ArtComputer

GOOD PRACTICE

Automated Test with the Robot

Good practices for automated tests

Foreword

Before writing your very first scenario, it's important to have a clear view of the structure of the tests.

You need to define what will be the policy for the naming convention, the dictionary, the dataset, the required subprojects.

A wrong choice at this level will impact all your tests until the end of the project (it will be very difficult to go back and update a wrong implementation as these choices are the foundations of your automated tests).

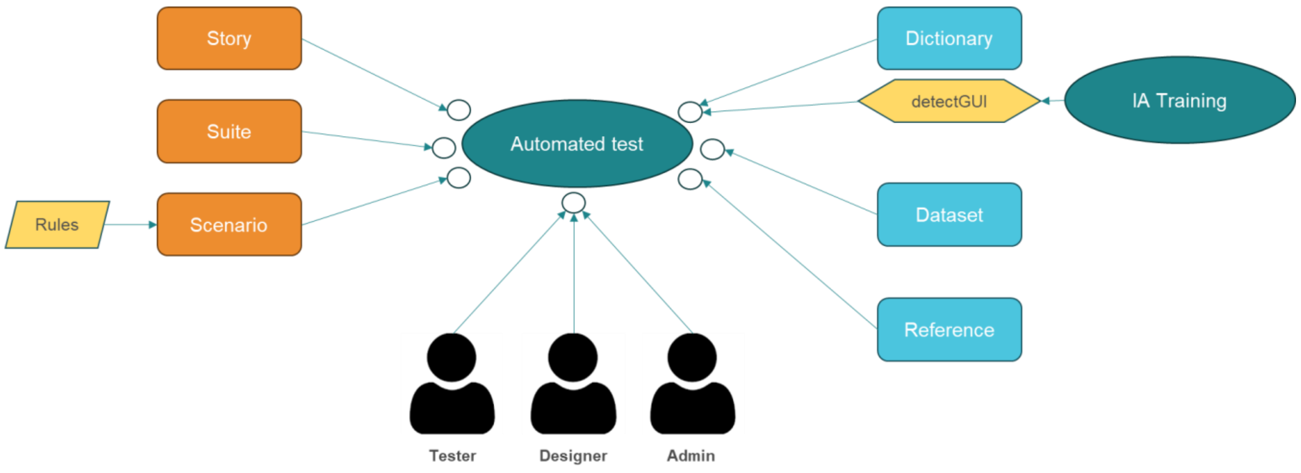
Unfortunately, there is no specific implementation, it depends of the complexity of your project, the size of your testing team, your skill…
In this document, I will try to give you some advices.
Maybe some advices will not be suitable for you… it's up to you to decide your point of view!

A strong and coherent naming convention will help you to design your scenario.
The accepted formats are:

- Dictionary starts always with @
- Dataset starts always with #
- Naming convention can be all in uppercase or lowercase: @URL_ACCEPTANCE
- Naming convention can be a mix @URL_Acceptance
- Naming convention can be with spaces or not: @URL_Environment Acceptance

Architecture of a test

To understand the objectives of this document, you need to understand the basic concepts of the automated tests.



Automated tests are defined with a scenario including all the steps to perform with the application.
Optionally, you can group scenarios into a suite.
Finally, you can group the scenarios/suites into a story.

The story will be the playground for the testers.
The Designer will create the scenarios, the suites and the stories.
The Admin will manage the people, the projects and the subprojects.

To interact with the scenario, you will need a dictionary to search for the definition of an element.
In some cases, you can use the function detectGUI that will allow you to interact with an element without the support of the dictionary. However, before using this function, you will need to train the Robot to recognize the different elements (like a list, a field, a menu…)
To execute a scenario, you need to provide the Robot with dataset.
Finally, the reference will be used to exchange data between the different scenario (E.g.: a new reference)

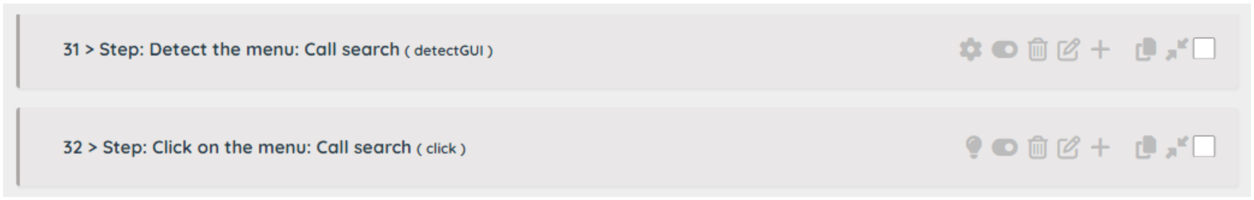
Comments in the tests

The comments are the most important thing in the automated tests.
Comments are present in all the modules from the dictionary, the dataset, the rules, the tests…
I suggest that you take great care in the definition of your comment!

