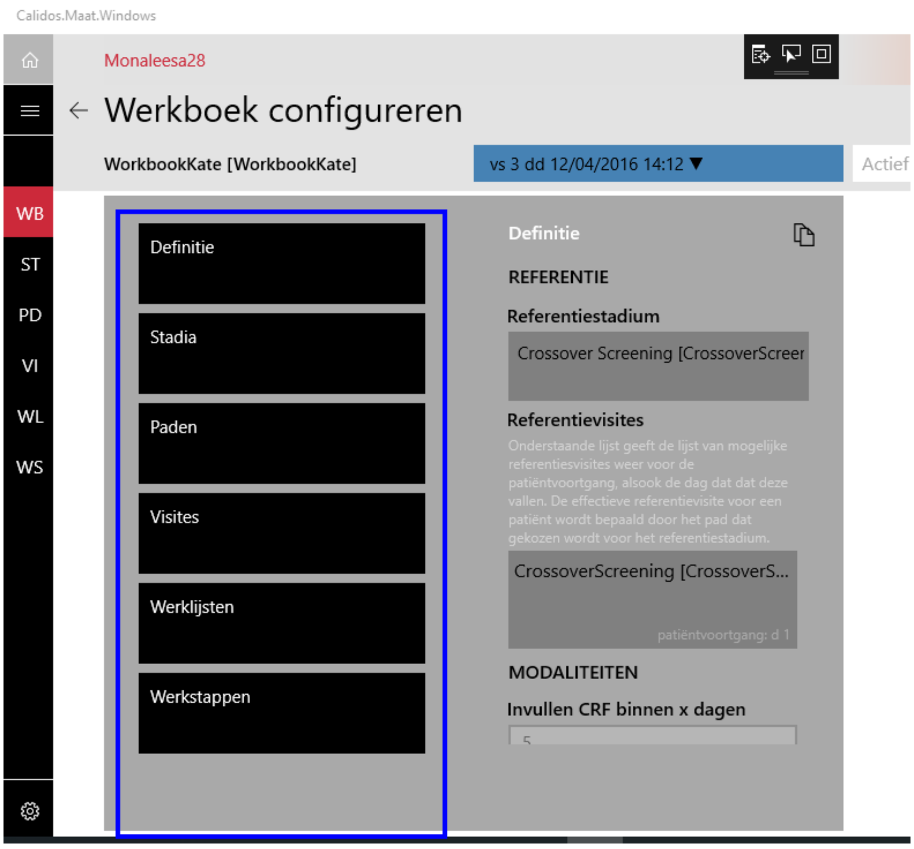
UseOptionsListViewControl

This function is designed to test if clicking one of the options on the left side of the screen results in the correct list appearing to the right of it. The instance is the number of the option in the view (first, second, third,...)



```
public void UseOptionsListViewControl(int _instance, Action _action)
{
    XamlControl _optionsListView = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UIOptionListView;
    XamlControl _selectedOptionsListView = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UISelectedOptionListView;
    XamlControl _selectedOptionsTitleText = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UISelectedOptionTitleText;
    XamlControl _control = GetChildFromList(_optionsListView, _instance);

    UseConfigListViewControl(_control, _action);

    if (_selectedOptionsListView.Top < 0 || GetChildFromList(_control,0).Name != _selectedOptionsTitleText.Name) Assert.Fail("Clicking
}
```

With options, we mean the following:



Onderstaande lijst geeft de lijst van mogelijke referentiesvisites weer voor de patiëntvoortgang, alsook de dag dat dat deze vallen. De effectieve referentievisite voor een patiënt wordt bepaald door het pad dat gekozen wordt voor het referentiestadium.
CrossoverScreening [CrossoverS...
patiëntvoortgang: d 1
MODALITEITEN
Invullen CRF binnen x dagen
5

ChooseWorkbookConfigWorkVersion

A function designed to choose the latest work version to test on, because otherwise some options will be disabled. Note that this is a hardcoded function, and if something should change to the version-selection button, the function will not work anymore. The reason this function is hardcoded is that we cannot access the version-options with the coded UI test builder.

```
public void ChooseWorkbookConfigWorkVersion()
{
    UseControl(UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UISelectVersionButton, Action.Click);
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Tab}");
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Up}");
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Enter}");
}
```

UseConfigListViewControl

This function navigates to one of the config-pages. You just have to give the control and the execution-method to it, the function does the rest.

```
public void UseConfigListViewControl(XamlControl _control, Action _action)
{
    if (_action == Action.Tab)
    {
        _control.WaitForControlExist();
        TabToControl(_control);
    }
    if (_action == Action.Arrows)
    {
        _control.WaitForControlExist();
        UITestControl _parent = _control.GetParent();
        UseControl(_parent.GetChildren()[0] as XamlControl, Action.Click);
        GoToControl(_control);
    }
    else if (_action == Action.Click)
    {
        _control.WaitForControlExist();
        Mouse.Click(GetMiddleOfControl(_control));
    }
    else Assert.Fail(_action.ToString() + "is not possible on this control!");
}
```

