
Read the ExtensiveAutomation Documentation

Release 20.0

ExtensiveAutomation

Aug 10, 2019

1	Extensive Automation	1
1.1	Concepts	1
1.2	Usages	2
1.3	License	2
1.4	Author	2
1.5	Contact	3
2	Download	5
3	Changes Logs	7
3.1	Current version	7
3.2	Previous versions	7
4	Client	15
4.1	The workspace	16
4.2	The analyzer	24
4.3	Server Explorer	26
4.4	Settings	27
4.5	Complements	28
5	Toolbox	33
5.1	Deployment	34
5.2	More	35
6	Web interface	39
6.1	Tests part	39
6.2	Admin part	39
7	Get started	41
7.1	Connection to the server	41
7.2	Write a test	42
7.3	Write a scenario	42
7.4	Execute a test	44
7.5	Result analysing	44
7.6	Best practices	45
8	Tests examples	47

8.1	Testcase (unit)	47
8.2	Testcase (suite)	47
8.3	variables	48
8.4	Scenario	49
8.5	Test campaign	50
8.6	Rest API	50
8.7	SSH controls	52
8.8	Web browsers	52
8.9	Android mobile	55
9	Tests Snippets	59
9.1	Shared data between tests	59
9.2	Basics actions	61
9.3	Data Generators	63
9.4	Networks protocols	65
9.5	User Interface	69
9.6	Checks	70
10	Global variables	73
10.1	Add/delete a variable	73
10.2	Describe environment test	73
10.3	Import/export variables	74
11	Assisted designs	75
11.1	Framework Tabulation	76
11.2	System tab	77
11.3	Tabulation application	78
11.4	Browser Tabulation	80
11.5	Android Tabulation	82
12	Troubleshooting	87
12.1	Errors codes	87
12.2	How to fix	87
13	Installation	89
13.1	Server	89
13.2	Plugins for server	89
13.3	Web Client	89
13.4	Qt Client	89
13.5	Toolbox	89
13.6	Plugins for client	89
14	Administration	91
14.1	Start/Stop of the server	91
14.2	Server status's	92
14.3	Server settings	92
15	Projects	93
15.1	Add a project	93
15.2	Delete a project	93
15.3	Link a project to a user	93
16	Users	95
16.1	Add user	95
16.2	Delete a user	95

17 Troubleshooting	97
17.1 Logs	97
17.2 Frequently Asked Questions	99
18 Tests definitions	101
18.1 Test Unit	101
18.2 Test Suite	102
18.3 Test Plan	102
18.4 Test Global	102
19 The fundamentals	103
19.1 Test case	103
19.2 Test steps	104
19.3 Cancellation of a test	104
19.4 Adding trace	105
19.5 Data	105
19.6 Put on hold	107
19.7 Interaction with the tester	107
19.8 Parameters of a test	108
20 The tracability	111
20.1 The events	111
20.2 Test reports	114
20.3 The logs	118
21 Interoperability	119
21.1 Adapters	119
21.2 Bookstores	121
21.3 Third party tools	124
21.4 Agents	125
22 Test engine	127
22.1 The scheduler	127
22.2 Parallelized executions	129
22.3 Synchronized executions	130
22.4 Distributed executions	131
23 Advanced examples	133
23.1 SSH adapter	133
23.2 HTTP adapter	134
23.3 Telnet adapter	135
23.4 MySQL adapter	136
23.5 SNMP adapter	137
23.6 FTP adapter (s)	138
23.7 SFTP adapter	139
23.8 ChartJS librairies	140
23.9 “Text” parameter	142
23.10“Json” parameter	143
23.11“alias” parameter	143
23.12“global” parameter	144
23.13“Dataset” parameter	144
23.14Using an agent	145
24 Requirements	147
24.1 Server	147

24.2	Customer	147
24.3	Toolbox	148
25	Architecture	149
25.1	Server	149
25.2	Graphic Client	150
25.3	Agents	150
26	Specifications	151
26.1	Version cycle	151
26.2	Server tree	151
26.3	Data model	152
26.4	Passwords management	152
26.5	File format	153
26.6	Storage of test results	154
26.7	Control Agents	155
26.8	The server logs	156
27	Contributions	157
27.1	Solution development	157
27.2	Plugins development for servers	157
27.3	Plugins development for qt client and agents	157
27.4	Documentation	158
28	API	159
28.1	Authentication	159
28.2	Usage example	160
28.3	Ressources	160

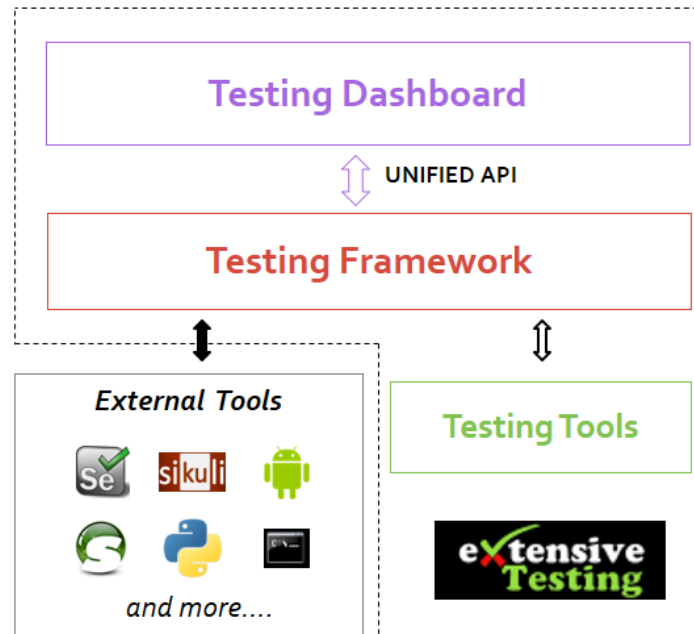
Welcome on the Extensive Automation project.

1.1 Concepts

ExtensiveAutomation is a generic automation framework for integration, regression and end-to-end usages. The framework provided a rich and collaborative workspace environment.

The project have several purposes since the creation:

- Unify all testing tools in one environnement
- Provide a complete testing dashboard
- Make automation easy
- Automate everything from anywhere.
- Provide a user friendly environment.



1.2 Usages

This solution is designed to make a lot of thing like:

- test automation in integration environment
- regression test automation
- end to end test automation
- deployment automation

Note: The solution is written in Python and also all tests.

1.3 License

This environment is open source, and is freely available under the LGPL 2.1 version. All source code is available in github (<https://github.com/ExtensiveAutomation>).

1.4 Author

Started since 2010, the solution was developed by Denis Machard. This project is an effort, driven in my spare time.

If you want to sponsor me then I am accepting donations via PayPal. The money received will be used to cover web site costs, domain name reservation and more :wink:.

1.5 Contact

If you have any questions, you can contact me with:

- email d.machard@gmail.com
- twitter https://twitter.com/Extensive_Auto
- from the google forum <https://groups.google.com/forum/#!forum/extensive-automation-users>
- github <https://github.com/ExtensiveAutomation>

CHAPTER 2

Download

A complete release is generated every two months or less and each component can also evolve at its own rate. The solution is made of several components.

The solution can be downloaded from the website <https://www.extensiveautomation.org>

The server is available through several types of packages:

- from sources
- docker image
- tar.gz

The graphical client is available in portable version.

The toolbox is available in 2 modes:

- portable version
- command line

Note: Client and toolbox can be run on Windows and Linux, in 64bits only.

Warning: The server must be executed in Linux.

Warning: Administrators rights can be necessary to execute properly the client or the toolbox.

3.1 Current version

Note: Version 20.0.0 available since 07/20/2019

- Image docker available
 - Rest API: support for CORS feature
 - No more automatic installation provided
 - Maximum of dependancies with libraries removed
 - No more plugins provided by default
 - Probes features removed, replaced by agent
 - MySQL database removed, replaced by sqlite
 - Optimization of the framework to reduce the CPU usage
 - New major version for the Qt client application, user interface improvment
 - New major version for the toolbox
 - New major version for the web interface, no more provided by default
 - Several bugs fixed
-

A detailed release notes is available in the installation package.

3.2 Previous versions

Version 18.0.0 released on 02/11/2018

- API XmlRPC is removed from the server
-

- Big improvement of the API REST
- New major client based on the API Rest
- Full support of Qt5.9 for the toolbox and graphical client
- Full support of python 3.6 for the toolbox and graphical client
- Code cleanup
- Several bugs fixed
- Update of selenium in 3.9.0
- The toolbox is no more embedded in the server side by default

Version 17.1.0 released on 10/22/2017

- Improvement in the REST api
- New features in the test framework library
- Several bugs fixed
- Improvement of the graphical interface of the client
- Experimental Ubuntu Support for the graphical client

Version 17.0.0 released on 06/04/2017

- 64bits support by default for the client and toolbox
- New major features in the test framework library
- New swagger for the REST api
- Update of selenium 3 et 2 in the toolbox
- Several bugs fixed

Version 16.1.0 released on 03/30/2017

- Several bugs fixed
- Improvement of the graphical interface of the client
- Installation improvement

Version 16.0.0 released on 25/02/2017

- Several bugs fixed
- Improvement of the REST api
- Changes on core server
- New features in the test framework library
- Optimization in server side to reduce the number of SQL requests
- Improvement of the graphical interface of the client
- 64bits Support for the graphical client and toolbox

Version 15.0.3 released on 04/11/2016

- Several bugs fixed
- New plugins for the graphical client
- Improvement of the REST API

- New features in the test framework library
- New interop module in test library

Version 14.0.0 released on 27/08/2016

- Several bugs fixed
- New features in the test framework library
- New major features on the REST api
- No more new feature in the XmlRPC api, just bug fix
- New features in the web interface
- Python2.7 no more supported on windows for the graphical client and toolbox
- Integration of the REST api in the graphical client
- Improvement of the graphical interface of the client
- New HP QC ALM plugin for the graphical client

Version 13.0.0 released on 23/06/2016

- Several bugs fixed
- New REST api on the server side
- New features in the test framework library
- Improvement in the core server
- Plugins support for the client and toolbox
- Improvement of the graphical interface of the client

Version 12.1.0 released on 29/04/2016

- Several bugs fixed
- New features in the test framework library
- Minors update on the XmlRPC API
- Improvement of the graphical interface of the client

Version 12.0.0 released on 12/02/2016

- Several bugs fixed
- New features on the XmlRPC API
- New features in the test framework library
- New features in the web interface

Version 11.2.0 released on 22/11/2015

- Several bugs fixed
- New features in the test framework library
- Improvement of the scheduler
- New public repository for the test framework library
- Offline installation support
- Minor changes on the XmlRPC api

Version 11.1.0 released on 18/10/2015

- Several bugs fixed
- New features on the XmlRPC API
- New features on the web interface

Version 11.0.0 released on 14/09/2015

- Several bugs fixed
- New features in the web interface
- Merge of agents and probes in the toolbox
- Update in the XmlRPC API
- Python 3.4 support for the graphical client and toolbox

Version 10.1.0 released on 12/07/2015

- Several bugs fixed
- CentOS 4 et 5 no more supported
- New features in the test framework library
- New features in the web interface

Version 10.0.0 released on 28/05/2015

- Several bugs fixed
- New features in the web interface
- Minor changes in the core server
- Update of the documentations
- Improvment of the graphical interface of the client

Version 9.1.0 released on 22/03/2015

- Several bugs fixed
- New features in the test framework library
- Product installation improved
- Improvment of the graphical interface of the client

Version 9.0.0 released on 05/01/2015

- Several bugs fixed
- New features in the test framework library
- Python 2.4 no more supported
- New features in the web interface
- Improvment of the graphical interface of the client

Version 8.0.0 released on 25/10/2014

- Several bugs fixed
- Improvment of the graphical interface of the client
- New features in the test framework library

- Minors changes in the XmlRPC API
- New features in the web interface

Version 7.1.0 released on 20/09/2014

- Several bugs fixed
- Documentations updated
- Optimization in server side to prepare a test
- New features in the core
- New features in the test framework library
- Improvement of the graphical interface of the client

Version 7.0.0 released on 08/08/2014

- Several bugs fixed
- Improvement in the scheduler
- Reverse proxy added on the front of the server
- Websockets support, activated by default
- New documentations
- tcp/443 used by default on all components
- SSL proxy support
- SSL used by default for agents and probes
- Improvement of the graphical interface of the client

Version 6.2.0 released on 02/06/2014

- Several bugs fixed
- Agents update
- Minors changes in the XmlRPC API
- New features in the test framework library
- Improvement of the scheduler

Version 6.1.0 released on 25/04/2014

- Several bugs fixed
- New features in the web interface
- New features in the test framework library
- Agents improvements

Version 6.0.0 released on 23/03/2014

- Several bugs fixed
- New packages for adapters and libraries
- New features in the XmlRPC API
- New features in the test framework library
- No more link with the twisted library

- SSL support on XmlRPC api
- Proxy socks4 support
- Agents Support

Version 5.2.0 released on 12/01/2014

- Several bugs fixed
- New minors features in the core server

Version 5.1.0 released on 08/12/2013

- New features in the web interface
- Several bugs fixed
- New features in the test framework library

Version 5.0.0 released on 15/09/2013

- Several bugs fixed
- New major features in the test framework library
- Improvement of the scheduler

Version 4.2.0 released on 08/04/2013

- Several bugs fixed
- New features in the web interface

Version 4.1.0 released on 10/03/2013

- Several bugs fixed
- New features in the web interface
- CentOS 6 Support
- Improvement of the scheduler

Version 4.0.0 released on 30/01/2013

- Several bugs fixed
- New features in the test framework library
- SSL support for the web interface
- New authentication method with sha1 and salt
- New features in the XmlRPC API

Version 3.2.0 released on 29/09/2012

- Several bugs fixed
- New features in the test framework library

Version 3.1.0 released on 14/07/2012

- Several bugs fixed
- New features in the test framework library
- Improvement of the scheduler
- New features in the XmlRPC API

Version 3.0.0 released on 09/06/2012

- Several bugs fixed
- New features in the XmlRPC API
- Improvement of the scheduler
- New repositories for adapters and backups

Version 2.2.0 released on 28/03/2012

- New majors features in the XmlRPC API
- Several bugs fixed
- New features in the test framework library

Version 2.0.0 released on 27/02/2012

- New features in the XmlRPC API
- Documentation added for the test framework and adapters
- Several bugs fixed
- Probes support

Version 1.2.0 released on 14/01/2012

- Improvement of the scheduler
- New features in the XmlRPC API
- New features in the test framework library
- Interface web added
- Several bugs fixed

Version 1.0.0 released on 13/12/2011

- First official version
- CentOS 5 support
- Several bugs fixed

Version 0.1.0 released on 17/05/2010

- First beta

The client allows to write and execute automatic tests but also to analyze the results in real time or deferred. It also allows you to share tests in a simple and effective way. To use the client, you must have a user account and be able to connect to the test server (tcp/443).

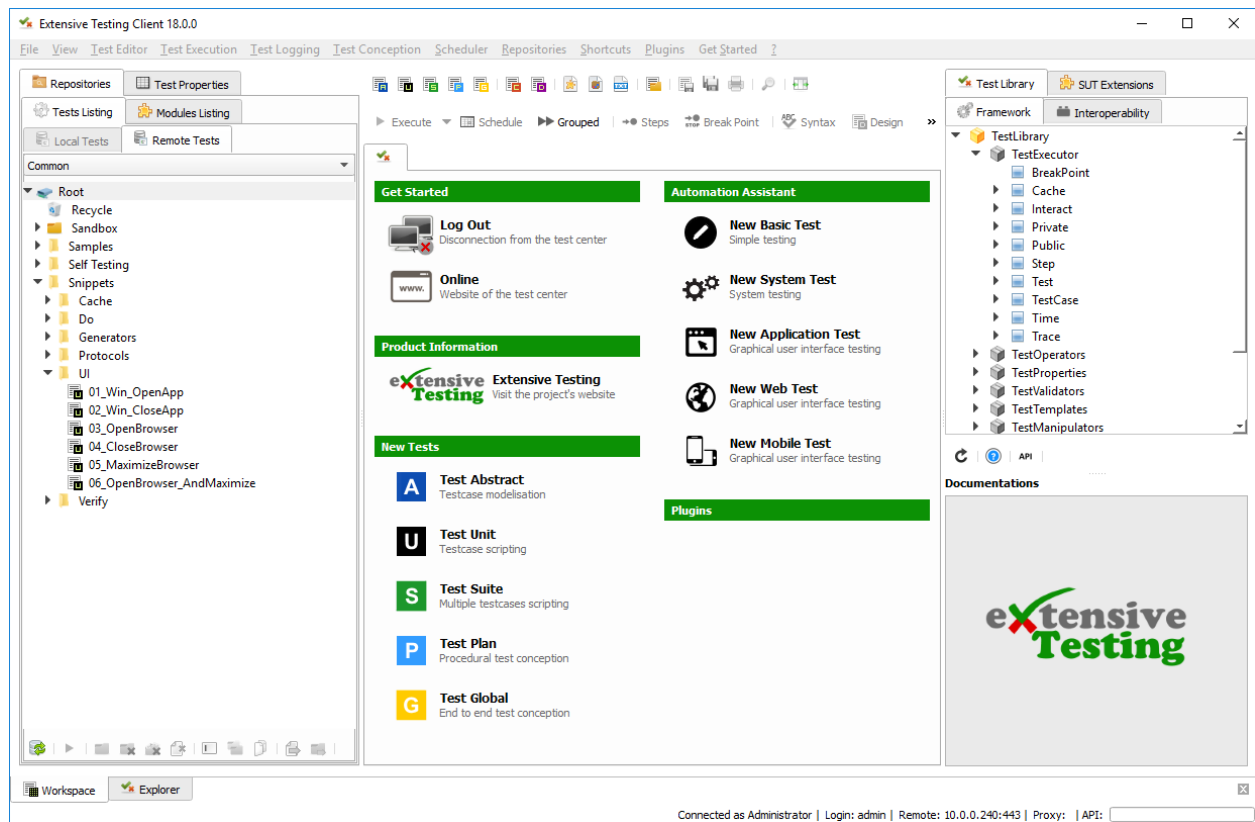
The client can also be used to develop extensions (adapters and libraries) to communicate with the system to be tested or piloted.