In the rules and in the tests, you have the possibility to define (just for information) if a comment is appropriate for the Business or in contrary if the comment is a technical step (for instance a technical flag).

- A business comment is represented by a lightbulb
- A Design (technical) comment is represented by a gear

Example of type of comment:



31 > Step: Detect the menu: Call search (detectGUI)

32 > Step: Click on the menu: Call search (click)

In this example, the detection of the menu is not important for the Business (it's a Designer comment)
The click on the menu is very important for the Business to understand what to do to test the screen.

Using this information can be very useful to prepare the scenarios for the UAT or if you want to validate your automated scenarios with the Business Analyst.

In the Tests and in the Rules, you can directly filter the Business comment on the screen.
If you need to export the tests/rules, the field commentType will support the values:

- 0: Design (technical) comment
- 1: Business comment

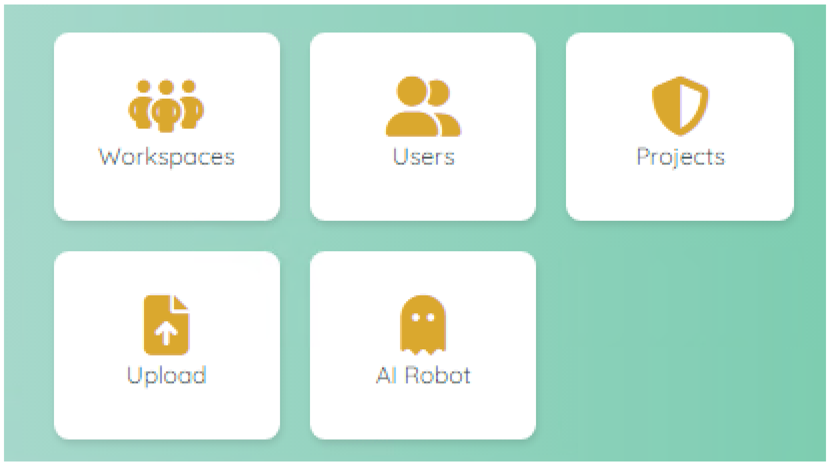
The Roles

There are 3 roles defined in the application.

<u>**Administrator**</u>:

The Administrator manages:

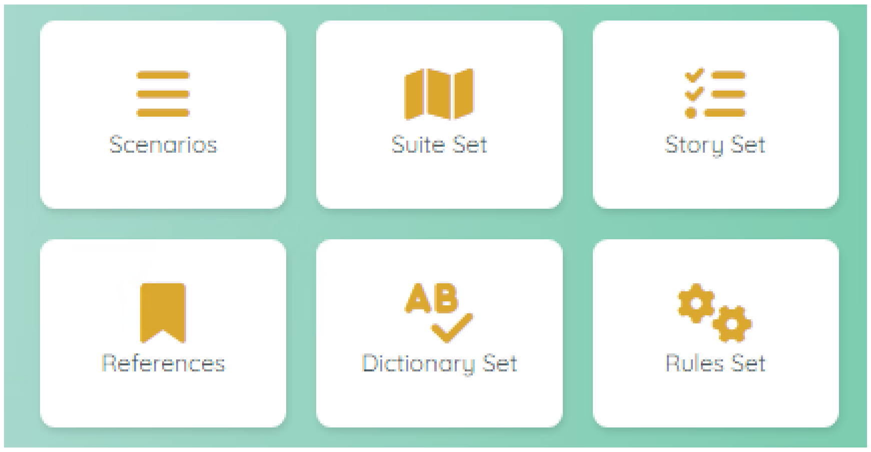
- The workspace (of the customer)
- The users
- The projects/subproject and the assignment of a user into a project
- The documents available for the upload function
- The AI Robot used to train the Robot to recognize element on a screen



Designer:

The Designer is responsible for:

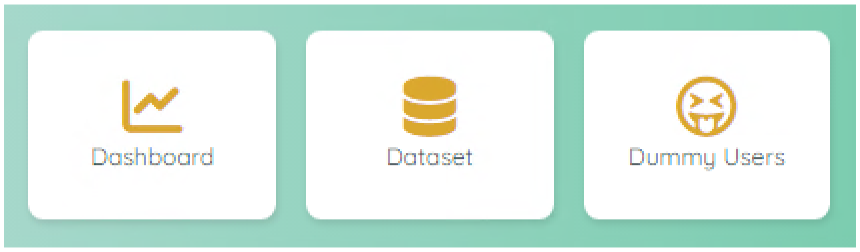
- The design of the Scenarios
- The design of the Suites
- The design of the Stories
- The management of the References
- The management of the Dictionary
- The design of the Rules