Automation of Coded UI Tests and its results

Automation: Adding Coded UI Tests to the build street

After looking into a lott of options the main conclusion here is that, for now, adding the coded ui tests to the build street is not possible. the main reasons for this are:

- TFS 2013 cannot run a virtual windows 10 environment to deploy the application to
- TFS 2013 only supports windows 8 environments and lower to use with test agents

While TFS is not upgraded to a newer version, the testing automation will only go as far as to manually start the testrun, then leaving the computer to AFK-test for that time. In the end the test run will produce a TRX file that contains all details concerning failed, passed, aborted or not executed tests.

However, should there be an upgrade to the TFS 2015 or cloud version in the near future, this link might provide a good start to setting up the automation: Specify the steps of your build on VSTS & TFS

Result Management Tools: Visualizing results

Coded UI Tests can help out with a lott of normally manual testing on your application. It's an easy - and timesaving - way of discovering problems, bugs or other unwanted things within your app. But while automated Coded UI Tests do all that work for you, you still don't have a full view of the status of testing the many facets your application should be tested upon. Therefore, we designed the Result Management Tools. These handy tools bring all the result data together in one HTML file, so to give the developer/tester a better view of the testing progression.

There are 2 Tools (or 3 if you count the HTML file too) that make all this possible, in combination with 3 XML files. A detailed explanation can be found in the next paragraphs.

Tool: TRX 2 XML Parser

As easy as it says. This tool will provide a fast way to reduce the redundant information the TRX file contains, and putt all useful information into the newly generated XML file. As said in previous paragraphs or chapters, the TRX file contains test run data. This is a lott of data - believe it or not - but not all of it is useful for the developer/tester.

RegisterTest

The first thing you have to know about the TRX file is that, for this tool to work, we need to add another thing to every coded ui test that has been created. A reference as to what it is actually testing. To indicate this we designed the RegisterTest method.

The RegisterTest method - defined in BaseTestClass - **needs to be called in every test method** as the **first line** of that test method. This is done **so to automatically check if a test is completed**, failed or not run at all. The method will write more details into the file so to know exactly where the tests are coming from (page/control and paradigm). To specify the page/control and paradigm we use GUID's (Global Universal Identifiers).

Running tests

A test run can be started in the CMD for developers from visual studio, by using the .dll of the Calidos.Maat.CodedUITests project. This can be done when in the bin/debug (or in this case DevAzure) folder and calling following line:
"MSTest /testcontainer: Calidos.Maat.CodedUITests.dll"

● MSTest	The program that will be used to start the testing
● testcontainer	The file that is used to search for test methods that can be run

> *Note:* *To change the folder to which the command prompt window opens by default, on windows, choose **Start**, point to **Microsoft Visual Studio 2015**, point to **Visual Studio Tools**, right-click **Developer Command Prompt**, and then choose **Properties**. In the **Developer Command Prompt Properties** dialog box, you can change the path to the default folder in the **Start in** box.*

Another option to start the test run is to just open your project in Visual Studio, select **Test**, then choose **Run**, and eventually click **All Tests**.

When tests are done running the program publishes a logfile in the bin/DevAzure folder under the new folder named "TestResults". More information about the TRX file and where it's deployed, or where you could deploy it to can be found in following link: How to: Deploy Files for Tests

Running the TRX 2 XML Tool

To run the TRX 2 XML application, it was necessary at first to add command line arguments under the properties tab. This will reference the target and input file paths. This will be automated in the near future.

"Command line arguments" during testing at this time:

```
-TestFilePath="C:\Dev\CTO\Src\Dev\D.Jasper.0\Client\Calidos.Maat\Calidos.Maat.CodedUITests\bin\DevAzure\TestResults\Jasper_VIRAPTORUS 2016-04-19 12_37_25.trx" -Output="C:\Temp\\"
```

When using the application it will recover the file (in this test it will be from the gray colored file path). It will execute all code and parse it into an XML file and put it in the designated folder (in this test it will be in the red file path)

Result XML

When we have the complete TRX file we can parse it with our TRX 2 XML tool that will handle the parsing process from the TRX to an XML file - known as the Result XML - with the use of a streamwriter and the System.Xml.Linq class.