Finally the graphical interface changes according to the level of access:

- tester level: write / execute tests, and analyze the results
- admin level: access to all features
- monitor level: read only access

The interface is divided into 3 main parts:

- the workspace
- the analyzer
- the server explorer

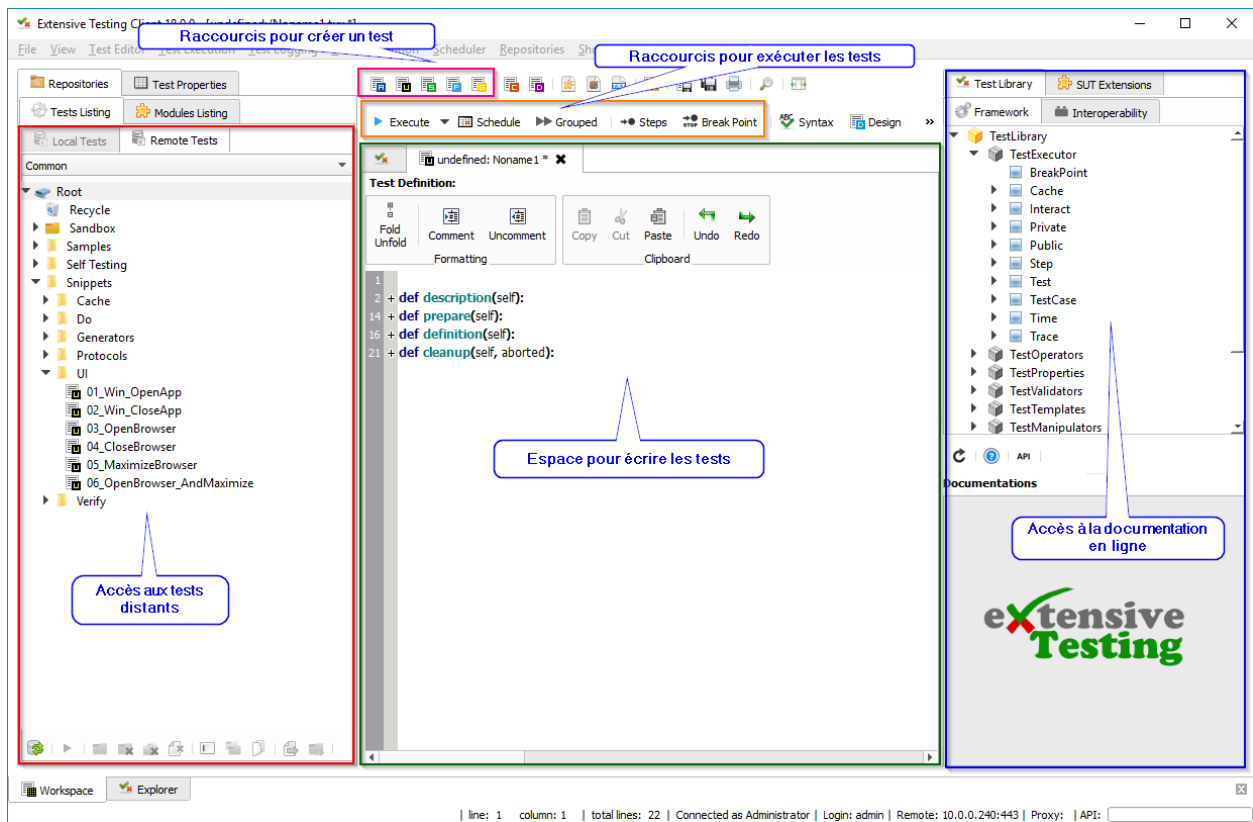


Note: The client is available on Windows and Linux, in 64bits mode

4.1 The workspace

The workspace is composed of 3 main parts:

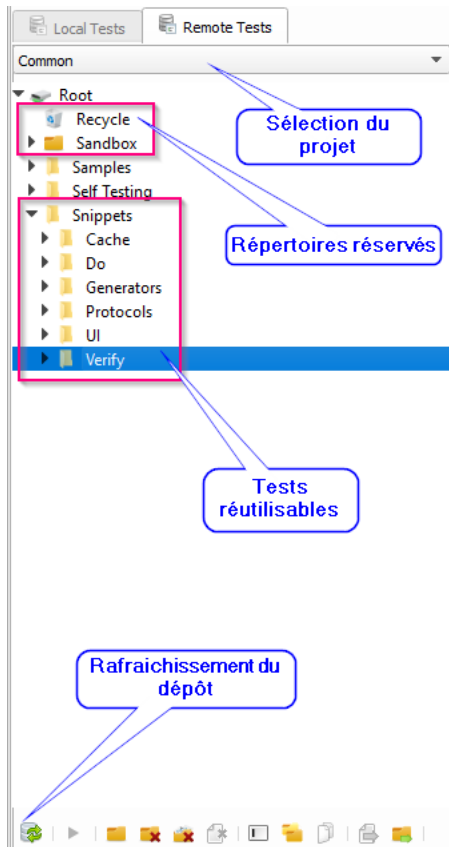
- access to all file repositories
- access to test design
- online documentation



4.1.1 Deposit of tests

The client provides access to the two test repositories: remote and local.

The remote repository allows to store its tests on the test server, so to share them with other users. The tree consists of files and directories. Test management can be done from the client. The tests can be organized by project if necessary.



Note: The Common project contains re-usable tests and various examples.

Note: The directories Recycle and Sandbox are reserved directories, delete them is impossible.

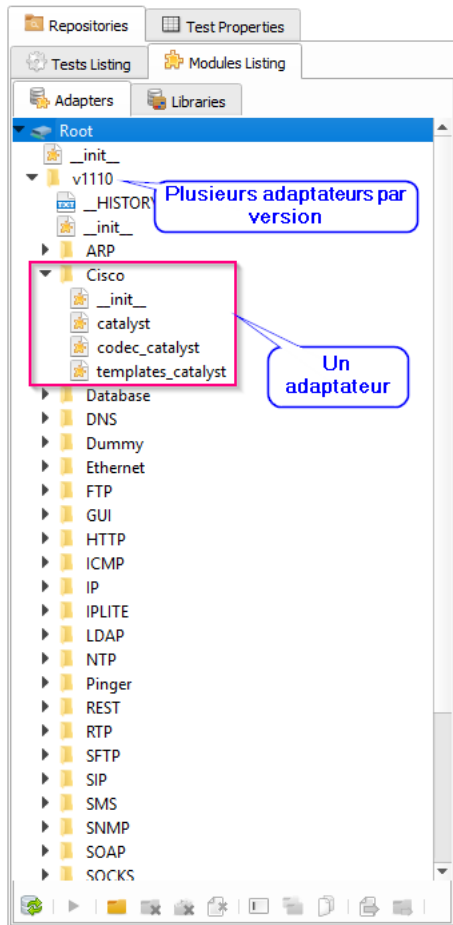
Note: It is possible to open a test by dragging and dropping the file to the writing space.

The **local repository** gives the possibility to store his tests on his post, so not shared. This feature is not enabled by default because it is not in the philosophy of the solution to use it. Nevertheless the deposit can be activated through the user's preferences.

Warning: Some features are missing in the local repository, its use is not recommended!

4.1.2 Depositing extensions

The client allows access to the depots of the extensions (adapters and libraries) and can also be used to develop new ones, which will be stored there too. These extensions are organized by version.



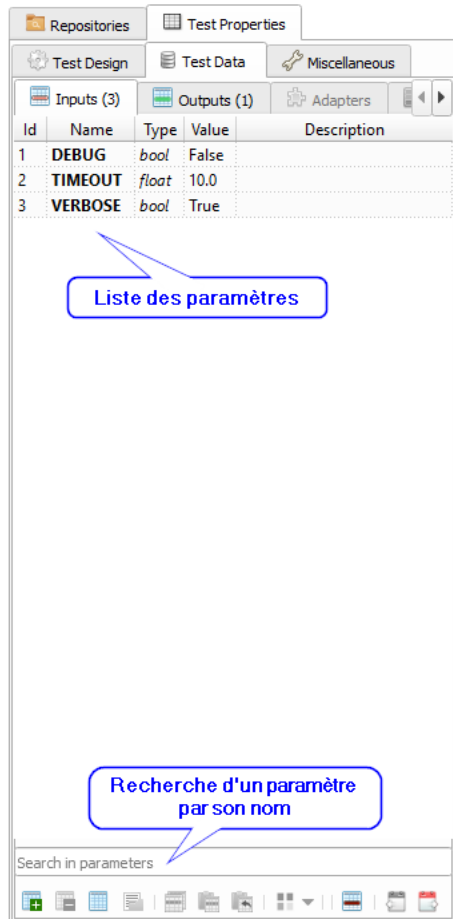
Note: The extensions are developed in Python.

4.1.3 Tests properties

The tests can be enriched with a number of properties. Available properties are:

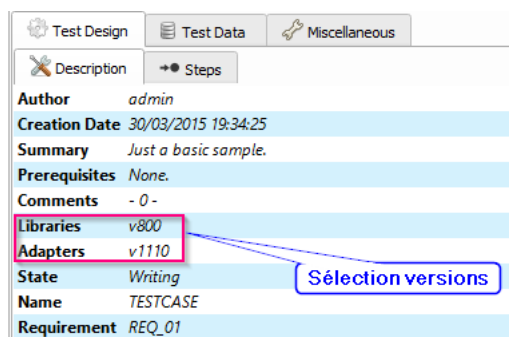
- the description of the test (author, date of creation, etc ...)
- incoming and outgoing variables
- the definition of agents and probes used by the test

The Test properties> Test Data> Inputs window contains the list of variables accessible from the test. Adding variables can be done by right clicking 'Add parameter'.



Note: To insert a parameter into a test, just drag & drop.

Note: It is possible to choose the version of the adapters and libraries to use for the test

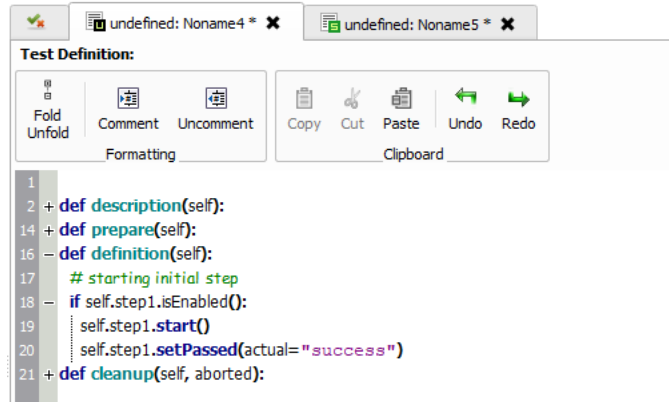


4.1.4 Textual design

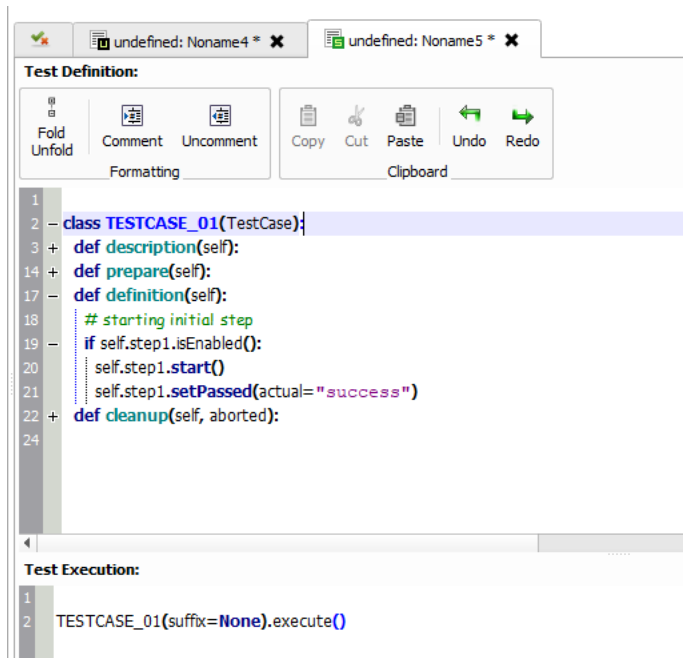
The design of a scripting test is possible with unit and suite. This design mode requires knowledge in development, i.e. python.



The unit test is a test case. It is divided into 4 sections automatically called by the framework.



The suite test represents one or more test cases. This type of test allows you to run the same test case by changing the input parameters.



Note: The Ctrl + F shortcut allows you to search for text in your tests.

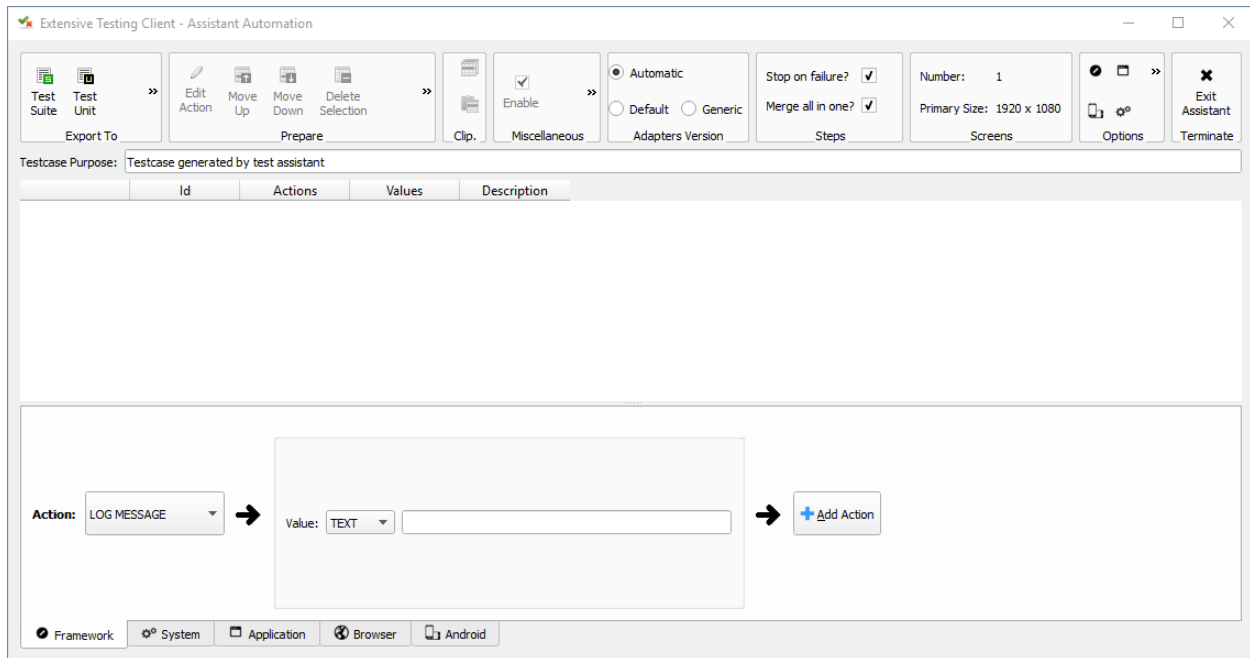
4.1.5 Assisted design

The design wizard allows you to write tests without knowledge in development. It covers the following actions:

- Call to the basic functions of the test framework

- SSH test
- Application test with screenshot (based on the Sikuli project)
- Website test (based on the Selenium project)
- Android mobile app test

The wizard is to describe the actions to perform, and if desired export them to a test unit or suite.

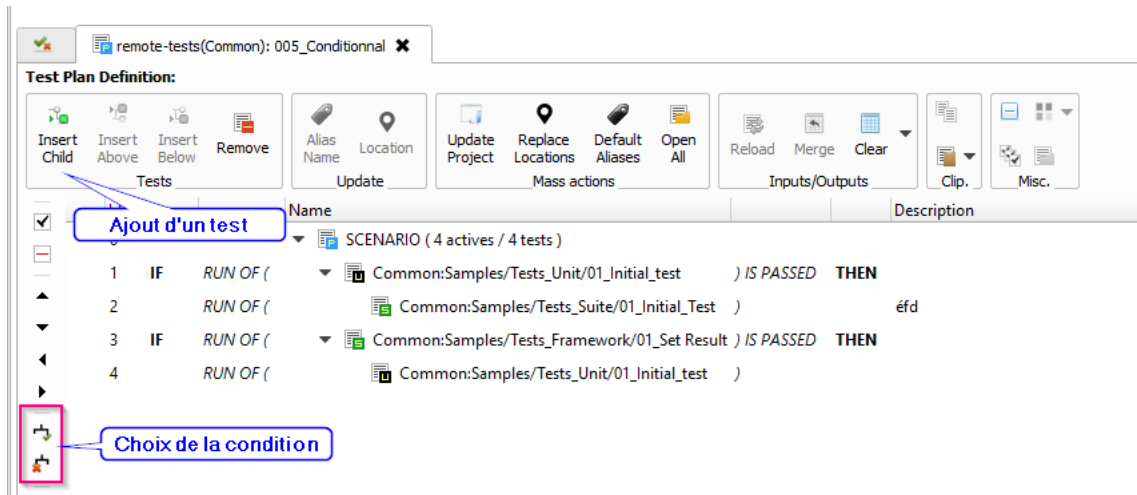


4.1.6 Conditional design

Conditional design allows you to build scenarios or test campaigns. This approach does not require developing knowledge. To perform this type of test, it is necessary to create a new plan or global test.



The test plan makes it possible to write test scenarios by including tests of the type unit or suite.

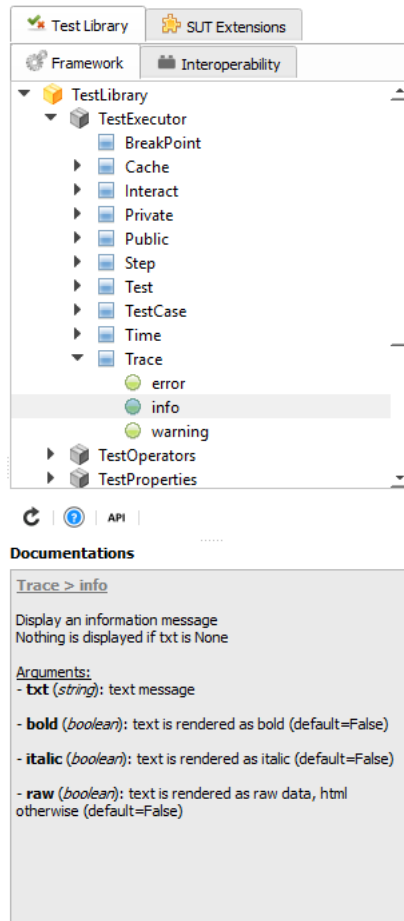


The global test is used to describe test campaigns by including tests plan, unit or suite.

Note: It is possible to override the test parameters.

4.1.7 Online documentations

The online documentation is generated by the server, it describes the set of available functions in the test framework and the different extensions.



Note: A drag & drop from the documentation on a test automatically inserts the skeleton of the function.

4.2 The analyzer

The analyzer makes it possible to follow the execution of a test in real time or deferred. It makes it possible to display all the events of the test and to facilitate the analysis of the errors.