Tester:

The Tester is responsible for:

- The execution of the Stories
- The management of the Datasets
- The management of the Dummy Users



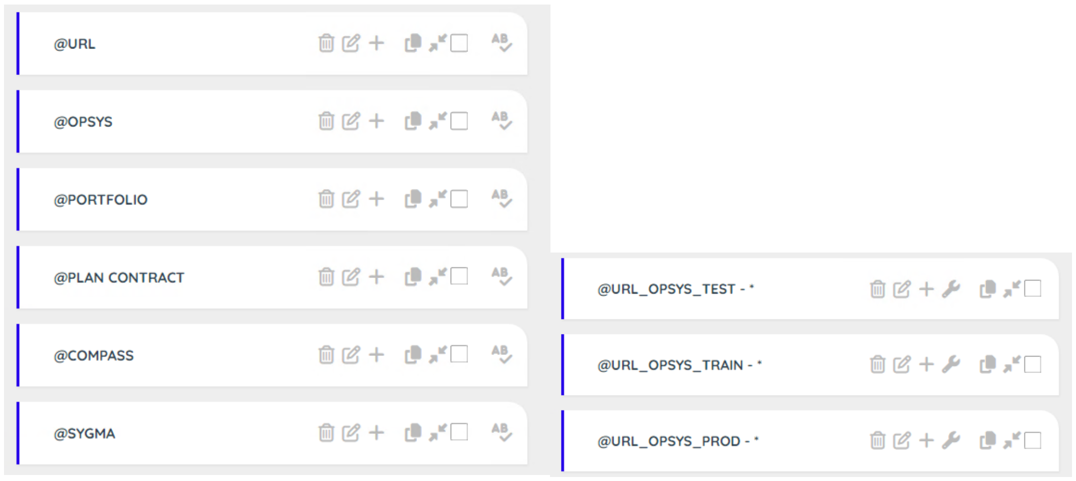
The Dictionary

The dictionary is composed of two elements: a header and a detail.
The header is used to group a set of dictionary definitions.
A dictionary is global to your project.

My suggestion is:

- Create a dictionary header to store the links to the application (E.g.: @URL)
  Define the links of the different environments,
  Define the links to have a direct access to some specific sections.
- Create a global entry for the generic word (E.g: @<name of your application>)
  Define the links of some common section (E.g.: a welcome page, a dashboard, a password field…)
- Create a header for each specific use case (E.g: Contract, Amendment)
  Define all the specific links required for the test (E.g.: access to a table value).
  However, I suggest using when it's possible the detectGUI function to reduce the number of entries in the dictionary. Also, if your application is still under development or not stable, a generic pattern will be valid for more long time than a very precise xpath.

  <u>Note</u>: the dictionary contains a field language, so it's possible to store translations.



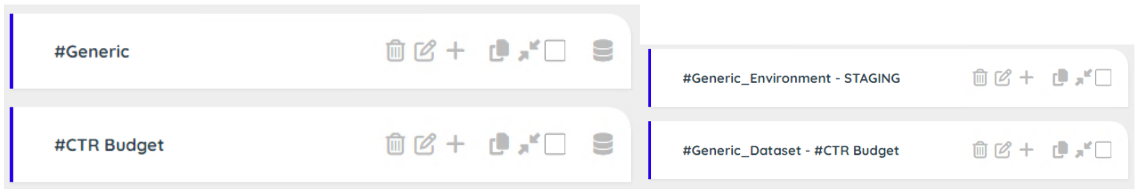
The Dataset

Like the dictionary, the dataset is composed of two elements: a header and a detail.
The header is used to group a set of data.
A dataset is valid for a project and a subproject (for all the users)

My suggestion is:

- Create a generic dataset to store global values (E.g.: #Environment, #Dataset)
  In this case, if you need to work into another environment, you have a central access point to the data.
  Define the links to have a direct access to some specific sections.
- Create a header dataset for each specific use case (E.g: Contract Blue, Amendment Red…)
  Define all the values required for the test (E.g.: a name, an amount, a domain…)
  Define specific value to transform your tests into data driven tests (E.g: a contract type can impact the workflow of your scenario)

Note: it's a good practice to associate for each story a setup scenario where you copy the most relevant data into the reference entity (reference is specific to a user). Like that the tester can execute the setup and update some values (E.g.: Amount) to avoid the need to duplicate a dataset!