The tool will search for the UnitTestResult tag within the TRX and pick up the necessary data to write into the XML. In this case, "outcome" and "testName" Then searches for the DebugTrace tag to find the data inserted by the RegisterTest method. Every value that contains the "MaatTest|" indicator will be obtained and also added to the XML file. The Result XML will have following syntax:

<Test TestId="Test Name" CategoryId ="Category GUID" ObjectId="Object GUID" ResultLabel="Outcome">

- TestId
    - The test name (test method name) that was given in the code of the test
    - CategoryId
    - The ID of the category, written as a GUID
    - ObjectId
    - The ID of the object, written as a GUID
- ResultLabel
    - Contains one of 4 different possible statuses:

Passed	The test passed
Failed	The test failed
Aborted	The test was being executed but got manually aborted
Not Executed	The test has not been executed at all

Tool: TestResultParser

The TestResultParser, in short, accepts 3 XML input files and creates an HTML file as its output. The first XML file we already discussed. In this section we will discuss the other 2 XML files and the end product, the HTML file, as well as how this tool actually works.

HTML file: Result Table

To understand what kind of input we need or use, we have to start at our end product. The HTML file will give a visual representation of the progress considering test completion of your application. This is done through a biaxial table or in other words a matrix. In our coded ui testing there is an axis that represents paradigms and one that represents controls/screens. A cell in the matrix is the junction of each paradigm and control/screen, or in better words a cell represents a test case. A test case will always show its percentage passed or in some cases the letter that represents if it is To Do (T), Not To Do (N) or Unknown (-). Each cell also contains one of 5 different states of testing:

Passed	100% Completion of test case
Failed	< 100% Completion of test case
To Do	This test case does not have a test written for it
Not To Do	This test case does not need a test written for it
Unknown	This test case is not yet been analyzed

As for cells always showing their total percentage passed. This means they also contain the percentage passed of their children. Even tho sometimes it occurs that a parent cell also is a test case itself. In this case, The cell shows an addition of both the test case's own percentage passed and the percentage passed of its children

> *Note:* When fixing test cases, it's good practice to always start fixing the lowest descendant of the parent that indicates a 'Failed' status. This to make sure the parent is, or is not, the problem of the 'Failed' status.

Definition XML

The Definition XML is used to manually define the axes of the table. Remember when we talked about GUIDs? Well here's where we define them along with an appropriate name for each paradigm and screen/control. The syntax has a CategoryDefs and ObjectDefs tag that contain their Objects/Categories that are defined as follows:

```
<Category id="Category GUID" name="Paradigm" info="Description" level="">

<Object id="Object GUID" name="Screen / Control" info="Description" level="">
```

- CategoryId / ObjectId
  - The ID, written as a GUID
- Name
  - Paradigm or screen/control name
  - Require to be unique
- Level
  - The level object will be calculated and therefore doesn't need to be defined because it will be overwritten in any case.

Target XML

The Target XML manually defines a target for each test case. The syntax is as follows:

```
<Target TargetName="Description" CategoryId="Category GUID" ObjectId="Object GUID" TargetLabel="Label"/>
```

- TargetName
  - Descriptive name of the category and object (in readable text)
  - This, to help understand what is exactly there (GUID only is hard to read)
  - CategoryId
  - The ID of the category, written as a GUID
  - ObjectId
  - The ID of the object, written as a GUID
- TargetLabel
  - Contains one of 4 different possible statuses:

To Do	This test case needs a test written for it
Not To Do	This test case does not need a test written for it
Unknown	This test case is not yet been analyzed
Done	This test case is manually acknowledged to be 100% completed

XSD Files

XML Schema Definition in full, once again a name that says it all. Every XML has an XSD behind them, mainly for one simple reason. We use Xsd2Code++ to map all defined objects in XML to objects in Visual Studio.

Running the TestResultParser tool

The tool executes a number of methods that result into generating HTML for the HTML file. Let's have a short look at them

First we create our full matrix in memory using three methods:

- ProcessCategories
- ProcessObjects
- CreateCells

ProcessCategories

- Creates a list of all categories for the vertical axis
- Calculates the level of each category
- Uses the Definition XML to obtain the right properties

ProcessObjects

- Creates a list of all objects for the horizontal axis
- Calculates the level of each object
    - Uses the Definition XML to obtain the right properties

CreateCells

- Creates all the cells (a total of: Horizontal Axis * Vertical Axis)
- Maps the parents and children correctly so to create totals

When the empty matrix is generated it gets time to populate it. This is where our Target and Result XML files come in.

ProcessTargetData

- Maps Target XML data to the correct cells using the GUIDs and the TargetLabels

ProcessResultData

- Maps Result XML data to the correct cells using the GUIDs and the ResultLabels

When all the virtual generation of the table is completed we can start writing the HTML code for it. The HTML is built in following order:

● ProduceHtml	
○ CreateHtmlHead	Create metadata
○ CreateHtmlBody	Create Table, Legend, Failed list
■ CreateTestResultDiv	Create Table wrapper
● CreateTestResultTableHead	Create Table Head (Objects/Screens)
● CreateTestResultTableBody	Create Table Body (Cells / Categories)
■ CreateLegendDiv	Create Legend in wrapper
■ CreateFailedDiv	Create Failed Test List in wrapper