The screenshot displays the Extensive Testing Client 18.0.0 interface. The main window shows a summary of test results and a detailed event log. The event log is filtered for 'DEBUG' events, showing a sequence of events from 'BEGIN' to 'END'. The event log table is as follows:

No.	Timestamp	From	To	Event Type	Component Type	Text
0	21:03:33.8686	TE	USER	INFO	TESTCASE	BEGIN [Id=#1]
1	21:03:33.8688	TE	USER	SECTION	TESTCASE	Designing
2	21:03:33.8689	TE	USER	SECTION	TESTCASE	Preparing
3	21:03:33.8690	TE	USER	SECTION	TESTCASE	Starting
4	21:03:33.8692	TE	USER	STEP-STARTED	TESTCASE [Step_1]	step sample
5	21:03:33.8695	TE	USER	STEP-PASSED	TESTCASE [Step_1]	success
6	21:03:33.8697	TE	USER	SECTION	TESTCASE	Cleaning
7	21:03:33.8699	TE	USER	INFO	TESTCASE	END

The interface also includes a summary table on the left, an event filter section, and a list of executed tests at the bottom left.

4.2.1 Visualization of events

Different types of events are possible (column event type):

- DEBUG
- INFO
- WARNING
- ERROR
- SEND
- RECEIVED
- STEP-STARTED
- STEP-PASSED
- STEP-FAILED
- MATCH-STARTED
- MATCH-INFO
- MATCH-STOPPED
- MATCH-EXCEEDED

Note: Filtering on the ERROR event allows you to quickly see why the test is in error.

Note: The SEND | RECEIVED filter is used to display messages sent or received by the system to be tested / piloted.

4.2.2 Detailed view

Selecting an event from the list displays the detailed view. The detailed view displays the content of the event and more.

20	18:26:04.2966	ADAPTER #45	USER	MATCH-STOPPED	HTTP>TCP [Match_0]	Template(0) match
21	18:26:04.2969	ADAPTER #44	SUT	SEND	HTTP	➡ GET /rest/administration/projects/listing HTTP/1.1
22	18:26:04.2974	ADAPTER #44	USER	MATCH-STARTED	HTTP [Match_0]	Wait the expected template(0) for 10.0 seconds
54	18:26:04.5485	SUT	ADAPTER #44	RECEIVED	HTTP	⬅ HTTP/1.1 200 OK (application/json)
55	18:26:04.5546	ADAPTER #44	USER	MATCH-STOPPED	HTTP [Match_0]	Template(0) match
57	18:26:04.5550	ADAPTER #45	SUT	SEND	HTTP>TCP	➡ disconnection
60	18:26:04.5556	SUT	ADAPTER #45	RECEIVED	HTTP>TCP	⬅ disconnected
61	18:26:04.5787	TE	USER	STEP-FAILED	TESTCASE [Step_1]	The response is KO. Checks of the response: Search
62	18:26:04.5791	TE	USER	SECTION	TESTCASE	Cleaning
63	18:26:04.9811	TE	USER	INFO	TESTCASE	END

Key

- IP4
- TCP
- SSL
- HTTP
 - body {"cmd": "/administration/projects/listing", "proje [...]
 - headers
 - response
 - phrase OK
 - code 200
 - version HTTP/1.1

Template Expected

Status Key	Status Value
IP4	
TCP	
SSL	
HTTP <ul style="list-style-type: none"> response <ul style="list-style-type: none"> version regex("HTTP/1.[10]") phrase OK code 200 headers <ul style="list-style-type: none"> regex("[c]ontent-[tT]y... regex("application/json.*") 	

4.3 Server Explorer

4.3.1 Visualization of the results

The complete history of test results is available from the client. They are sorted by date and time of execution. The client can display the reports and download the logs generated during the execution of the test.

Project:

Common

Refresh

Partial

Full

Empty all

Delete Result

Open

Manage

Exports

Name	Result	Run	Replay Id	Created	User	Size on disk
Root						
2018-01-14		1				
Noname1	✓			2018-01-14 21:03:32	admin	
private_storage.zip	✓	0		2018-01-14 21:03:33		22 Bytes
Noname1.trx	✓	0				5.5 KB
2018-01-07		15				
/Self Testing/USER INTERFACE/000_Client_Launch_...	⚠			2018-01-07 18:37:15	admin	
/Self Testing/SYSTEM/000_System	✓			2018-01-07 18:36:57	admin	
/Self Testing/SYSTEM/000_System	✗			2018-01-07 18:35:57	admin	
/Self Testing/REST API/0001_Rest_Api	✗			2018-01-07 18:25:49	admin	

4.3.2 Visualization of test reports

Test reports are visible directly from the client. Two types of reports are available:

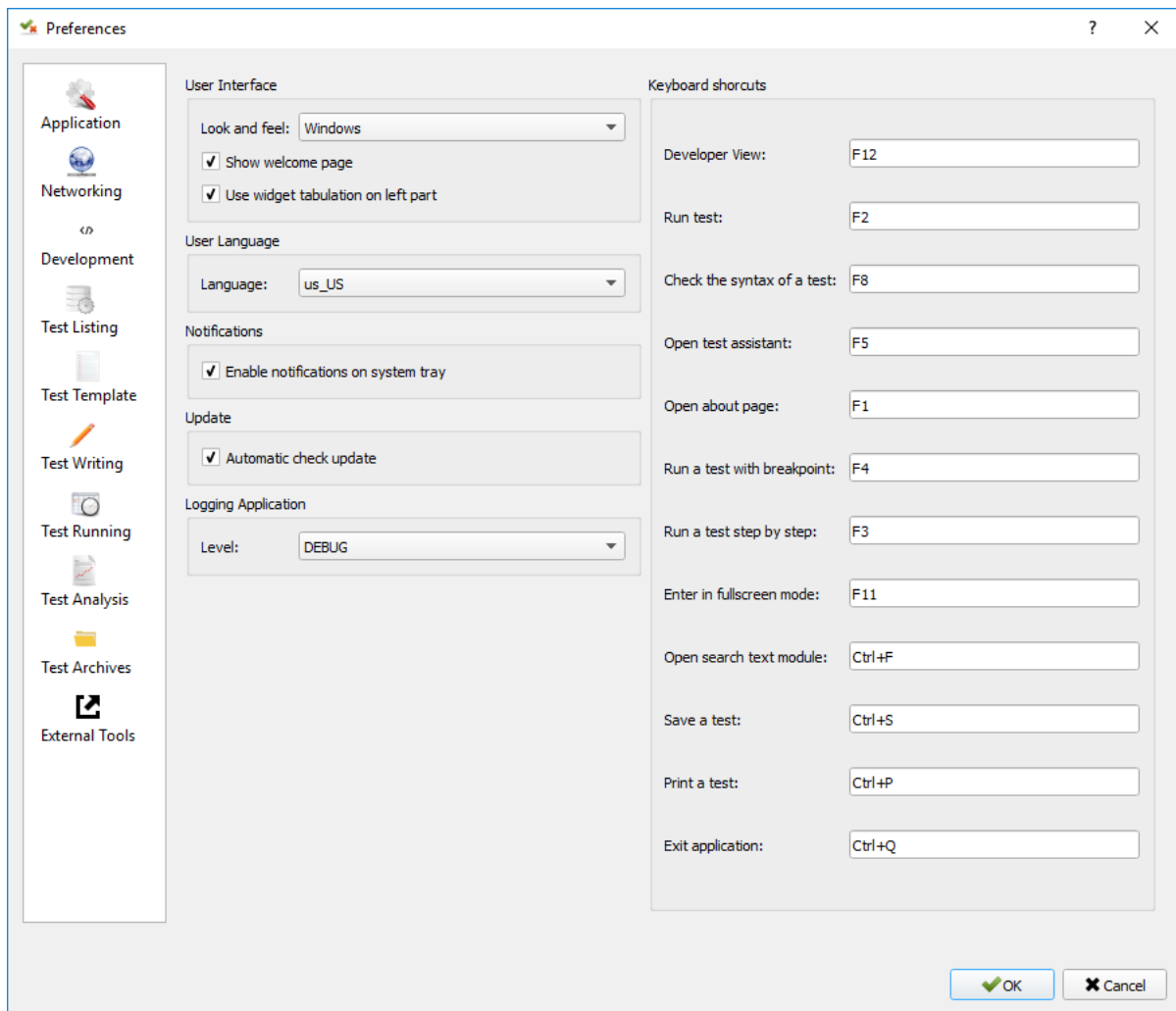
- advanced report
- simple report



Note: The reports are exportable in html, xml and csv formats.

4.4 Settings

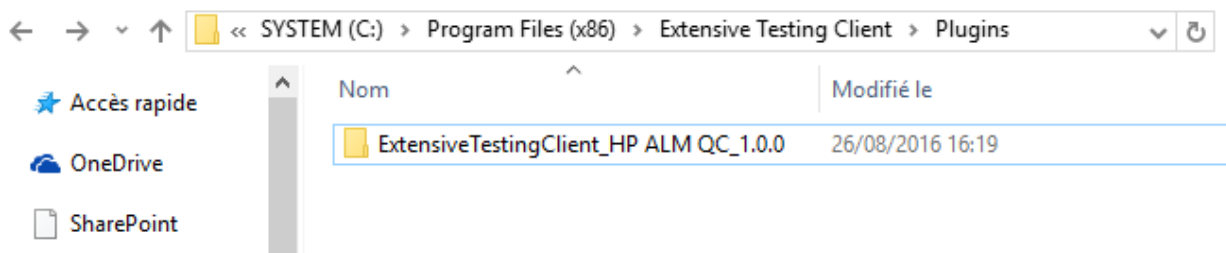
Client behavior can be changed through the user's preferences.



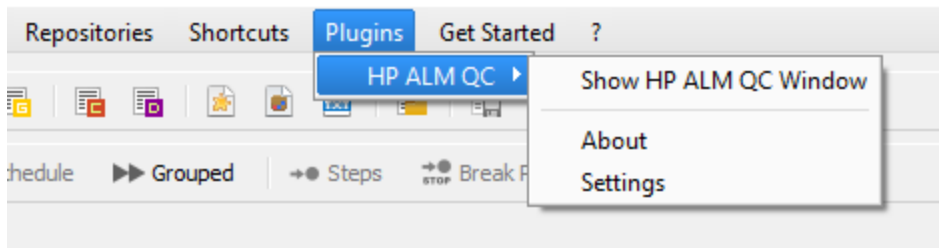
Note: Preferences are stored in the settings.ini file.

4.5 Complements

It is possible to add plugins in the client. Plugins are to be added to the Plugins directory.



Plugins are accessible in the Plugins menu after restarting the client.



Note: It is necessary to restart the client to take into account the plugins deployed.

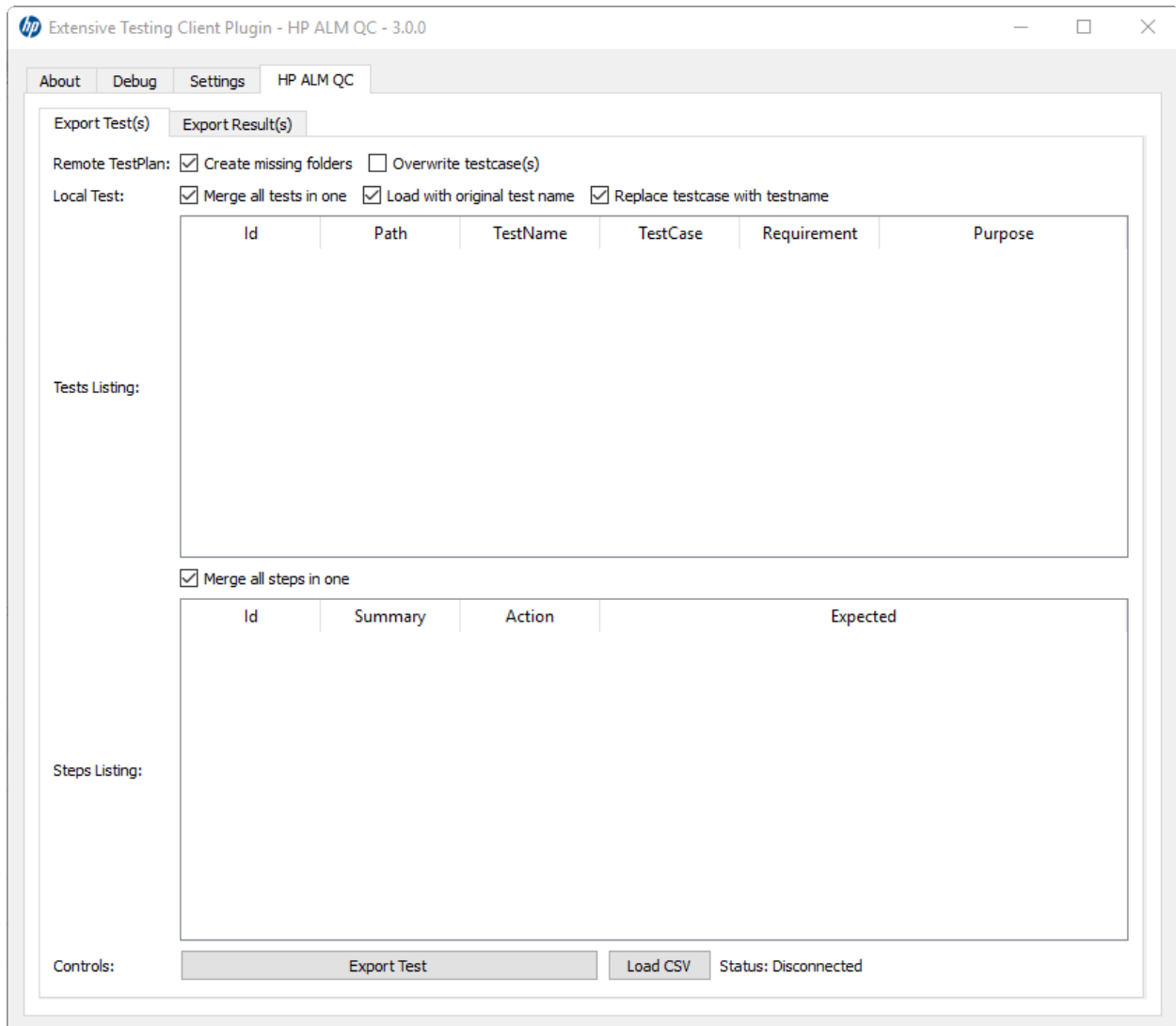
4.5.1 HP ALM plugin

The HP ALM plugin allows you to export tests and results from the Extensive Client to HP ALM QualityCenter. This approach makes it possible to be independent with respect to QC.

The configuration of the plugin is done in the page “Settings“, it is necessary to configure at least:

- username
- the password
- the domain
- the project

To export a test, you must generate the test design from the client and click on the HP ALM plugin available on the toolbar.



The export of the results can be done from the archive exploration window, The plugin must be available in the toolbar when a test report is loaded.

Note: The plugin is compatible with an HP ALM QC > = 12, the REST API is used.

4.5.2 Jenkins plugin

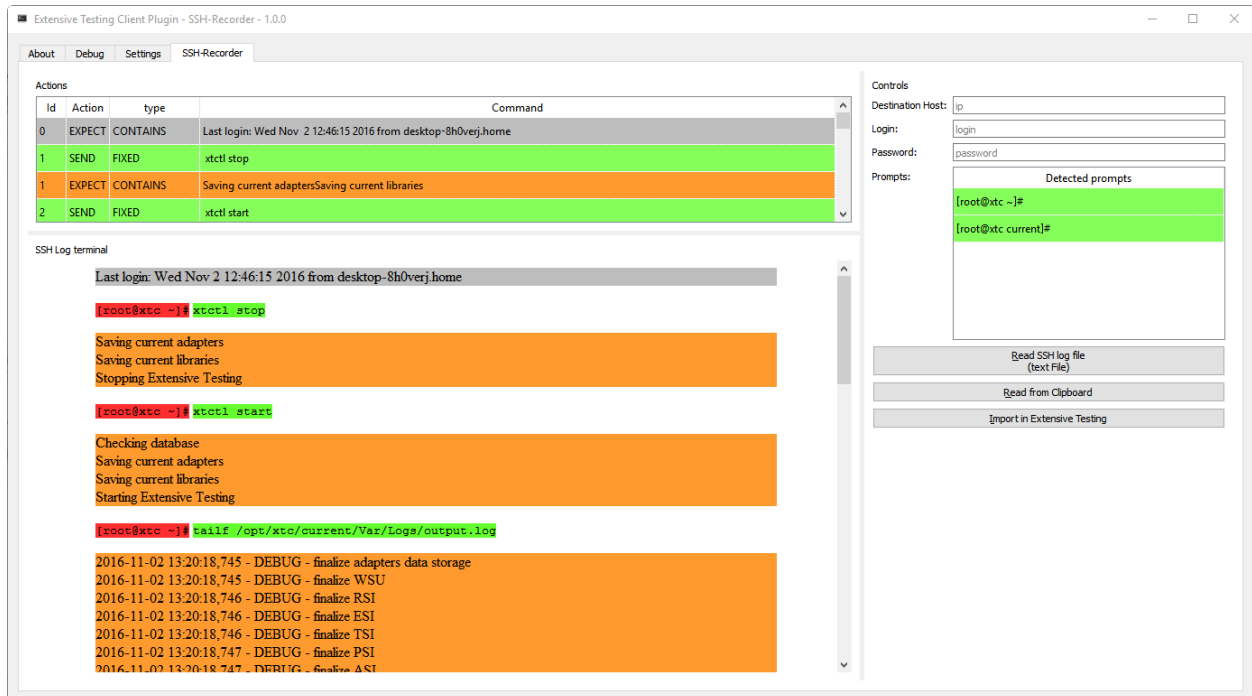
The Jenkins plugin does not do much in this version ... It just provides a link to the web interface of its favorite Jenkins.

4.5.3 Shell Recorder Plugin

The Shell Recorder plugin allows you to import a sequence of shell commands into the design wizard and generate the associated test. It allows to replay easily a sequence of commands.

The first step is to import an ssh session (from a putty terminal for example) from the clipboard or by directly importing a text file containing the sequence of shell commands.

The plugin automatically detects the prompt in the sequence to parse the associated commands and results. If the prompt is not detected, it can be changed manually.



4.5.4 SeleniumIDE Plugin

The use of the SeleniumIDE plugin involves basic use. It can convert a file saved with the SeleniumIDE plugin of firefox in the design assistant.