#Generic
#CTR Budget
#Generic_Environment - STAGING
#Generic_Dataset - #CTR Budget

The Reference

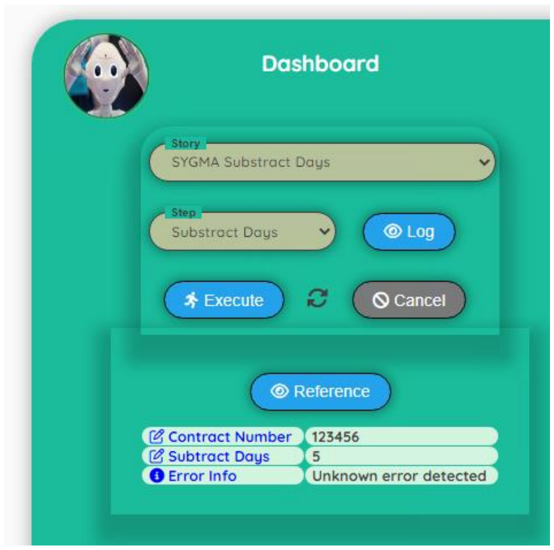
The reference is used to store all the data that we need to share with the scenarios (or stories).
A reference is valid for a project and a subproject and a specific user.

It's possible to import the reference of another user (for instance, people 'A' starts the creation of the contract, people 'B' imports the reference from user 'A' and continue the tests with an amendment…)

My suggestion is:

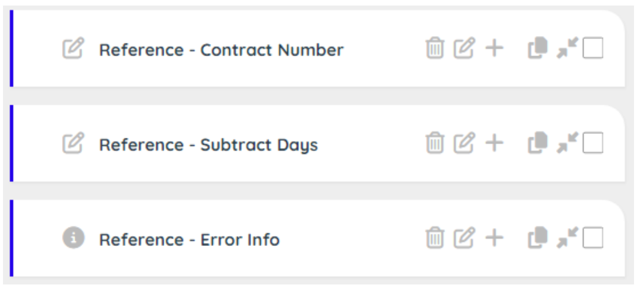
In the scenario use the function setReference() to write a value into the reference.
The third parameter is the comment, it's a good practice to describe the objective of the reference.

In the story definition, you can decide what references you want to expose to the tester (input or output). Try to keep the list short.



Story: SYGMA Substract Days
Step: Substract Days

Contract Number	123456
Subtract Days	5
Error Info	Unknown error detected



Reference - Contract Number
Reference - Subtract Days
Reference - Error Info

The Scenario

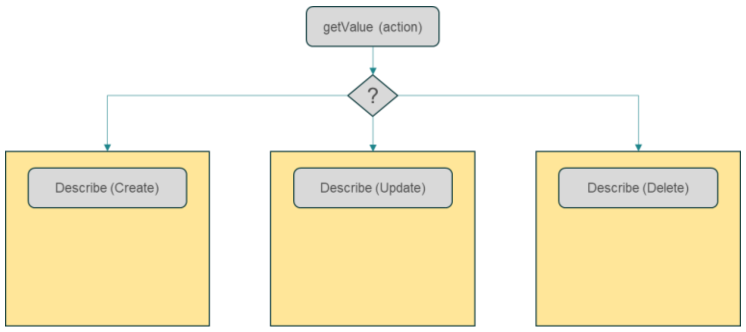
The scenario contains the tests (steps). It is visible by the Designer and the Admin but not by the Tester.

The step ‘Describe’ and ‘It’ are optional but I strongly suggest to use it to keep your scenario easily readable by the team.
The step ‘Describe’ offers the possibility to define a context. A context is a milestone for the Robot. If you define contexts and the test fails, the Robot will restart automatically after the last successful context.

Note 1: it’s a good practice to associate for each story a setup scenario where you copy the most relevant data into the reference entity (reference is specific to a user). Like that the tester can execute the setup and update some values (E.g.: Amount) to avoid the need to duplicate a dataset!

Note 2: You can use the dataset to create data driven scenarios.
Dataset has a specific behaviour when you key the same domain with different values.
In this case, the Robot will select randomly a value for the set of data!

This feature is very useful if you want to create dynamic data. For instance, if you create multiple record for the name (John, Tom, Eric, Phil, Cathy, Maria, Kelly) you will create a test with a random name. Moreover, you can use the dataset to drive your test (data driven), so depending on a data, you will execute a specific section of a scenario. If you have a huge number of use cases and you want to create a automatic sanity check, you can use this feature to randomly check a specific use case every time your run the scenario!

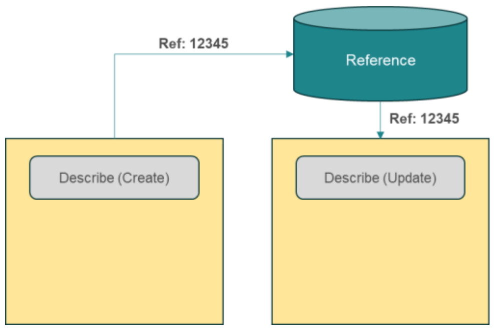


The Suite

The suite is optional and contains links to the scenarios. Suite is visible by the Designer and the Admin but not by the Tester.