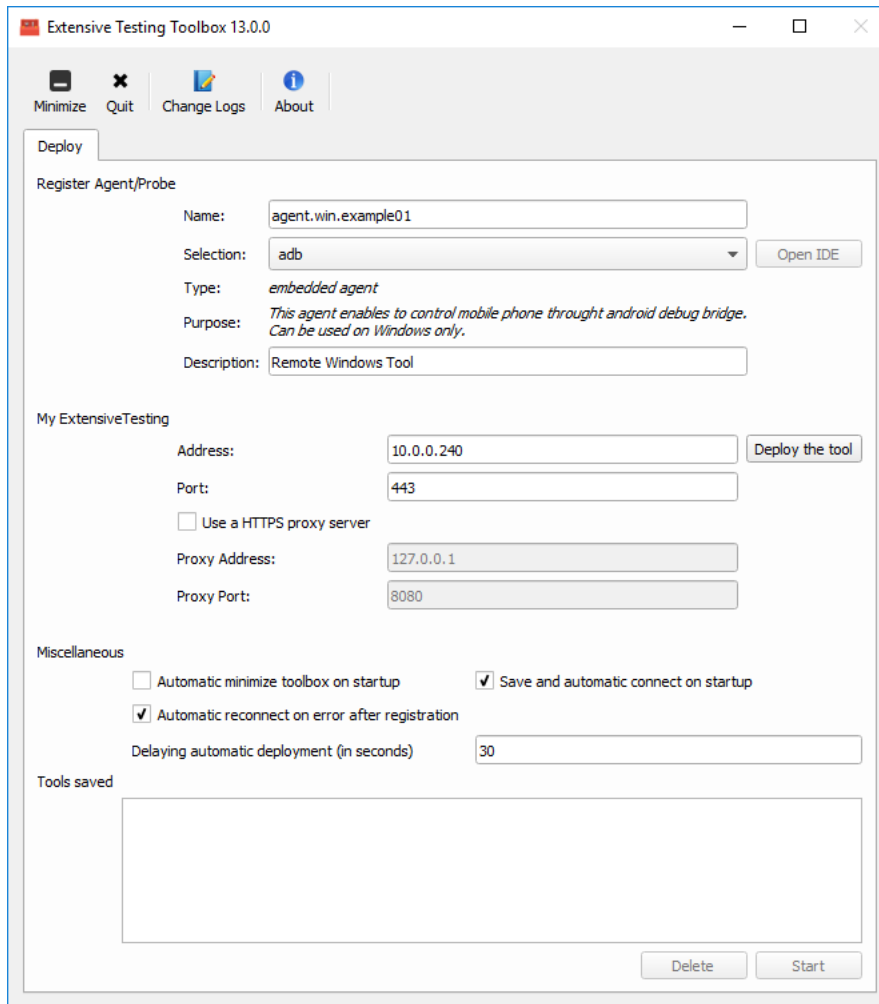
Tip: It is more efficient to use the live assistant to be in tune with the philosophy of the solution.

CHAPTER 5

Toolbox

The toolbox allows you to start agents or probes on dedicated workstations.

- Agents are required to run tests with Selenium on dedicated workstations or to deport the execution of a test.
- Probes can be used to retrieve logs automatically during the execution of a test.



5.1 Deployment

This window allows you to choose the agent or probe to start. The type of agent or probe to start can be chosen in the drop-down list. Finally, an agent or probe needs to be registered with the test server in order to use it.

An agent will allow you to perform a distributed run of your tests. For example, an agent deployed on several machines will allow to run the same test on different environment to test or pilot.

The complete list of available agents and probes are described in the *Server Add-ons > Agents or Probes* chapter.

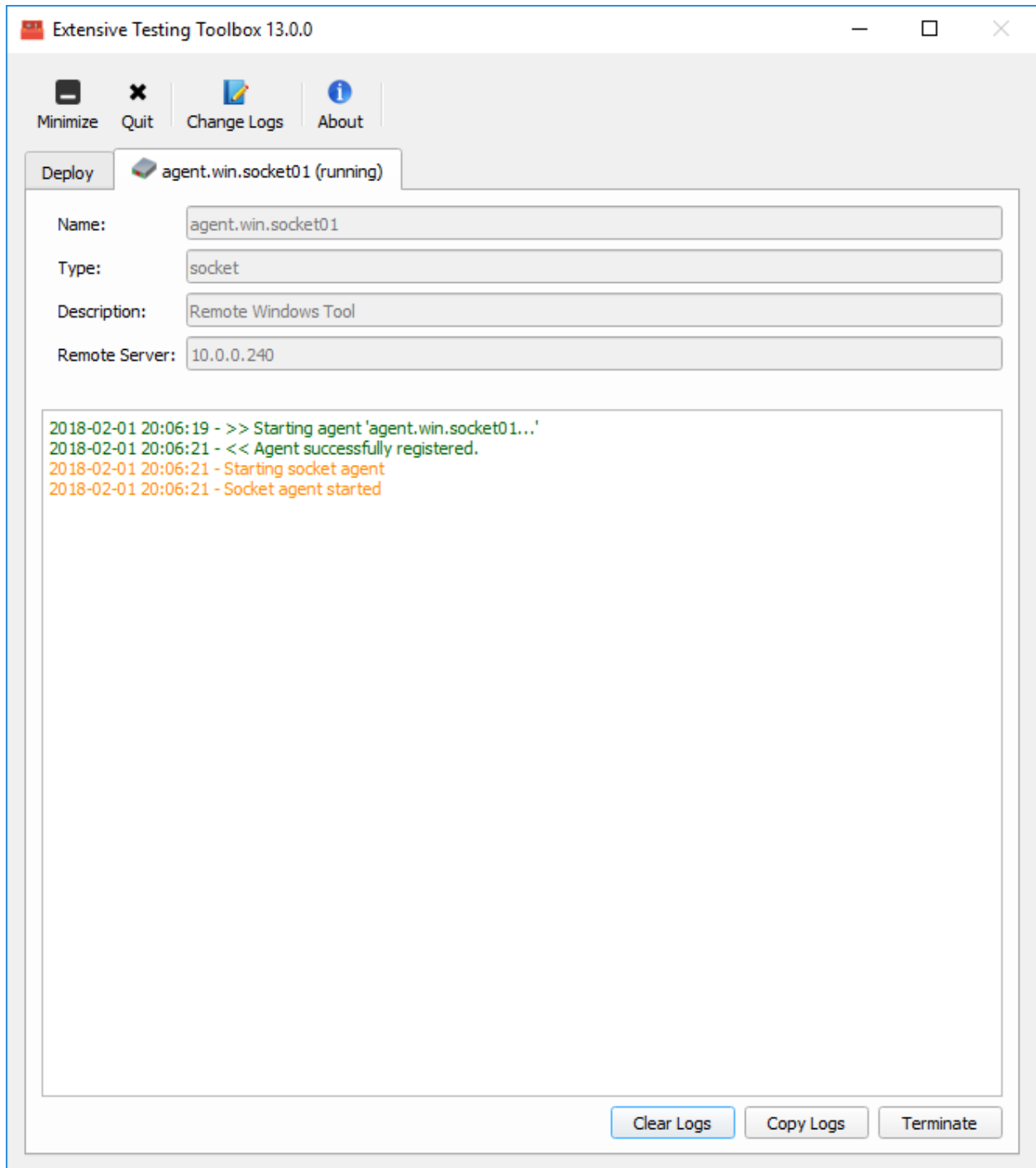
Note: The agent name or probe must be unique for successful registration.

Tip: For a better visibility of the agents or probes available, it is advisable to respect the following formalism for the names:

[Agent | probe] [Environment] [prénom_testeur] [name] [instance-number]...

Example: agent.win.denis.socket01

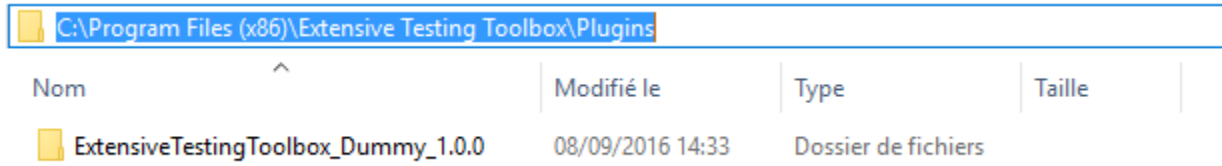
Example of a deployed and running agent:




5.2 More

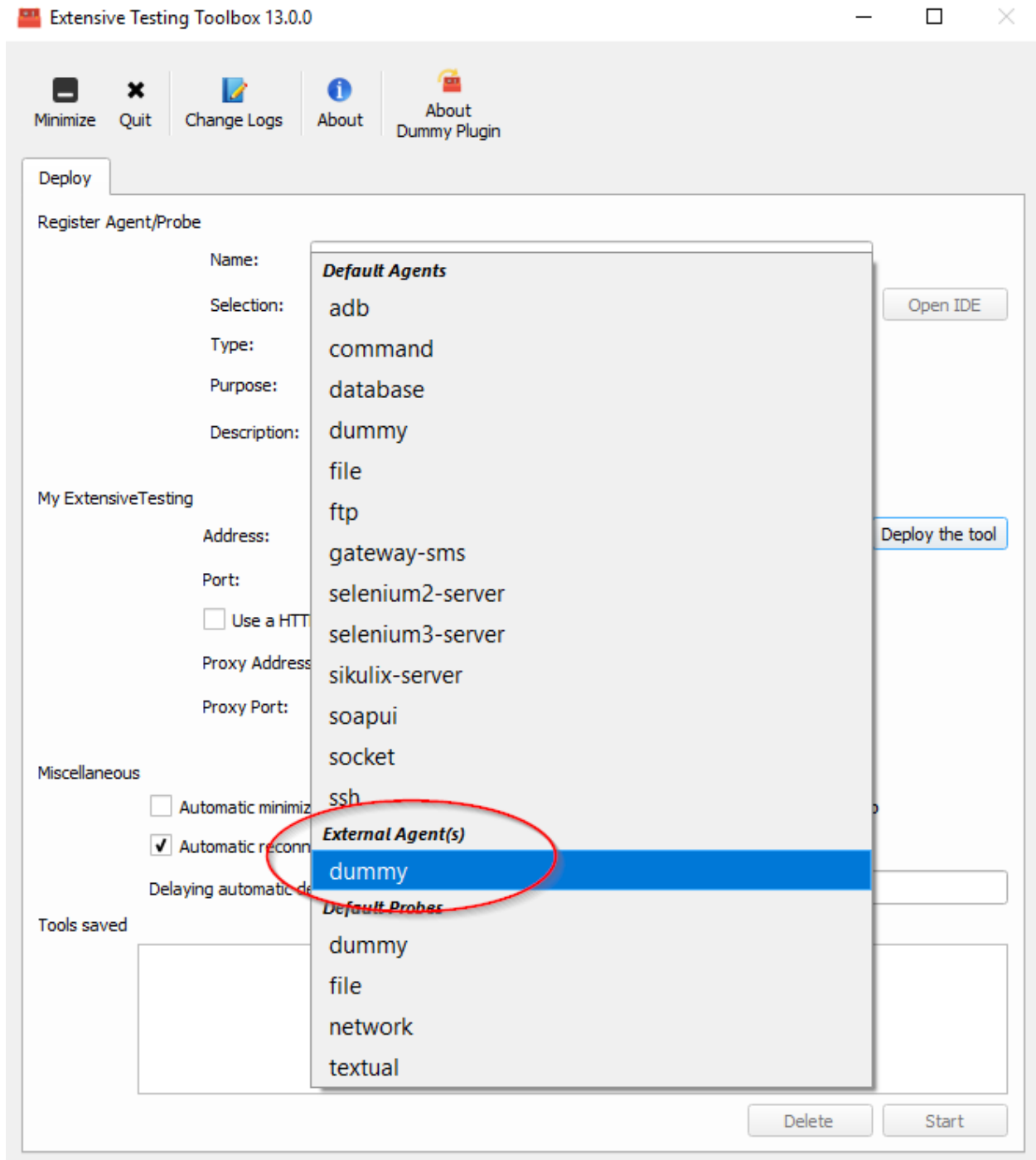
The toolbox can be enriched with new plugins.

To do this, follow the procedure described in the chapter *Contributions > Development plugins > Toolboxes*. Plugins are to be dropped into the Plugins directory.



C:\Program Files (x86)\Extensive Testing Toolbox\Plugins				
Nom	Modifié le	Type	Taille	
 ExtensiveTestingToolbox_Dummy_1.0.0	08/09/2016 14:33	Dossier de fichiers		

After restarting the toolbox, the add-in appears in the list of “external” agents



6.1 Tests part

6.1.1 Global variables

The shared variables enable to describe your dataset. JSON format must be used. These variables are reachable from all tests.

6.2 Admin part

6.2.1 Users

Users account must be created to use properly the product. The creation of users can be done through the web interface or from the rest api.

Some parameters must be provided:

- username
- password
- privileges (admin, monitor, tester)
- authorized projects

Note: Tests results can be received by email if the account is configured with an email.

6.2.2 Projects

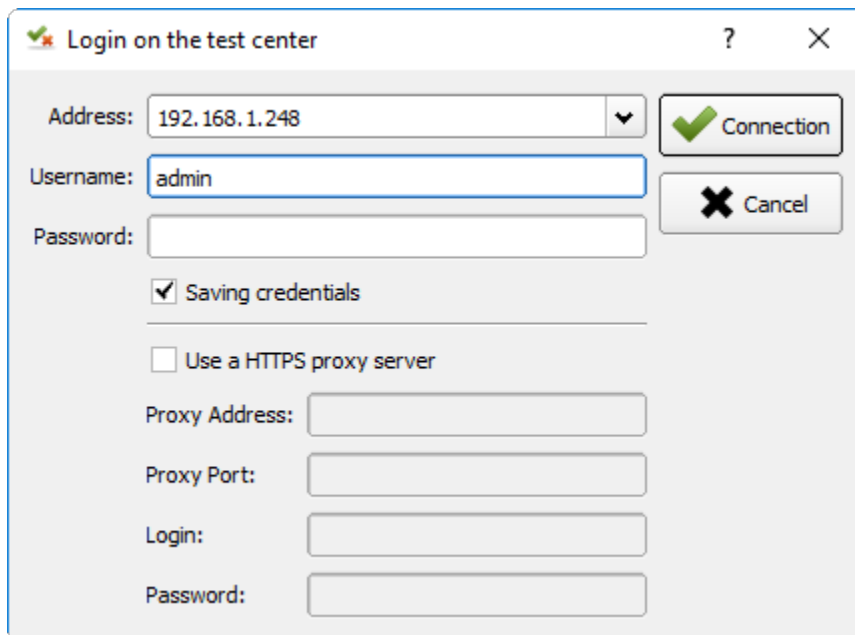
Tests files can be organized per project. The adding or removing of a project can be done from the web interface or the REST api.

Note: The Common project exists by default and can be read from all users, this project cannot be removed.

7.1 Connection to the server

After the opening of the client, the first step is to connect to the remote server. To do that, you need an account and the remote address of your automation server.

The connection window is available from the Get Started > Connect menu or from the welcome tabulation. If the connection is successful, the user can see all remote tests available.



The screenshot shows a dialog box titled "Login on the test center" with a standard Windows-style title bar (minimize, maximize, close buttons). The dialog contains the following fields and controls:

- Address:** A text field containing "192.168.1.248" with a dropdown arrow on the right.
- Username:** A text field containing "admin".
- Password:** An empty password field.
- Buttons:** On the right side, there are two buttons: "Connection" (with a green checkmark icon) and "Cancel" (with a red X icon).
- Checkboxes:**
 - ☒ Saving credentials
 - ☐ Use a HTTPS proxy server
- Proxy Fields:** Below the checkboxes, there are three empty text fields labeled "Proxy Address:", "Proxy Port:", and "Login:".
- Password Field:** Below the "Login:" field, there is an empty password field labeled "Password:".

Note: The admin user can be used in the discover of the solution.

7.2 Write a test

The first utilization consists to create a very simple testcase and display a parameter.

1. Create a test of the type Unit



2. Add the parameter MON_PARAMETRE of the type str with the value “bonjour”

Inputs (4)				Outputs (1)		Actions	
Id	Name	Type	Value				
1	DEBUG	bool	False				
2	MON PARAMETRE	str	bonjour				
3	TIMEOUT	float	10.0				
4	VERBOSE	bool	True				

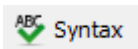
3. Change the test in the definition section to display the value of the parameter.

```
def definition(self):
    # starting initial step
    if self.step1.isEnabled():
        self.step1.start()

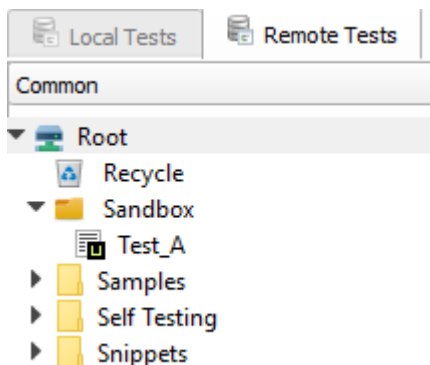
    Trace(self).info(txt=input('MON PARAMETRE '), bold=False, italic=False, multiline=False, raw=False)

    self.step1.setPassed(actual="success")
```

Note: It is possible to check the syntax of the test before execution by clicking on the button Syntax.



4. Save the test in the repository with the name “Test_A” in the Sandbox directory



7.3 Write a scenario

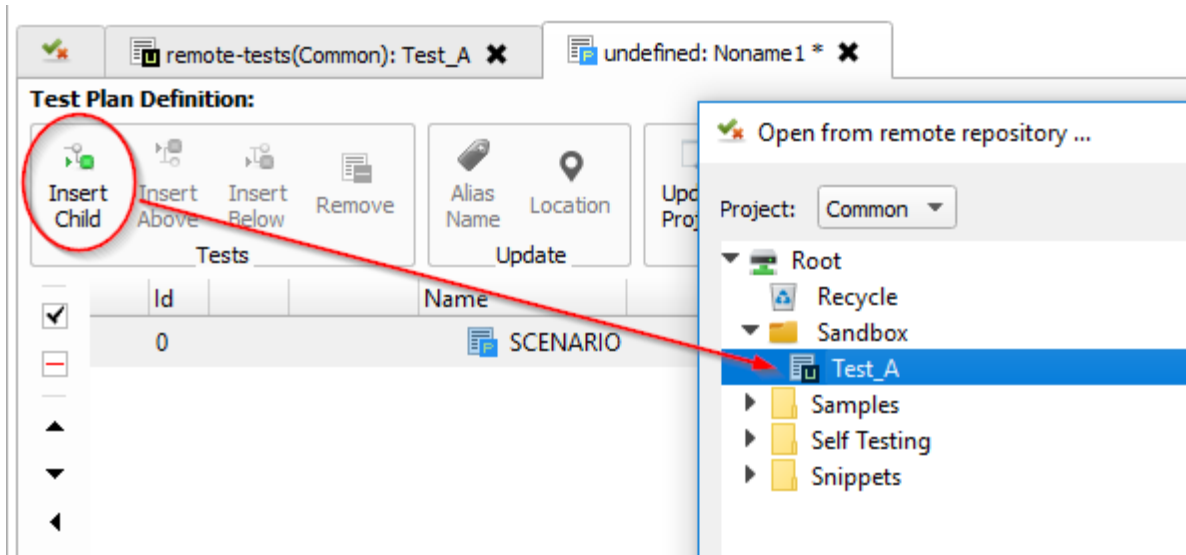
Note: This mini guide assumes that you have followed the chapter Writing a script test.

The following example explains how to create its first scenario with an overload of test variables.

1. Create a Plan type test.



2. Insert test “Test_A” in the scenario. Click on the Insert Child button and select the *Test_A* test.

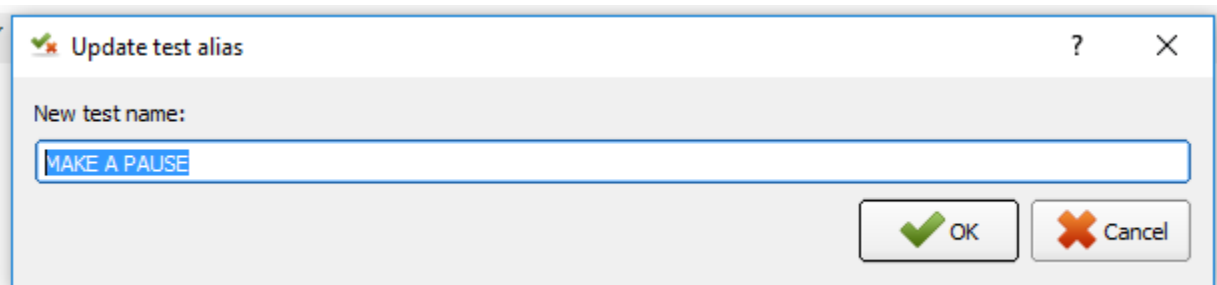


3. After insertion, click on the *Test_A* test and insert the same test again.

Id	Name	Description
0	SCENARIO (2 actives / 2 tests)	
2	IF RUN OF (Common:/@Sandbox/Test_A) IS PASSED THEN	
3	RUN OF (Common:/@Sandbox/Test_A)	

4. Save the scenario in the test repository with the name “Scenario_A” in the Sandbox directory.
5. Add the parameter MON_PARAMETRE with the value “goodbye” at the scenario level.

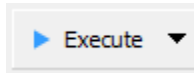
Tip: Do not hesitate to define an alias for the name of the test to make the scenario more readable.



7.4 Execute a test

Note: This mini guide assumes that you have followed the chapters *Writing a script test* and *Writing a scenario*.

You can run a test by clicking the Execute button. Open the *Test_A* and *Scenario_A* tests and run them.



7.5 Result analysing

Note: This mini guide assumes that you have followed the chapters *Writing a script test* and *Writing a scenario*.

The first analysis window shows the execution of the test “Test_A” and in particular the message “hello”.

Extensive Testing Client 18.0.0 - [remote-tests(Common)://@Sandbox/Test_A.tux]

File View Test Editor Test Execution Test Logging Test Conception Scheduler Repositories Shortcuts Plugins Get Started ?

Replay Kill Comment Controls Verdict Report Design Statistics Exports

Summary Options Pause BreakPoint Interact

			NB	MIN	AVG	MAX
OK	1	TESTCASE	1	0.002	0.002	0.002
KO	0	TESTABSTRACT	0	0.000	0.000	0.000
UNDEFINED	0	TESTUNIT	1	0.286	0.286	0.286
		TESTSUITE	0	0.000	0.000	0.000
		TESTPLAN	0	0.000	0.000	0.000
TOTAL	1	TESTGLOBAL	0	0.000	0.000	0.000

Events Filter

Pattern: ^(?IDEBUG)

Syntax: Regular expression

Column: Event Type

☐ Case sensitive filter

Next

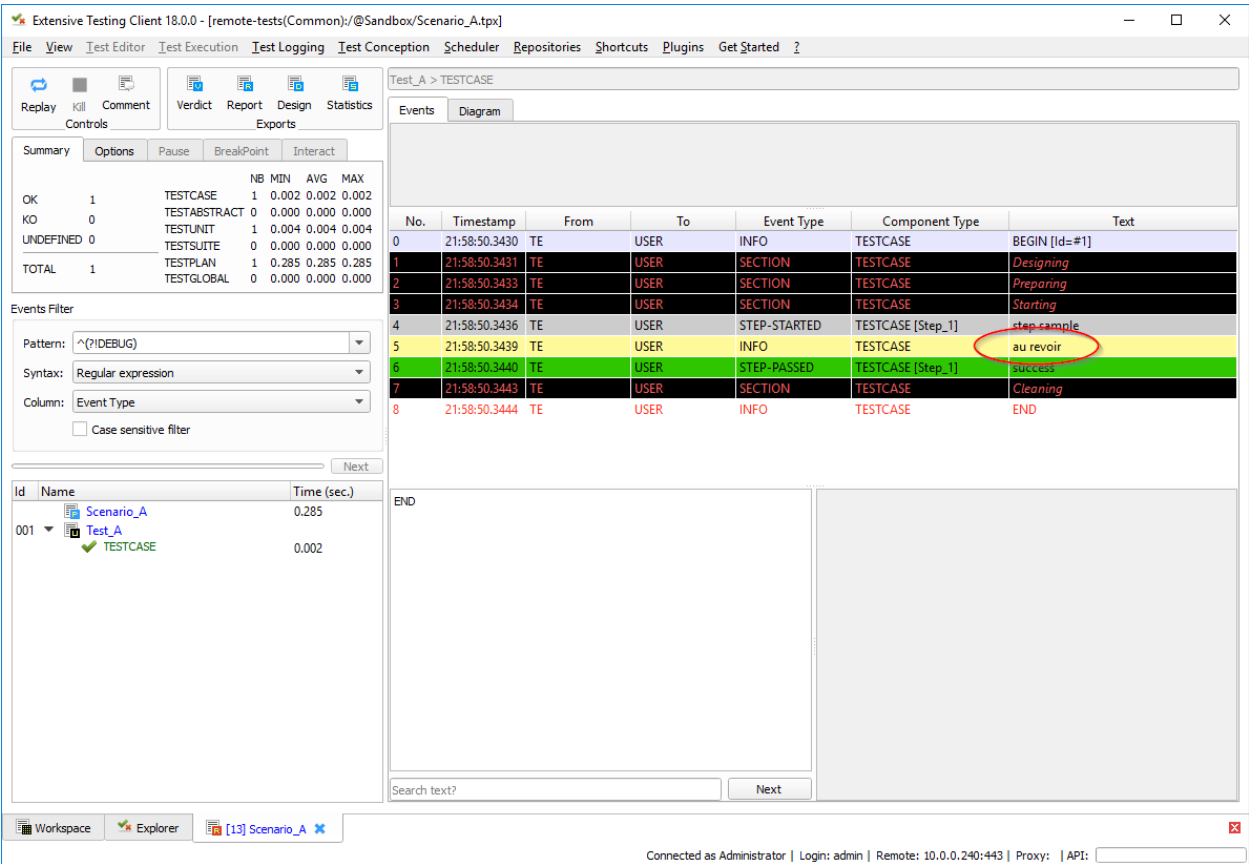
No.	Timestamp	From	To	Event Type	Component Type	Text
0	21:57:06.4141	TE	USER	INFO	TESTCASE	BEGIN [Id=#1]
1	21:57:06.4142	TE	USER	SECTION	TESTCASE	Designing
2	21:57:06.4144	TE	USER	SECTION	TESTCASE	Preparing
3	21:57:06.4145	TE	USER	SECTION	TESTCASE	Starting
4	21:57:06.4147	TE	USER	STEP-STARTED	TESTCASE [Step_1]	step sample
5	21:57:06.4149	TE	USER	INFO	TESTCASE	bonjour
6	21:57:06.4152	TE	USER	STEP-PASSED	TESTCASE [Step_1]	success
7	21:57:06.4155	TE	USER	SECTION	TESTCASE	Cleaning
8	21:57:06.4156	TE	USER	INFO	TESTCASE	END

Search text? Next

Workspace Explorer [11] Test_A

Connected as Administrator | Login: admin | Remote: 10.0.0.240:443 | Proxy: | API:

The 2nd analysis window shows the execution of the “Scenario_A” test and in particular the “goodbye” message.



This first usage shows how to run a test and a scenario as well as the overloading of the test variables.

7.6 Best practices

Tip: To keep readability in script type tests, do not use *try / except*. The framework catches all the exceptions at its level.

Tip:

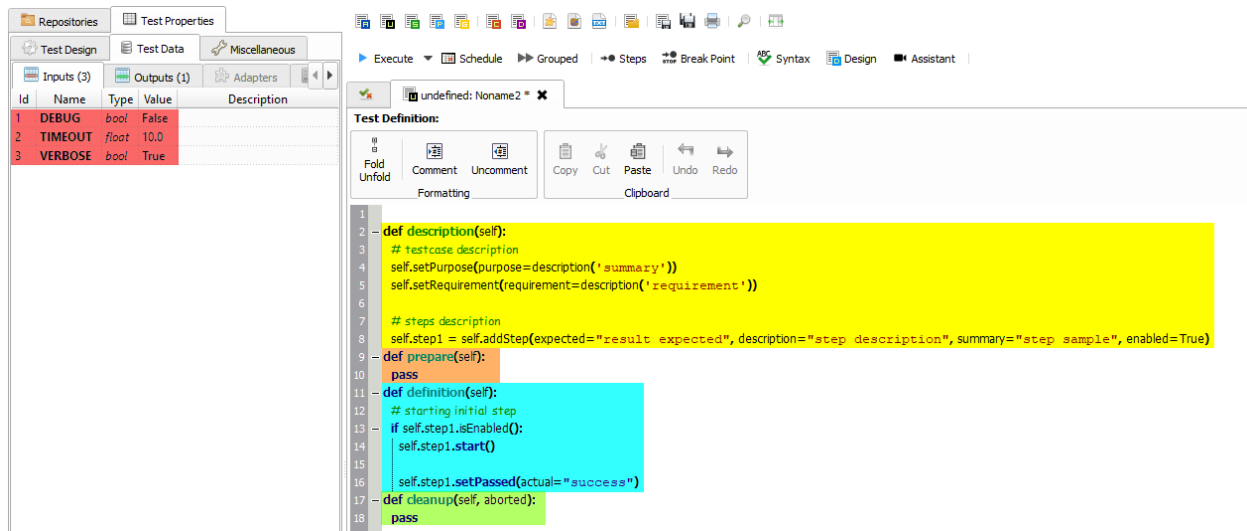
It is essential to take the time to declare the test steps because they allow

- quickly understand the test without the script.
- to have relevant and understandable test reports.

Tip: To facilitate the maintenance of your tests and make them reusable, you should not have hard value in your test. It is necessary to systematically put them in test parameters, it is done for.

8.1 Testcase (unit)

This example shows how to use a test case. A test case consists of 4 sections automatically executed by the test framework as well as associated test parameters.



8.2 Testcase (suite)

A test suite allows you to run several test cases afterwards. The example shows how to loop on a test case while modifying the incoming data.

The screenshot shows the ExtensiveAutomation interface. On the left, the 'Test Design' tab displays a table of inputs:

Id	Name	Type	Value	Description
1	DEBUG	bool	False	
2	LISTE_VALEURS	list	1 2 5 0 4	
3	TIMEOUT	float	10.0	
4	VERBOSE	bool	True	

The 'Test Definition' tab shows the Python code for the test case:

```

1 class TESTCASE_01(TestCase):
2     def description(self, valeur):
3     def prepare(self, valeur):
4     def definition(self, valeur):
5         # starting initial step
6         if self.step1.isEnabled():
7             self.step1.start()
8             Trace(self).info(txt=valeur, bold=False, italic=False, multiline=False, raw=False)
9             self.step1.setPassed(actual="success")
10    def cleanup(self, aborted, valeur):
11
12 Test Execution:
13 for v in input('LISTE_VALEURS'):
14     TESTCASE_01(suffix=None).execute(valeur=v)
  
```

Red arrows indicate the mapping from the input table to the code arguments:

- The 'DEBUG' input (Id 1) maps to the 'valeur' argument in the 'description' method (line 2).
- The 'LISTE_VALEURS' input (Id 2) maps to the 'valeur' argument in the 'prepare' method (line 3), the 'valeur' argument in the 'definition' method (line 4), and the 'valeur' argument in the 'cleanup' method (line 10).
- The 'TIMEOUT' input (Id 3) maps to the 'valeur' argument in the 'definition' method (line 4).
- The 'VERBOSE' input (Id 4) maps to the 'valeur' argument in the 'cleanup' method (line 10).

It is therefore possible to add as many arguments as necessary to the `execute()` function and add them identically to the level of the 4 sections.

Note: It is possible to add a prefix at the test case level using the `prefix` argument.

8.3 variables

The variables can be used from a test, there are several types. The example below shows how to retrieve a parameter from its test.

A test parameter can be retrieved at the test level using the `input` function. The name of the parameter to be recovered is to be specified.

The screenshot shows the 'Test Design' tab with a table of inputs and a 'Test Definition' code editor.

Id	Name	Type	Value	Description
1	DEBUG	bool	False	
2	TIMEOUT	float	10.0	
3	VERBOSE	bool	True	

The 'Test Definition' code snippet is as follows:

```

1
2 + def description(self):
9 - def prepare(self):
11 - def definition(self):
12     # starting initial step
13     if self.step1.isEnabled():
14         self.step1.start()
15
16         Trace(self.info(txt=input('TIMEOUT'), bold=False, italic=False, multiline=False, raw=False)
17
18         self.step1.setPassed(actual="success")
19 + def cleanup(self, aborted):

```

A red arrow points from the '10.0' value in the 'TIMEOUT' row to the `input('TIMEOUT')` call in the code.

8.4 Scenario

A scenario allows you to run several test cases one after the other with result conditions between them. It is possible to override the test parameters at the scenario level.

The screenshot shows the 'Test Plan Definition' interface for a test plan named 'remote-tests(Common): 005_Users'.

The 'Tests' section includes a table of tests:

Id	Name	Description
0	SCENARIO (12 actives / 12 tests)	
32	GENERATE NEW HASH PASSWORD	
31	IF REST USER ADD IS PASSED THEN	
35	REST USER UPDATE	
37	REST USER STATUS	
34	REST USER LISTING	
36	REST USER PROFILE	
42	REST USER CHANGE PASSWORD	
43	REST USER RESET PASSWORD	
44	REST USER DISCONNECT	
39	REST USER DUPLICATE	
41	REST USER REMOVE DUPLICATE	
33	REST USER REMOVE	

The 'SCENARIO' section shows a list of tests with their status and description:

- 32 GENERATE NEW HASH PASSWORD
- 31 IF REST USER ADD IS PASSED THEN
- 35 REST USER UPDATE
- 37 REST USER STATUS
- 34 REST USER LISTING
- 36 REST USER PROFILE
- 42 REST USER CHANGE PASSWORD
- 43 REST USER RESET PASSWORD
- 44 REST USER DISCONNECT
- 39 REST USER DUPLICATE
- 41 REST USER REMOVE DUPLICATE
- 33 REST USER REMOVE

8.5 Test campaign

A campaign allows you to run multiple scenarios. It is possible to overload the test parameters at the campaign settings level.

The screenshot shows the 'Test Global Definition' window for a campaign named 'remote-tests(Common): 0001_Rest_Api'. The interface includes a toolbar with various actions like 'Insert Child', 'Insert Above', 'Insert Below', 'Remove', 'Update', 'Replace Locations', 'Default Aliases', 'Open All', 'Reload', 'Merge', 'Clear', 'Clip', and 'Misc.'. Below the toolbar is a table listing the tests in the campaign.

	Id	Name	Description
0		GLOBAL SCENARIO (5 actives / 11 tests)	
12	RUN OF (PREPARE TEST)
21	RUN OF (LOGIN TO THE REST API)
1	RUN OF (SESSION REST RESSOURCES)
6	RUN OF (PROJECT REST RESSOURCES)
7	DONT RUN OF (USER REST RESSOURCES)
8	DONT RUN OF (TEST REST RESSOURCES)
16	DONT RUN OF (VARIABLES REST RESSOURCES)
11	DONT RUN OF (TASK REST RESSOURCES)
20	DONT RUN OF (PUBLIC REST RESSOURCES)
18	DONT RUN OF (SYSTEM REST RESSOURCES)
15	RUN OF (LOGOUT FROM THE REST API)

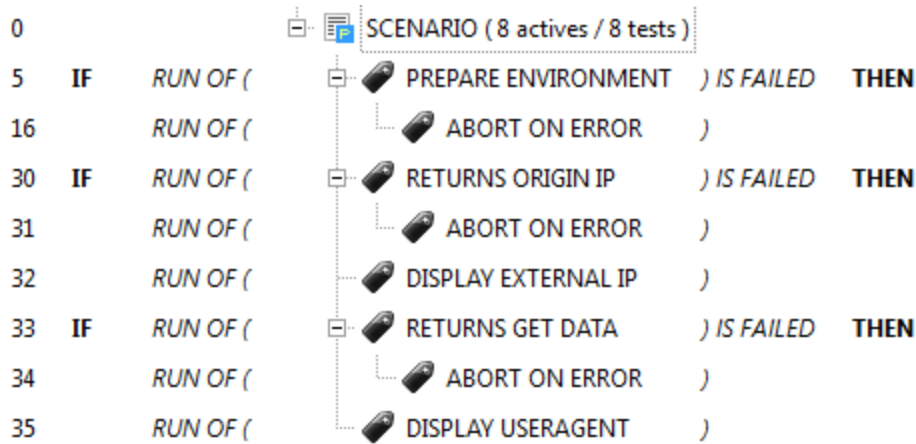
8.6 Rest API

To write a REST API test, it is recommended:

- to use the reusable test /Snippets/Protocols/04_Send_JSON
- describe the target server in JSON (ip / destination port, http support)

Example:

The test calls the `httpbin.org` service in https and calls the `ip` service to get the client's actual ip in json.