Why do you need to create suite?
To answer to this question, I will give you an example where the suite offers a real added value.

Example: When creating a new contract, we get a new reference. With this reference, we can update a new section (searching for the new created reference., update the fields)



<u>Solution A:</u> One big scenario with the creation and the update, but if the update fails, we need to create again a new contract with a new reference.

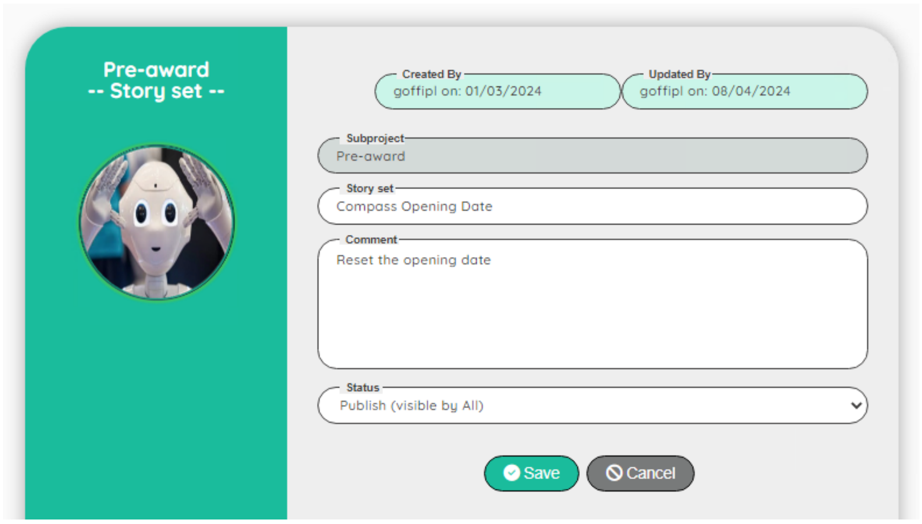
<u>Solution B:</u> We split the scenario into two parts: 1) create the new contract and store the reference, 2) read the reference and update the section. This solution is ok, because if the update fails, we don't need to create a new contract. However, for the tester, this is not logical because from a business point of view, the creation of a contract includes updating the section!

<u>Solution C:</u> we create a suite with the scenario 'Create' and the scenario 'Update'. If the scenario 'Update' fails, it will be managed by the Robot and for the Tester only one big scenario is visible.

<u>Note</u>: it's possible to use the solution A with a context 'Create' and a context 'Update'. In this case, if the section 'Update' fails, it will be managed by the Robot. However, the solution C has the advantage to keep the scenario short!

The Story

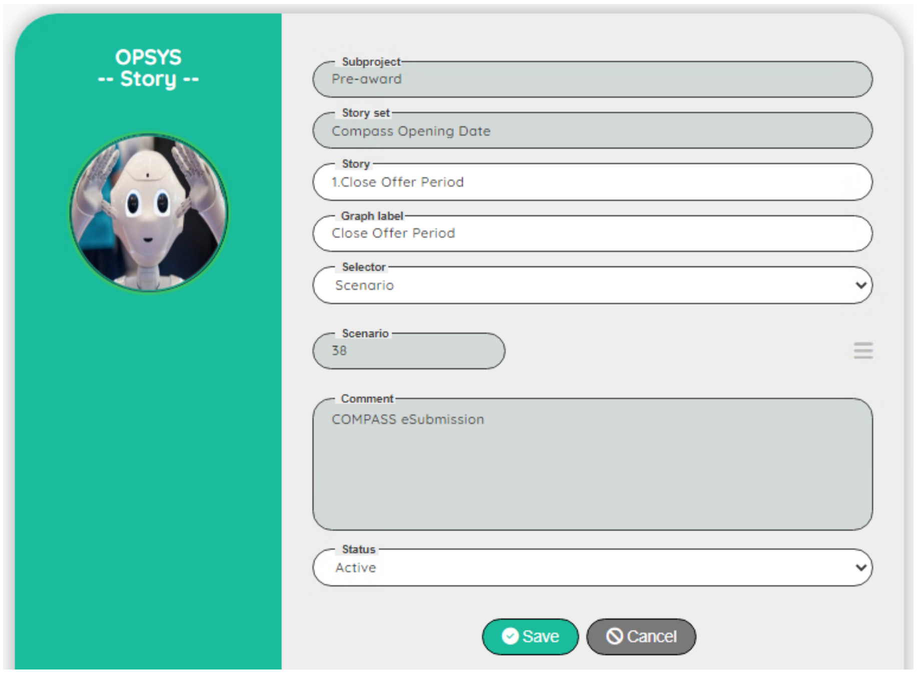
The story is the unique way to present a test to a Tester.
However, to be visible by a tester, the status must be published (by default it is just active and visible by the Designer and the Admin). Like that, as a designer, you can use the story to execute 'private' tests (E.g.: Sanity check or very specific admin task) and 'public' tests (published)



Created By: goffipl on: 01/03/2024
Updated By: goffipl on: 08/04/2024
Subproject: Pre-award
Story set: Compass Opening Date
Comment: Reset the opening date
Status: Publish (visible by All)

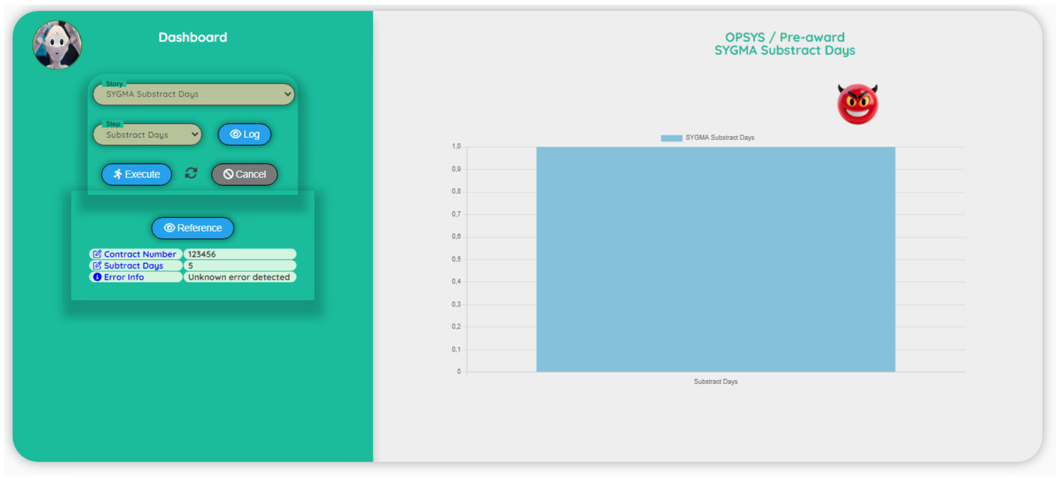
The story is oriented to the Tester with the idea to be easy to use.

A story contains the links to the scenarios and/or the suite and offer the possibility to rename the tests with a more business words.



Subproject: Pre-award
Story set: Compass Opening Date
Story: 1.Close Offer Period
Graph label: Close Offer Period
Selector: Scenario
Scenario: 38
Comment: COMPASS eSubmission
Status: Active

The story is executed from a dashboard, where the Tester has a visibility on the execution.

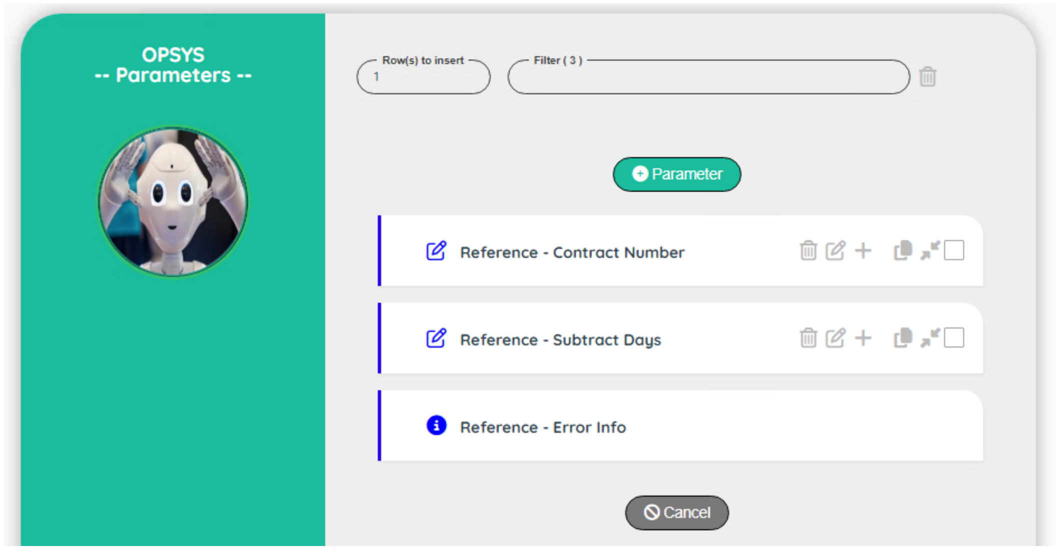


Story: SYGMA Substract Days

Step: Substract Days

Contract Number	123456
Subtract Days	5
Error Info	Unknown error detected

A story contains the links to the reference to display to the tester (Input or Output)