The scenario breaks down into several stages:

1. Preparation of the environment: description of the tested environment (address, network port, etc.) The environment is configured in the *ENVIRONMENT* parameter of the *PREPARE ENVIRONMENT* test (Id = 5)

```

{
  "PLATFORM": {
    "CLUSTER": [
      { "NODE": {
        "COMMON": {
          "HOSTNAME": "httpbin"
        },
        "INSTANCES": {
          "HTTP": {
            "REST": {
              "HTTP_DEST_HOST": "httpbin.org",
              "HTTP_DEST_PORT": 443,
              "HTTP_DEST_SSL": true,
              "HTTP_HOSTNAME": "httpbin.org",
              "HTTP_AGENT_SUPPORT": false,
              "HTTP_AGENT": null
            }
          }
        }
      }
    ]
  },
  "DATASET": [ ]
}

```

2. If the environment preparation does not work then the scenario is stopped by calling the test reusable Snippets/Do/02_Terminate (Id = 16)
3. A REST request is sent and the expected response is described using the reusable test /Snippets/Protocols/04_Send_JSON (Id = 30). If this step does not work then we cancel the test (Id = 31)

The response received is verified by the framework and what was described by the tester in the *HTTP_RSP_BODY* parameter

```
origin          [!CAPTURE:EXTERNAL_IP:]
```

The configuration indicates that the response must verify that the `origin` key is present and save the value in the cache with the ``EXTERNAL_IP`` key

4. The value received in the response is displayed with the reusable test ``Snippets/Cache/02_Log_↵Cache`` (Id = 32)

Note: The example presented below is available in full in the test samples /Samples/Web_API/001_httpbin_rest.tpx.

8.7 SSH controls

To write an SSH test, it is advisable:

- to use the reusable test /Snippets/Protocols/01_Send_SSH
- to describe the target server in JSON (ip, account, password at least)

```
0          SCENARIO ( 5 actives / 5 tests )
7 IF RUN OF ( INITIALIZE ENVIRONMENT ) IS FAILED THEN
9 RUN OF ( ABORT PREPARE ON ERROR )
2 RUN OF ( CHECK SERVER STATUS )
3 IF RUN OF ( SHOW SERVER VERSION ) IS PASSED THEN
8 RUN OF ( LOGS CAPTURED INFORMATIONS )
```

The test is broken down into several stages:

1. Loading the description (ip, account, password) of the target machine into the cache
2. Calling the /Snippets/Protocols/01_Send_SSH generic test to retrieve the server version The version (if found on the screen) is saved in the cache with the `SERVER_VERSION` key If the version is not found, the test goes into error.

```
# checking server version
xtctl version
.*Server version: [!CAPTURE:SERVER_VERSION:]\\n.*
```

3. View the version from the cache.

Note: The complete example is available in the test samples /Self Testing/SYSTEM/000_System.tpx.

8.8 Web browsers

To write a web application test, you must:

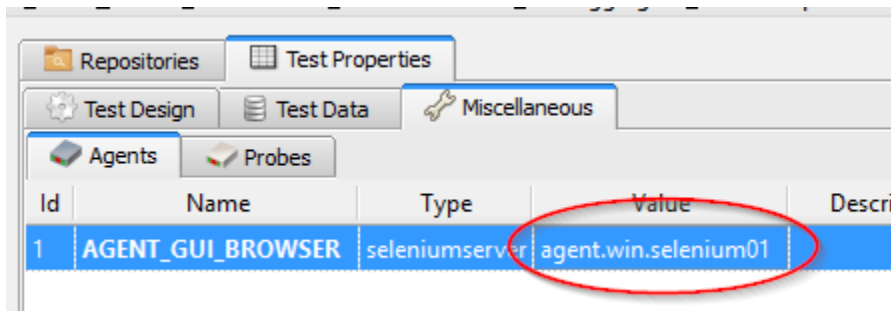
- deploy a selenium agent on a machine with a firefox, chrome, internet explorer or edge browser

- have access to the source code of the web page from his browser
- have knowledge of xpath
- know the basics of HTML

The recommended approach for writing web tests is as follows:

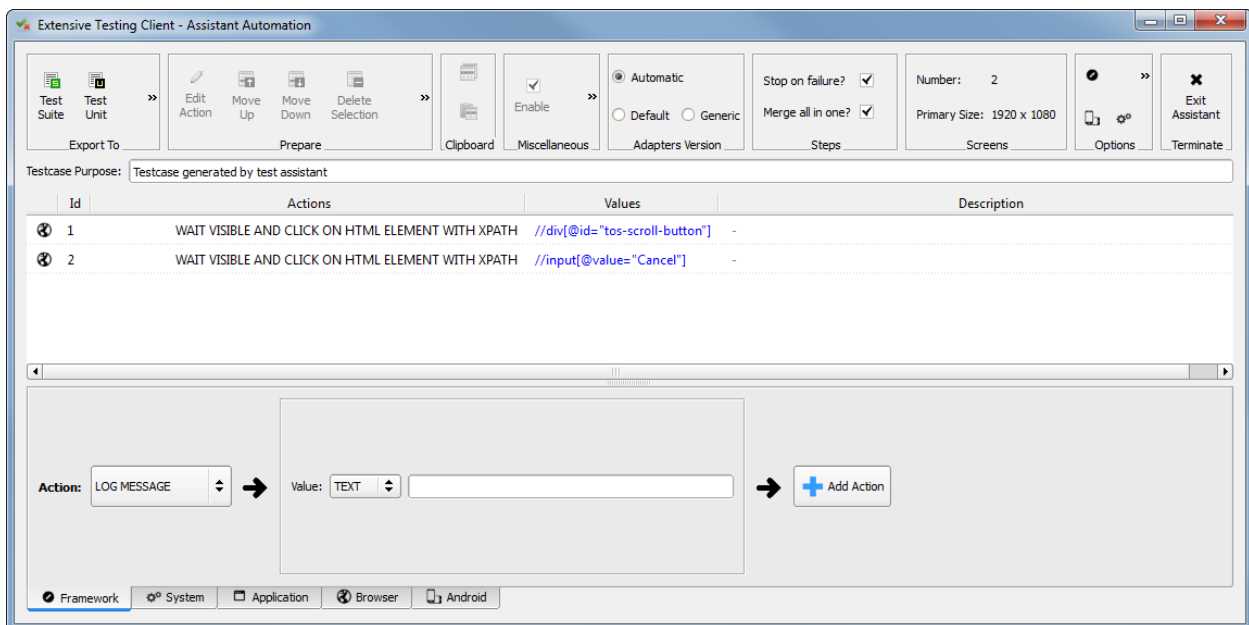
- identify the number of pages displayed to script (and the possible reuse of these pages)
- identify the different sequence of pages to create the scenarios
- identify user paths

To perform this type of test, you must declare the agent that will be used

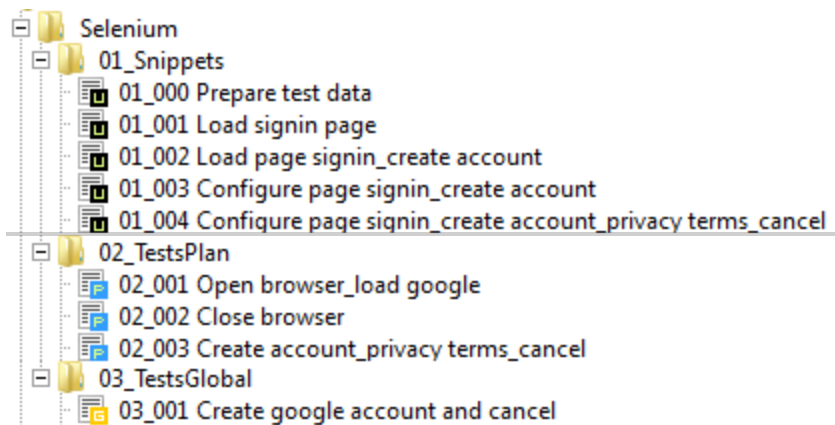


The writing of the tests is done through the assistant. It allows to describe the different stages and generate the equivalent unit test. The sequence of pages are to be described in the flat tests. The user path is to be defined in a global test.

The solution also recommends using only xpath to identify HTML elements.

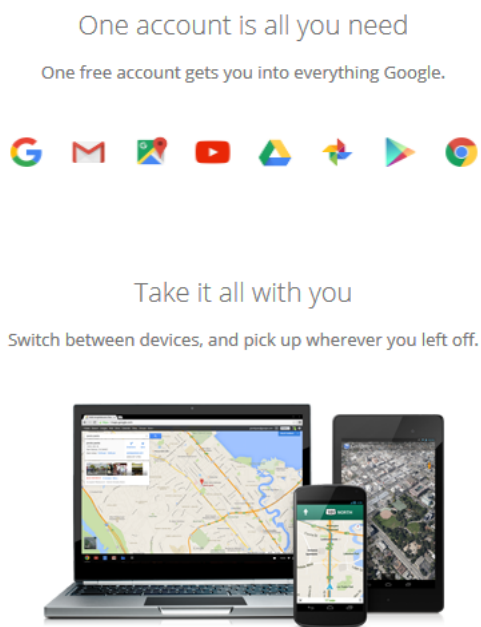


The example below shows how to create a Google Account using a random name and first name.



Example of result:

Create your Google Account

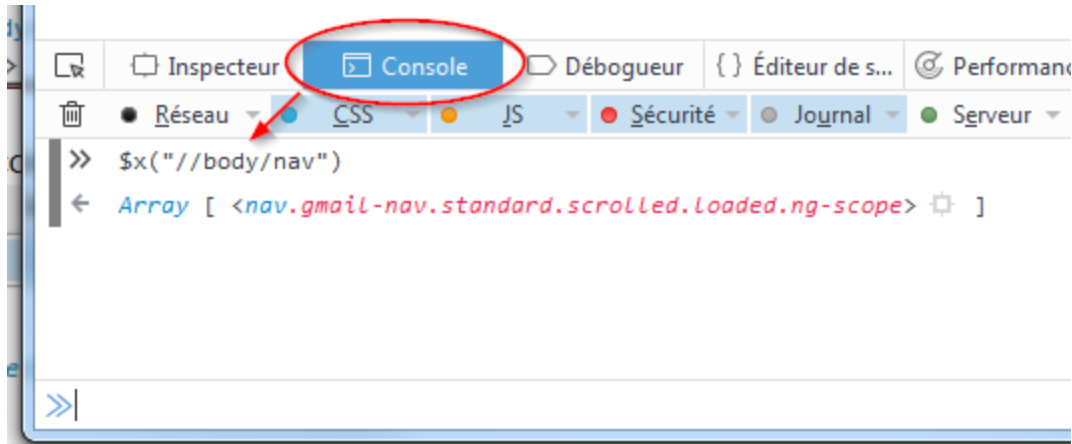


A screenshot of the Google Account creation form. The form includes the following fields:

- Name: Two text boxes containing "GtGUdlyW" and "EzMeJGXj".
- Choose your username: A text box containing "uqEjKSQg" and a dropdown menu showing "@gmail.com".
- I prefer to use my current email address: A link.
- Create a password: A text box with masked characters.
- Confirm your password: A text box with masked characters.
- Birthday: A dropdown menu showing "April", and two text boxes containing "21" and "1974".
- Gender: A dropdown menu showing "Other".
- Mobile phone: A dropdown menu showing a flag icon and "+33", followed by a text box.
- Your current email address: A text box.

A red circle highlights the "Name" and "Choose your username" fields.

Tip: It is possible to use browser development tools to validate xpaths.



Note: The example presented below is available in full in the test samples /Samples/Tests_Gui/Selenium/.

Note: Selenium3 requires at least Java 8 on the client machine.

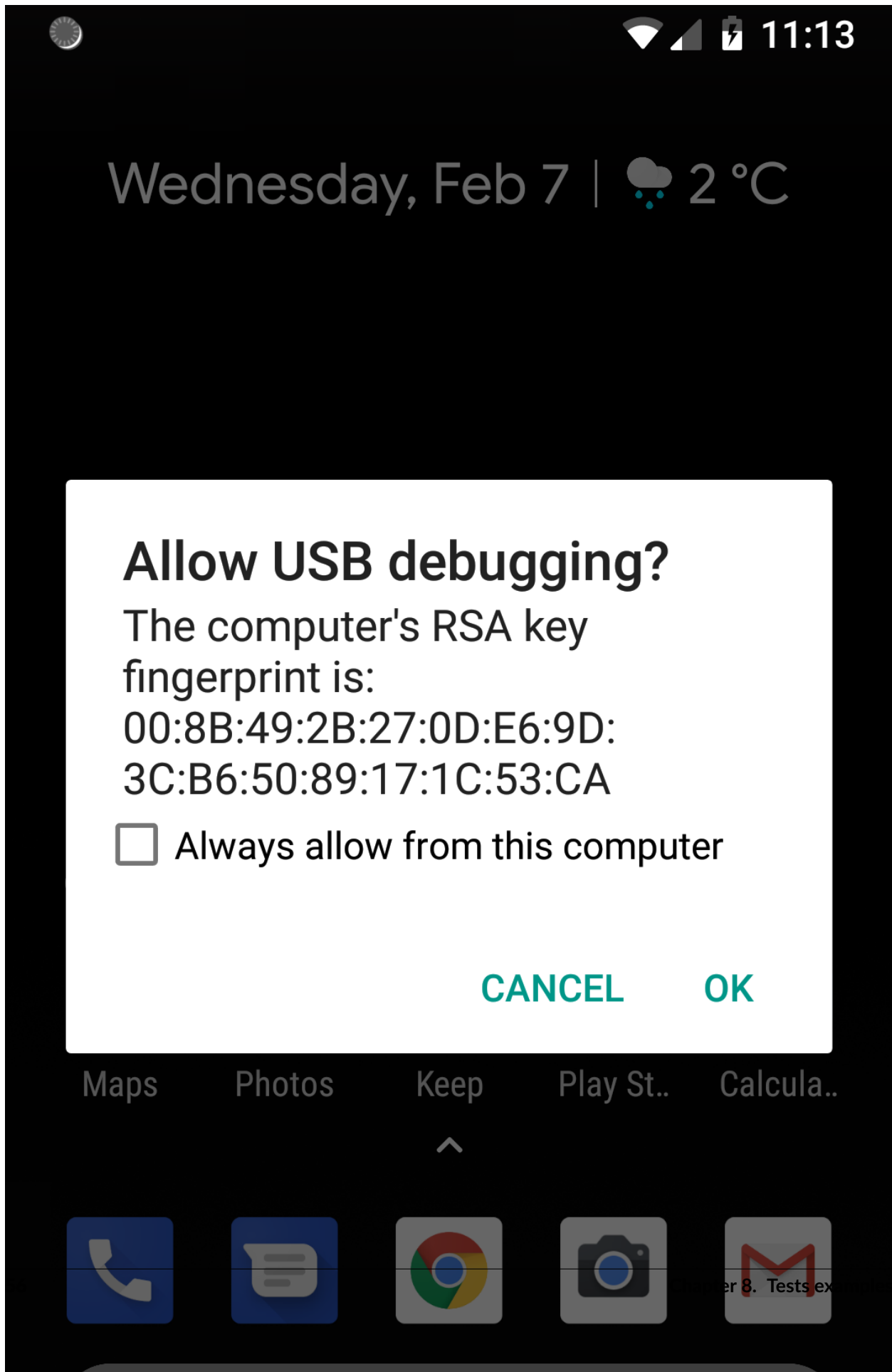
Browsers	Version Selenium	Gecko
Firefox <47	Selenium 2	Non
Firefox > 47	Selenium 3	Oui
IE	Selenium 3	N/A
Chrome	Selenium 3	N/A

8.9 Android mobile

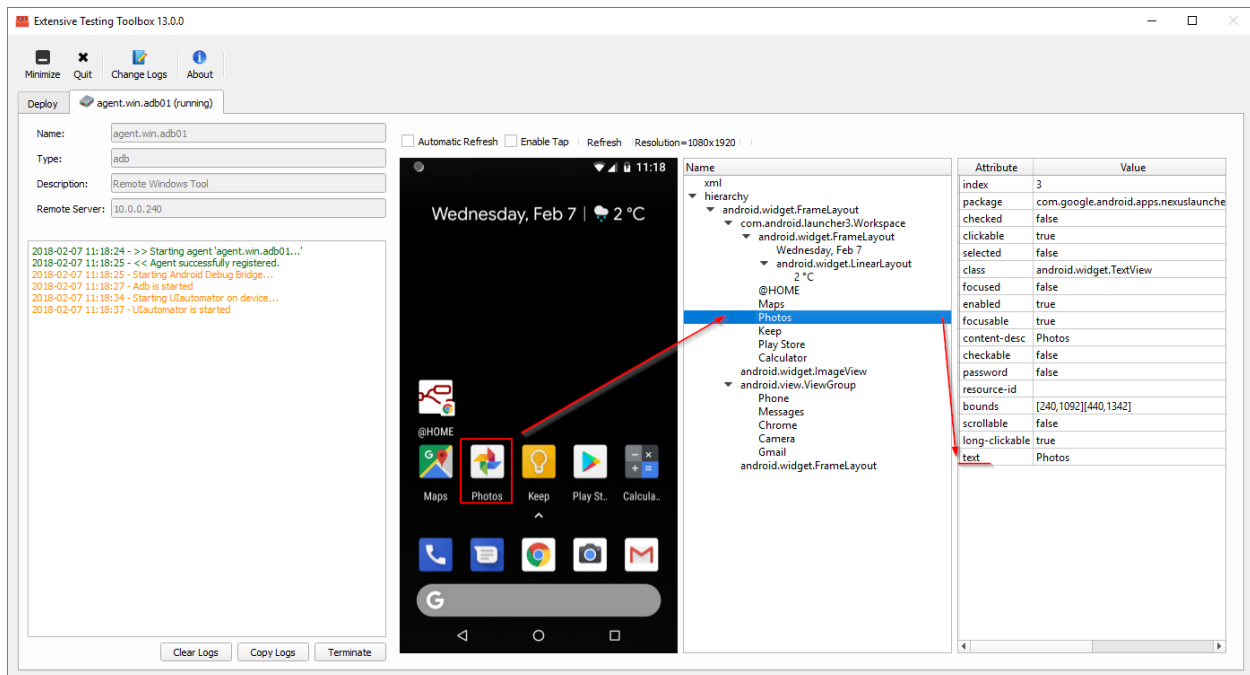
To write the test of a mobile application, you must:

- Have an Android mobile phone connected in USB on a PC
- Deploy an adb agent on a computer with an android mobile connected to it.
- Have access to the xml description of applications from the agent

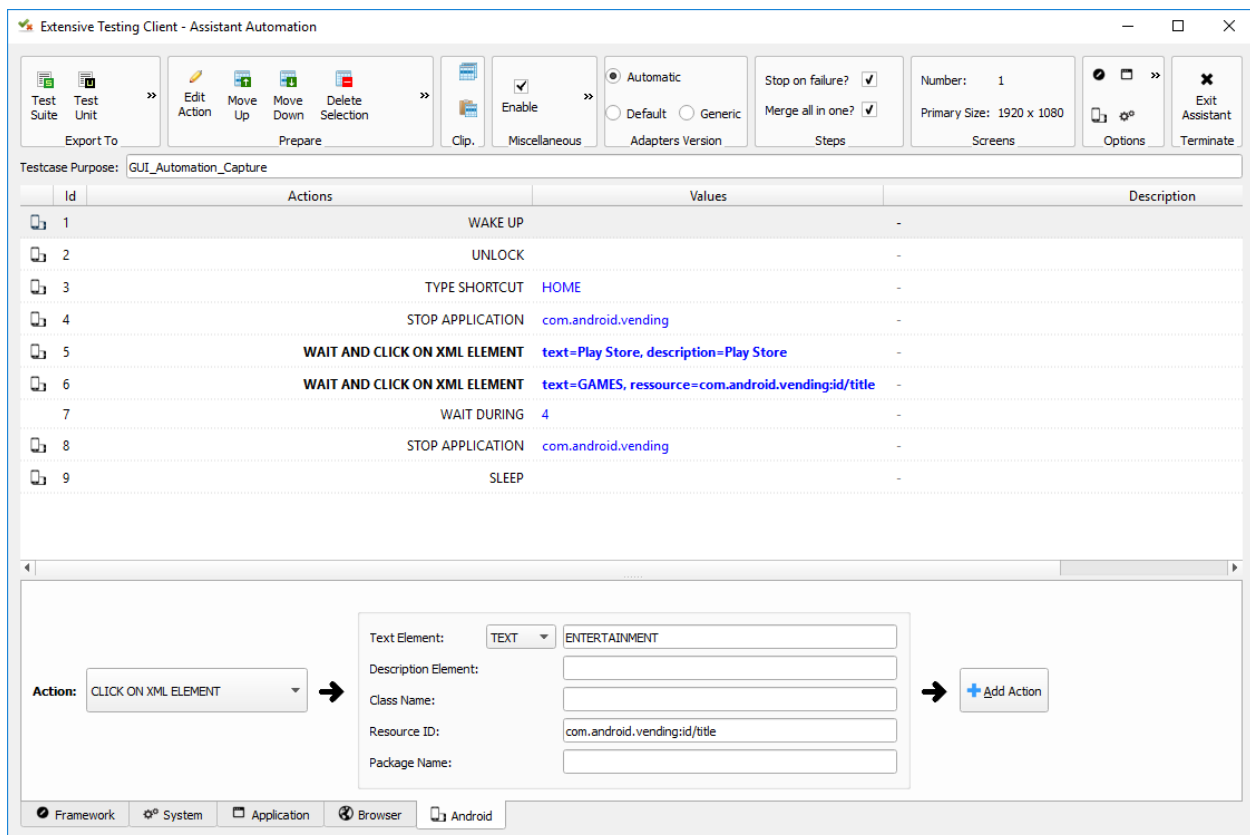
Connecting the adb agent on the android mobile requires accepting the RSA key.



After login, the agent displays a preview of the screen on the pc, it is possible to browse the interface from the agent and have the XML elements available in the page.



The writing of the tests is done with the assistant. It allows to describe the different stages and generate the equivalent unit test. It is essential to rely on the adb agent for have the list of available XML elements and attributes.



Note: The complete example is available in test samples `/Samples/Tests_Mobiles/03_PlayStore.tux`.

Important: Enabling USB *debug* mode is required on the phone.

The interest of reusable tests

- factorize the test database
- reuse the tests
- limit scripting to design scenarios

These types of tests are to be used in test plan mode.

9.1 Shared data between tests

9.1.1 Add in the cache a new data

Important: path of the reusable test /Snippets/Cache/01_Set_Cache.tux

This reusable test consists of saving a value in the data cache during the execution of a test.

Parameter(s) to configure:

The DATAS parameter contains the list of values to save with the format:

```
# my comment  
[!TO:CACHE:<MA_CLE>:];my value
```

Example

```
# Save misc data  
[!TO:CACHE:EXAMPLE:];hello world  
  
# Save server information in the cache  
[!TO:CACHE:SERVER_DESCRIPTION:];[!FROM:INPUT:TEST_PURPOSE:]
```

Note: It is possible to save several values with this test.

9.1.2 Display a value from the cache

Important: path of the reusable test /Snippets/Cache/02_Log_Cache.tux

This reusable test makes it possible to display the value of a key present in the cache during the execution of the test.

Parameter(s) to configure:

Parameters	Description
MESSAGES	Contains the list of parameters to log in the test

```
# display cache
[!FROM: CACHE:EXAMPLE:]

# log timeout input
[!FROM: INPUT:TIMEOUT:]
```

Note: It is possible to display multiple values at one time

9.1.3 Reset the cache

Important: path of the reusable test /Snippets/Cache/02_Reset_Cache.tux

This reusable test makes it possible to totally empty the cache. No parameters to configure.

Note: This test can be used when several scenarios are chained together in a global test.

9.1.4 Checking a value in the cache

Important: path of the reusable test /Snippets/Cache/02_Checking_Cache.tux

This reusable test makes it possible to check the value in a key present in the cache.

Parameter(s) to configure:

Parameters	Description
CHECKING	List of values to check in the cache

Operators available:

Parameters	Description
contains	Check if the value contains a string
matches	Check if the value matches the regular expression
==	Check if the value equals
!=	Check if the value is different from
>	Check if the value is greater than
<	Check if the value is less than
>=	Check if the value is greater than
<=	Check if the value is less than

```
# Check if value contains the test string
[!FROM:CACHE:EXAMPLE:] contains test
```

Note: It is possible to check multiple values at one time

9.1.5 Delete entry from the cache

Important: path of the reusable test /Snippets/Cache/02_Delete_Cache.tux

This reusable test is used to delete a entry in the cache according to the key.

Parameter(s) to configure:

Parameters	Description
MESSAGES	List of keys to delete

```
# delete the key EXEMPLE from the cache
[!FROM:CACHE:EXAMPLE:]
```

Note: It is possible to delete several keys at one time

9.2 Basics actions

9.2.1 Hold on a test

Important: path of the reusable test /Snippets/Do/01_Wait.tux

This reusable test allows you to wait for xx seconds while the test runs.

Parameter(s) to configure:

Parameters	Description
DURATION	duration in seconds

9.2.2 Stop a test

Important: path of the reusable test /Snippets/Do/02_Terminate.tux

This reusable test makes it possible to force the stopping of a scenario on error occurrences.

Note: It is possible to customize the stop message by setting the variable STOP_TEST_MSG.

9.2.3 Load test environment

Important: path of the reusable test /Snippets/Do/03_Initilize.tux

This reusable test is used to load the test environment data into the cache (ip addresses, server access account, etc.).

An environment is described with 4 levels:

- environment
- cluster
- node
- instance

An environment may consist of one or more clusters.

```
{
  "PLATFORM": {
    "NOM_CLUSTER_1": [ .. ],
    "NOM_CLUSTER_2": [ .. ]
  }
}
```

A cluster consists of a list of nodes.

```
{
  "NOM_CLUSTER_1": [
    { "NOM_NOEUD_1": { .. },
      "NOM_NOEUD_2": { .. }
    }
  ]
}
```

A node consists of one or more instances.

```
{
  "NOM_NOEUD_1": {
    "COMMON": { ... },
    "INSTANCES": {...}
  }
}
```

An instance is made up of several keys / values.

```
{
  "INSTANCES": {
    "TYPE_INSTANCE_1": {
      "NOM_INSTANCE_1": { ... },
      "NOM_INSTANCE_2": { ... }
    },
    "TYPE_INSTANCE_2": { ... }
  }
}
```

Parameter(s) to configure:

Parameters	Description
ENVIRONMENT	Link to a shared variable or directly contains JSON.

Example of a test environment containing an http server with an instance of type rest. After loading into the cache, the REST instance is accessible by using the NODE_HTTP_REST key. All keys in COMMON are automatically copied to each instance.

```
{
  "PLATFORM": {
    "CLUSTER": [
      { "NODE": {
        "COMMON": {
          "HOSTNAME": "httpbin"
        },
        "INSTANCES": {
          "HTTP": {
            "REST": {
              "HTTP_DEST_HOST": "httpbin.org",
              "HTTP_DEST_PORT": 443,
              "HTTP_DEST_SSL": true,
              "HTTP_HOSTNAME": "httpbin.org",
              "HTTP_AGENT_SUPPORT": false,
              "HTTP_AGENT": null
            }
          }
        }
      }
    ]
  },
  "DATASET": [ ]
}
```

The DATASET key can contain datasets.

9.3 Data Generators

9.3.1 Hash SHA

Important: path of the reusable test /Snippets/Generators/01_Gen_Sha.tux

This reusable test is used to generate a hash of a value and store it in the cache.

Parameter(s) to configure:

Parameters	Description
DATA_IN	Hash character string
CACHE_KEY	Key name
SHA	Type of hash realize (sha1, sha256, sha512)

9.3.2 Hash MD5

Important: path of the reusable test /Snippets/Generators/01_Gen_Md5.tux

This reusable test is used to generate md5 hash and store it in the cache.

Parameter(s) to configure:

Parameters	Description
DATA_IN	Hash character string
CACHE_KEY	Name of the key or the result will be saved in the cache

9.3.3 UUID

Important: path of the reusable test /Snippets/Generators/01_Gen_Uuid.tux

This reusable test is used to generate an uuid and store it in the cache.

Parameter(s) to configure:

Paramètres	Description
CACHE_KEY	Name of the key to save the result in the cache

9.3.4 BASE64

Important: path of the reusable test /Snippets/Generators/01_Gen_Base64.tux

This reusable test is used to encode or decode a string and store the result in the cache.

Parameter(s) to configure:

Parameters	Description
CACHE_KEY	Name of the key to save the result in the cache
DECODE	Set to True to encode
ENCODE	To set to True to decode
URLSAFE	Set to True if the result after encoding is to be used in an url
STR_BASE64	Character string to encode / decode

9.3.5 GZIP

Important: path of the reusable test /Snippets/Generators/01_Gen_Gzip.tux

This reusable test can compress or uncompress a string and store the result in the cache.

Parameter(s) to configure:

Parameters	Description
CACHE_KEY	Key name
COMPRESS	To set to True to compress
UNCOMPRESS	Set to True to decompress
STR_GZIP	Character string to compress / decompress

9.4 Networks protocols

9.4.1 SSH

Important: path of the reusable test /Snippets/Protocols/01_Send_SSH.tsx

This reusable test is used to send a sequence of ssh commands. It is used in conjunction with the reusable test /Snippets/Do/03_Initilize.tux to load an environment into the cache.

Parameter(s) to configure:

Parameters	Description
SERVERS	List of servers to contact
COMMANDS	List of commands to run on the remote machine
TIMEOUT_CONNECT	Max time to connect to the remote machine

The *COMMANDS* parameter is waiting for one or more blocks of 4 lines. Each block must respect the following formalism:

1. A comment explaining the action, this information is used to initialize the test step
2. The command to execute
3. The string expected on the screen, if the expected value is not found then the step will be in error.
(optional line)
4. empty

Warning: Each block will be executed even if the previous one is in error.

The following example performs the following actions:

1. Send 3 pings on the remote machine whose ip is stored in the DEST_HOST cache
2. Verification of having the message on the screen indicating that the 3 packets have been sent.
Then the mddev value is stored in the cache with the STATS key

3. The second block clears the screen by sending the clear command.
4. Finally the test is waiting to find the prompt on the screen

```
# send a ping
ping -c 3 [!CACHE:SVR:DEST_HOST:]
.*3 packets transmitted, 3 received, 0% packet loss.*mdev = [!CAPTURE:STATS:] ms.*

# clear the screen
clear
.*root@.*
```

Note: It is possible to run the test multiple times by providing a server list.

Note: By default, the test waits for a maximum of 20 seconds to find the expected string. This value can be configured with the TIMEOUT parameter.

Note: By default, the test waits 10 seconds to connect to the remote server. This value can be configured with the TIMEOUT_CONNECT parameter.

9.4.2 HTTP

Important: path of the reusable test /Snippets/Protocols/01_Send_HTTP.tsx

This reusable test makes it possible to send an HTTP request by checking the response received. It is used in conjunction with the reusable test /Snippets/Do/03_Initilize.tux which loads an environment into the cache.

Parameter (s) to configure to set the destination:

Parameters	Description
SERVERS	List of servers to contact
TIMEOUT_CONNECT	Max time to connect to the remote machine

Parameter (s) to configure the HTTP request to send:

1	HTTP_REQ_BODY	custom	{ "login": "[!CA [...]"
2	HTTP_REQ_HEADERS	custom	Content-Type: applic [...]
3	HTTP_REQ_METHOD	str	POST
4	HTTP_REQ_URI	custom	/session/login

Parameter (s) to configure the expected HTTP response (and which will allow to consider the test as valid):

Parameters	Description
HTTP_RSP_BODY	Body of the expected answer.
HTTP_RSP_CODE	The expected HTTP code. 200 by default
HTTP_RSP_HEADERS	List of expected headers
HTTP_RSP_PHRASE	The expected HTTP sentence. OK by default
HTTP_RSP_VERSION	The expected HTTP version. HTTP / 1. [0 1] default

6	HTTP_RSP_BODY	custom	
7	HTTP_RSP_CODE	str	200
8	HTTP_RSP_HEADERS	custom	set-cookie: session_ [...]
9	HTTP_RSP_PHRASE	str	OK
10	HTTP_RSP_VERSION	str	HTTP/1.1

Note: The use of regular expressions is possible to check or save a value in the body of the answer or in the headers.

The screenshot shows the ExtensiveAutomation interface. On the left, a table lists test parameters: HTTP_REQ_METHOD (POST), HTTP_REQ_URI (/session/login), HTTP_RSP_BODY (logged in "projec [...]), HTTP_RSP_CODE (200), HTTP_RSP_HEADERS (set-cookie: session_id=[...]), HTTP_RSP_PHRASE (OK), and HTTP_RSP_VERSION. A red circle '1' highlights the HTTP_RSP_HEADERS row. On the right, a 'Test Config > Custom Values' dialog box is open, showing a regular expression: `[sS]et-[cC]ookie: session_id=[!CAPTURE:CAPTURED_SESSION_ID:];expires=.*`. A red circle '2' highlights this expression. The background shows a 'Tests' panel with a list of tests including SCENARIO (2/2 t), RUN (, GENERATE H, and BEST SESSION.

9.4.3 XML

Important: path of the reusable test /Snippets/Protocols/01_Send_XML.tsx

This snippet enable to send HTTP request with XML in body. The response can be checked too. This snippet should be used with /Snippets/Do/03_Initialize.tux to load the test environment in the cache.

Parameter(s) to configure the remote destination:

Parameters	Description
SERVERS	List of servers to test
TIMEOUT_CONNECT	Timeout to connect on the remote machine

Parameter (s) to configure the HTTP request to send:

Parameters	Description
HTTP_REQ_BODY	Request body
HTTP_REQ_HEADERS	List of headers to add
HTTP_REQ_METHOD	HTTP method (GET, POST, etc..)
HTTP_REQ_URI	URI

Parameter(s) to configure the expected HTTP response (and the test will be pass in this case):

Parameters	Description
HTTP_RSP_BODY	Xpaths to check
HTTP_RSP_CODE	HTTP code expected. 200 by default
HTTP_RSP_HEADERS	List of expected headers
HTTP_RSP_NAMESPACES	List of namespaces
HTTP_RSP_PHRASE	HTTP phrase expected. OK by default
HTTP_RSP_VERSION	HTTP version expected. HTTP/1.[0 1] by default

Warning: The test will be failed if the response does not content XML.

9.4.4 JSON

Important: path of the reusable test /Snippets/Protocols/01_Send_JSON.tsx

This snippet enable to send a HTTP request with JSON in body and to check the associated response. This snippet should be used with /Snippets/Do/03_Initilize.tux to load the test environment in the cache.

Parameter(s) to configure to set the remote machine:

Parameters	Description
SERVERS	List of remote machine to reach
TIMEOUT_CONNECT	Timeout of connection with the remote machine

Parameter(s) to configure the request to send:

Parameters	Description
HTTP_REQ_BODY	Request body
HTTP_REQ_HEADERS	List of header to add
HTTP_REQ_METHOD	HTTP method (GET, POST, etc..)
HTTP_REQ_URI	URI to call

Parameter(s) to configure the expected response (and the test will be pass in this case):

Parameters	Description
HTTP_RSP_BODY	List of xpath to check
HTTP_RSP_CODE	HTTP code expected. 200 by default
HTTP_RSP_HEADERS	List of header expected
HTTP_RSP_PHRASE	HTTP phrase expected. OK by default
HTTP_RSP_VERSION	HTTP version expected. HTTP/1.[0 1] by default

Warning: The test will be failed if the response does not content JSON.

9.5 User Interface

9.5.1 Open application in Windows

Important: path of the reusable test /Snippets/UI/01_Win_OpenApp.tux

This snippet enable to open a application on a Windows or Linux machine. The parameter AGENT_GUI must be configured with the agent to use.

Parameter(s) to configure:

Parameters	Description
APP_PATH	Application path to open

9.5.2 Close an application in Windows

Important: path of the reusable test /Snippets/UI/02_Win_CloseApp.tux

This snippet enable to close a application on a Windows or Linux machine. The parameter AGENT_GUI must be configured with the agent to use.

Parameter(s) to configure:

Parameters	Description
APP_NAME	Name of the application to close

9.5.3 Open a web browser

Important: path of the reusable test /Snippets/UI/03_OpenBrowser.tux

This snippet enable to open a browser on a Windows or Linux machine. The parameter AGENT_GUI_BROWSER must be configured with the agent to use.

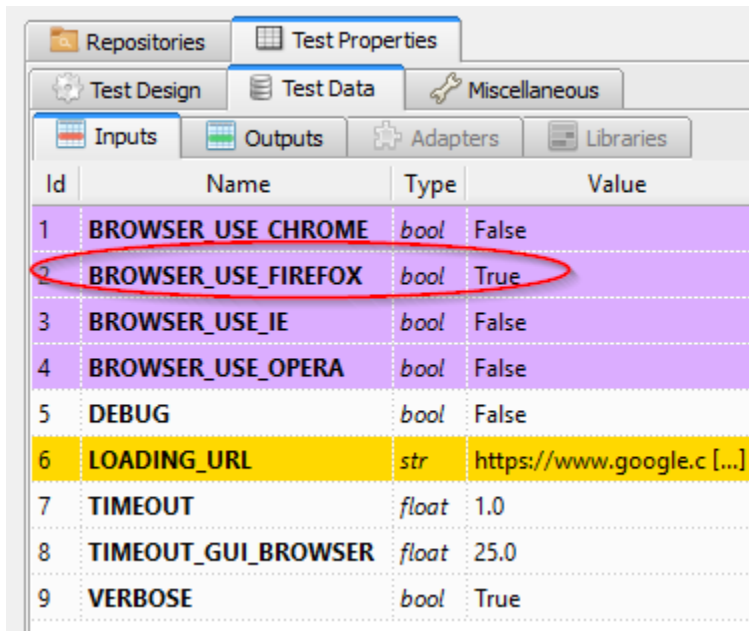
Parameter(s) to configure:

Parameters	Description
LOADING_URL	Website url to load

It's possible to select the browser to user, the following browsers are supported:

- Firefox
- Chrome

- Internet Explorer
- Opera
- Edge



Test Properties			
Test Data			
Inputs			
Outputs			
Adapters			
Libraries			
Id	Name	Type	Value
1	BROWSER_USE_CHROME	bool	False
2	BROWSER_USE_FIREFOX	bool	True
3	BROWSER_USE_IE	bool	False
4	BROWSER_USE_OPERA	bool	False
5	DEBUG	bool	False
6	LOADING_URL	str	https://www.google.c [...]
7	TIMEOUT	float	1.0
8	TIMEOUT_GUI_BROWSER	float	25.0
9	VERBOSE	bool	True

Note: the url must started with http:// or https://

9.5.4 Close a web browser

Important: path of the reusable test /Snippets/UI/03_CloseBrowser.tux

This snippet enable to close a browser on a Windows or linux machine. The parameter AGENT_GUI_BROWSER must be configured with the agent to use.