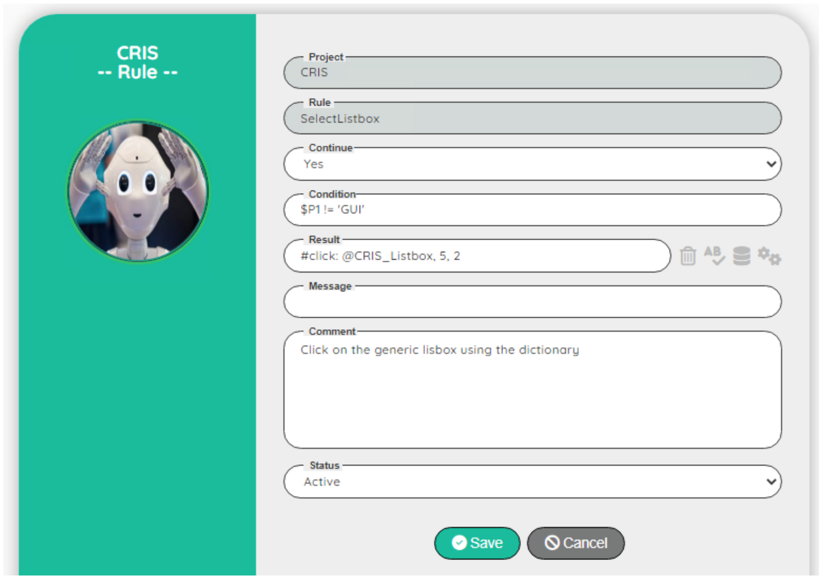
The Rules

The rules contain a set of steps that can be reused in the scenarios.
Rules are visible by the Designer and the Admin but not by the Tester.
Rules are defined at the level of the project.

A rule is like a subroutine and can be used to allow the Robot to make decision.
A rule step is composed of a condition and a result (if condition is true than evaluate the result).
condition and result must be defined in a JavaScript format.

All the functions that you can use in a scenario are available.
To call a Robot function, you must add the prefix # in the name of the function.
Example: the click function will be called #click in the rule.

When you call a rule in a scenario, you can pass up to two extra parameters.
In the rule, the parameters are identified with the variable $P1 and $P2.

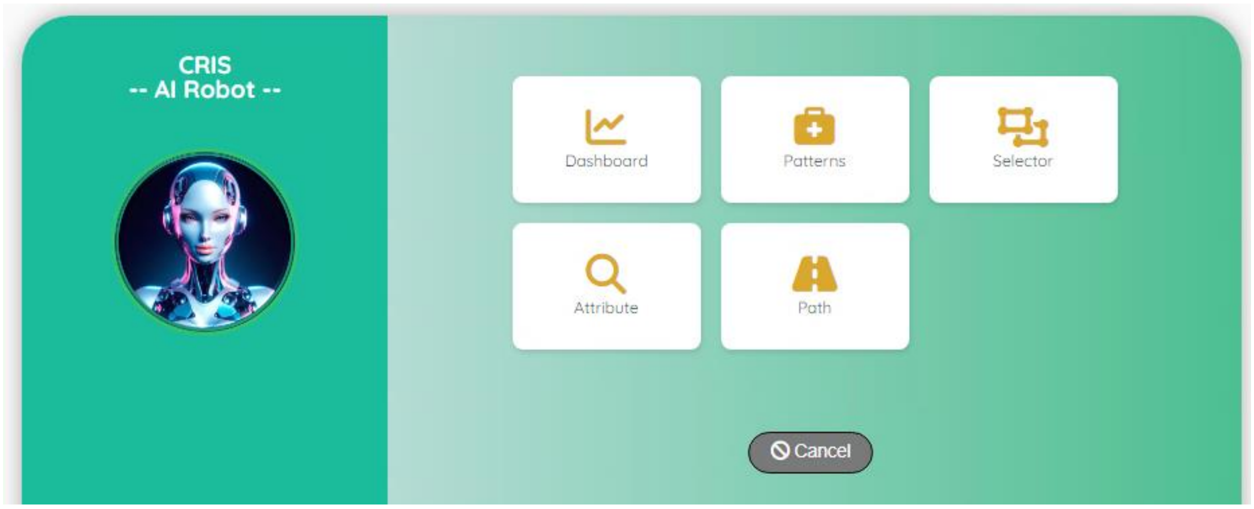


Project	CRIS
Rule	SelectListbox
Continue	Yes
Condition	$P1 != 'GUI'
Result	#click: @CRIS_Listbox, 5, 2
Message	
Comment	Click on the generic lisbox using the dictionary
Status	Active

You can see the rules as an open-source subroutine available for all the projects.
If you use an exotic framework, you will need to code the specific behavior of the component with a rule.
Example: a homemade calendar or a list box not designed with the standard html tag <select> and <option>

AI Robot

The AI Robot is another section in the Robot application.
It allows the Administrator to create generic patterns to detect elements on the screen (via the function detectGUI).
To use this section, the Administrator needs to have a good skill in html and xpath as he will act as the 'Expert' to tell the Robot what is the best path in all the proposed solutions!



To define a strong generic pattern, it's sometimes required to train the Robot multiple times.
Indeed, based on statistics, the Robot will improve the pattern at each training session.
However, you can speed up the process by checking the attributes to remove or simplify.
Example: you train the Robot to detect a button:

- In the attribute, you see a specific ID or name (E.g.: ID=Save_Button) – remove this attribute, otherwhile your pattern will not work with the Cancel button (ID=Cancel_Button)
- In the attribute there is a class 'ux-button ux-primary ux-grey') – see if 'ux-primary' and 'ux-grey' are consistent for all the buttons. If it is not the case, remove the attribute or simplify it (E.g: keep just the 'ux-button').
  Be careful when you simplify an attribute, you can remove the beginning or the end but not the middle of the value.
  with 'A B C D' valid simplifications are: 'A', 'A B', 'A B C', 'B C D', 'C D' and 'D'
  Invalid simplifications are: 'A C', 'A C D', 'A B D'…

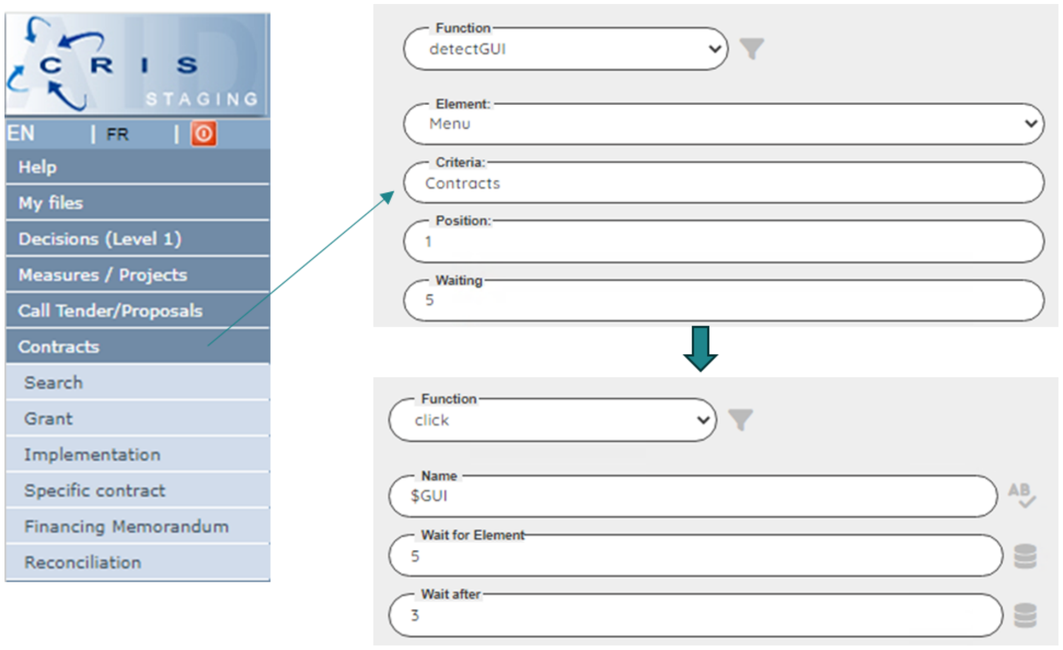
As already said, in the Dictionary section, the use of the detectGUI function reduces the number of entries in the dictionary. Also, if your application is still under development or not stable, a generic pattern will be valid for more long time than a very precise xpath.

You can also boost your productivity in the creation of the scenarios.
With the use of the dictionary, you need to inspect the screen and define the correct path to get the element.
In some cases, the xpath is not very easy to design and required a strong skill.

With the use of the AI Robot, during the training, the Robot will generate possible xpaths for you and your job will be to select the best one (easier to decide which is best than designing from scratch!)

With the detectGUI function, you just have to look at the screen, decide what type of element you are processing (input filed, menu, submenu…) and what label to use as search criteria.

Example: in the application CRIS, we have a menu Contracts, to click on this item, we can search for the menu with the detectGUI function searching for the label 'Contracts'.
Now, to click on the item, we can pass $GUI as the parameter for the click function.



Function: detectGUI
Element: Menu
Criteria: Contracts
Position: 1
Waiting: 5

Function: click
Name: $GUI
Wait for Element: 5
Wait after: 3