9.6 Checks

9.6.1 XML checks

Important: path of the reusable test /Snippets/Verify/01_Check_XML.tux

This snippet enable to check a XML content with xpath.

Parameter(s) to configure:

Parameters	Description
XML_STR	raw XML to inspect
XML_XPATH	xpath
XML_NAMESPACES	namespaces definitions

Example of value for the XML_STR parameter:

```
<NewDataSet>
<Table>
  <Country>France</Country>
  <City>Le Touquet</City>
</Table>
<Table>
  <Country>France</Country>
  <City>Agen</City>
</Table>
<Table>
  <Country>France</Country>
  <City>Cazaux</City>
</Table>
<Table>
  <Country>France</Country>
  <City>Bordeaux / Merignac</City>
</Table>
<Table>
  <Country>France</Country>
  <City>Bergerac</City>
</Table>
</NewDataSet>
```

Example of value for the XML_XPATH parameter.

```
(//NewDataSet/Table)[1]/City [!CAPTURE:CITY:]
```

The value will be accessible from the cache with the CITY key.

9.6.2 JSON checks

Important: path of the reusable test /Snippets/Verify/01_Check_JSON.tux

This snippet enable to check JSON content with jsonpath

Parameter(s) to configure:

Parameters	Description
JSON_STR	Json to inspect
JSON_XPATH	jsonpath

Example of value for the JSON_STR parameter:

```
{
  "args": {},
```

(continues on next page)

(continued from previous page)

```
"headers": {  
  "Connection": "close",  
  "Host": "httpbin.org",  
  "User-Agent": "ExtensiveTesting"  
},  
"origin": "190.117.217.129",  
"url": "https://httpbin.org/get"  
}
```

Example of value for the JSON_XPATH parameter.

```
headers.Connection    [!CAPTURE:CX:]
```

The value will be accessible from the cache with the CX key.

CHAPTER 10

Global variables

Global variables (or shared variables) are used to describe test environment. Variables are accessible from a test from the parameter of the type `global` or `list-global`.

10.1 Add/delete a variable

The adding or removing of a variable can be done from the web interface or the REST API. JSON must be used in variable. There are automatically availables from tests in properties.

10.2 Describe environment test

The description of a test environment must respect the following rules. This type of init must be used with the reusable test `/Snippets/Do/03_Initialize.tux`

Node declaration `SAMPLE_NODE`:

```
{
  "COMMON": {
    "HOSTNAME": "extensiveautomation"
  },
  "INSTANCES": {
    "SSH": {
      "ADMIN": {
        "SSH_DEST_HOST": "127.0.0.1",
        "SSH_DEST_PORT": 22,
        "SSH_DEST_LOGIN": "root",
        "SSH_DEST_PWD": "",
        "SSH_PRIVATE_KEY": null,
        "SSH_PRIVATE_KEY_PATH": null,
        "SSH_AGENT_SUPPORT": false,
        "SSH_AGENT": {
```

(continues on next page)

(continued from previous page)

```
        "type": "ssh",
        "name": "agent.ssh01"
      }
    }
  }
}
```

Data test declaration SAMPLE_DATASET_AUTH:

```
{
  "login": "admin",
  "password": ""
}
```

Environment declaration SAMPLE_ENVIRONMENT:

```
{
  "PLATFORM": {
    "CLUSTER": [
      { "NODE": "Common: SAMPLE_NODE" }
    ],
    "DATASET": [
      { "AUTH": "Common: SAMPLE_DATASET_AUTH" }
    ]
  }
}
```

10.3 Import/export variables

It's possible to export or import in mass the variables from REST API in CSV format

Warning: Variables are encoded in base64.

CHAPTER 11

Assisted designs

The client contains a automation assistant to create tests without knowledge in development. The assistant can be used for:

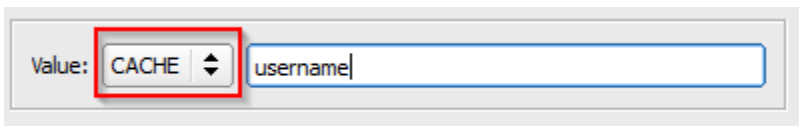
- Use the basic functions of the framework
- Execute system commands (ssh)
- Test applications with a heavy client
- Test web applications
- Run actions on an Android mobile

The test consists of a sequence of actions to perform. The wizard automatically generates a test unit or test suite. An existing test (script) can be updated from the wizard too.

To add an action in the assistant, you have to

- select the action to perform
- configure it
- save the action

The wizard natively supports the use of the cache. It is therefore possible save or retrieve values from the cache.



Note: Il est possible de mélanger les différents types d'actions.

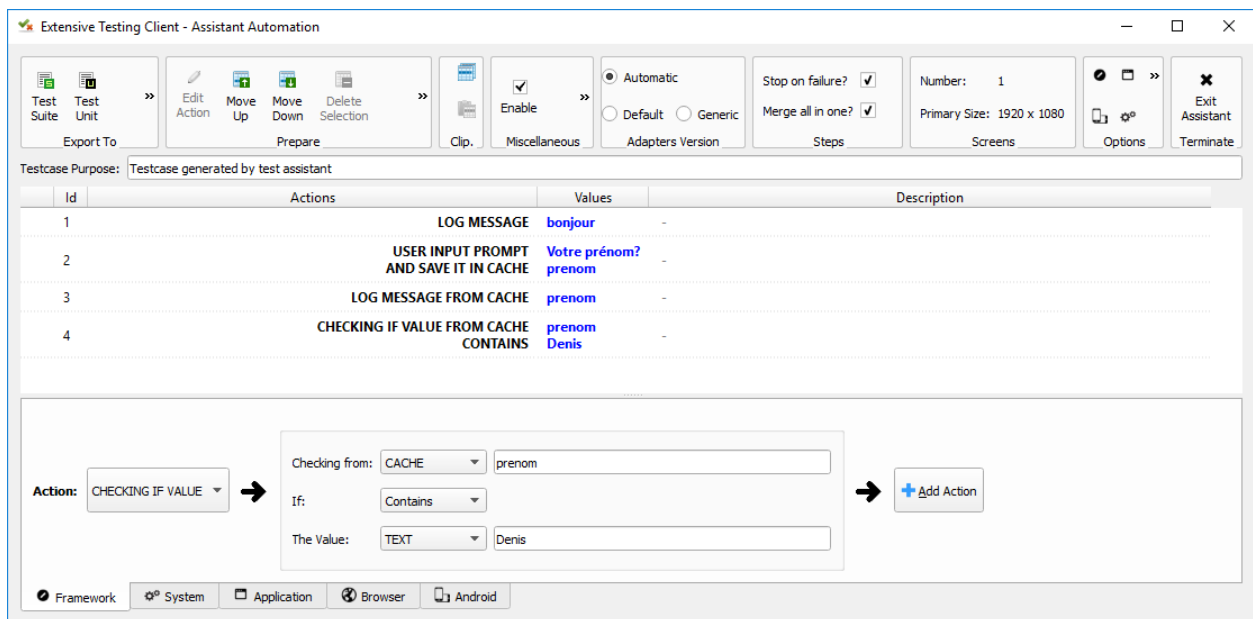
Important: The wizard allows to generate tests in automatic mode but it is also possible to add its own code inside with the USERCODE action.

11.1 Framework Tabulation

The framework tab allows you to use the basic functions of the test framework.

Example of a test done with the assistant:

1. Display the message “hello” in the test
2. Ask the user during the execution his first name and save it in the cache with the `firstname` key
3. Display the first name in the test log
4. Check from the cache if the first name contains a specific value.



List of available actions:

Note: In red, the essential actions.

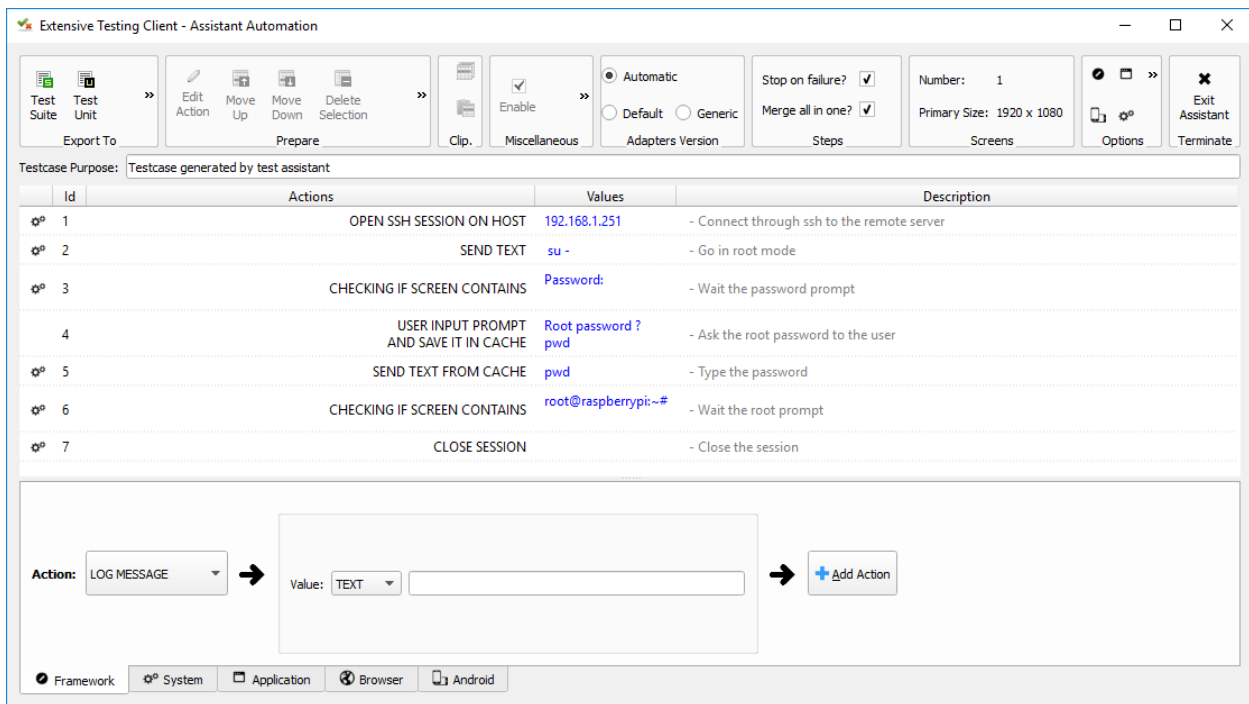
LOG MESSAGE	Displays an informational message during test execution
LOG WARNING	Display a warning message during test execution
SET VALUE	Saves a data in the cache
RESET CACHE	Blank the cache completely
USERCODE	Add custom code in the test
WAIT DURING	Wait for xx seconds
CHECK IF VALUE	Check if the value contains a specific text
ASK SOMETHING	Request a value to the user (interaction mode)

11.2 System tab

The system tab allows you to execute commands on a remote server available via SSH.

Example of a test done with the assistant:

1. Opening the ssh session on the remote machine 192.186.1.251
2. Sending the text *su -*
3. Waits to detect the text *Password:* on the screen
4. Ask the user for the root password and store it in the cache with the *pwd* key
5. Send the root password using the value stored in the cache
6. Waiting to detect on the screen the connection prompt
7. Close the SSH connection.



List of available actions:

Note: In red, the essential actions.

OPEN SSH SESSION	Open an SSH session
CLOSE SESSION	Close the session
CLEAR SCREEN	Blank screen
SEND TEXT	Send a string of characters
SEND SHORTCUT	Sending a keyboard shortcut (to interrupt an action)
CHECKING IF SCREEN	Check if the screen contains specific text

Note: Using the OPEN SSH SESSION action is mandatory before you can use the others available.

11.3 Tabulation application

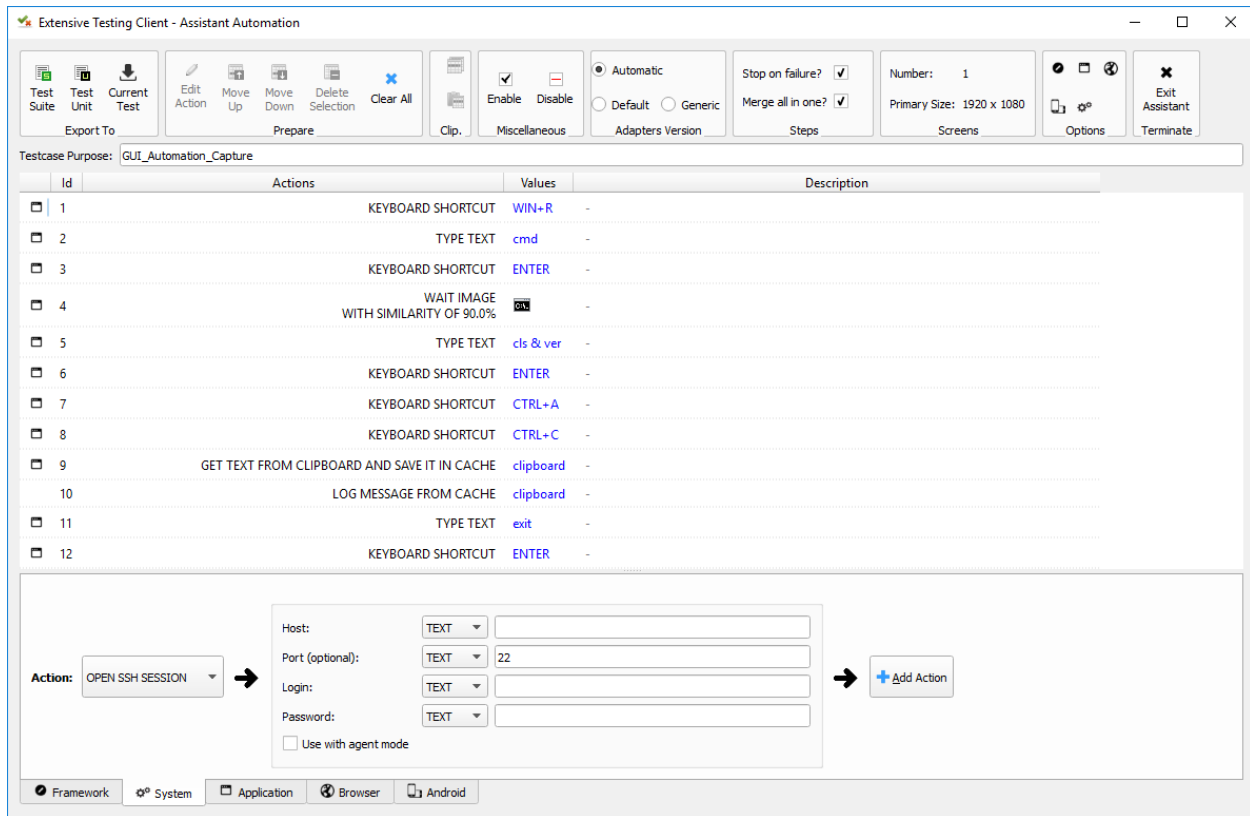
The application tab allows you to automate rich applications by allowing:

- to simulate the keyboard
- to simulate the mouse
- search for graphic elements on the screen
- to search for text

Warning: an agent sikulix-server is needed to use the actions.

Example of a test done with the assistant:

1. Send the keyboard shortcut *Win + R* to open the run window
2. Write the text *cmd*
3. Send the *Enter* keyboard shortcut to open a cmd window.
4. Waiting to detect the icon of the cmd window
5. Write the text *cls & ver* to display the version of Windows
6. Send the *Enter* keyboard shortcut to validate
7. Send the keyboard shortcut *Ctrl + A* to select the text in the window
8. Send the keyboard shortcut *Ctrl + C* to copy the selected text to the clipboard
9. Get the text from the clipboard and save it in the cache
10. Displays the text copied from the cache
11. Write the *exit* text in the cmd window
12. Send the *Enter* keyboard shortcut to close the window.



List of available actions:

Note: In red, the essential actions.

Mouse control

CLICK ON POSITION	Click on the position (x, y)
DOUBLE CLICK ON POSITION	Double click on the position (x, y)
RIGHT CLICK ON POSITION	Right click on the position (x, y)
MOUSE WHEEL DOWN	Turn the mouse wheel down
MOUSE WHEEL UP	Turn the mouse wheel up
MOVE TO POSITION	Move the cursor to the position (x, y)

Keyboard control

TYPE TEXT	Writes text
TYPE PATH	Writes text (to use for paths)
TYPE PASSWORD	Writes text (to be used to type a password)
GET TEXT FROM CLIPBOARD	Retrieves the text present in the clipboard
KEYBOARD SHORTCUT	Allows you to type a keyboard shortcut

String control

CLICK ON WORD	Search a word on the screen and click on it
DOUBLE CLICK ON WORD	Search for a word on the screen and double-click on it
RIGHT CLICK ON WORD	Search for a word on the screen and right-click on it
WAIT WORD	Search a word until it appears
WAIT AND CLICK ON WORD	Search a word until it appears and click on it

Image Control

CLICK ON IMAGE	Search an image and click on it
DOUBLE CLICK ON IMAGE	Search an image and double-click on it
RIGHT CLICK ON IMAGE	Search an image and right-click on it
WAIT IMAGE	Search an image until you see it on the screen
WAIT AND CLICK ON IMAGE	Search an image until you see it on the screen and click on it
HOVER MOUSE ON	Find an image and move the mouse cursor over it
DRAG IMAGE AND DROP TO	Find an image and drag and drop to position (x, y)

11.4 Browser Tabulation

The browser tab allows you to automate web applications by allowing:

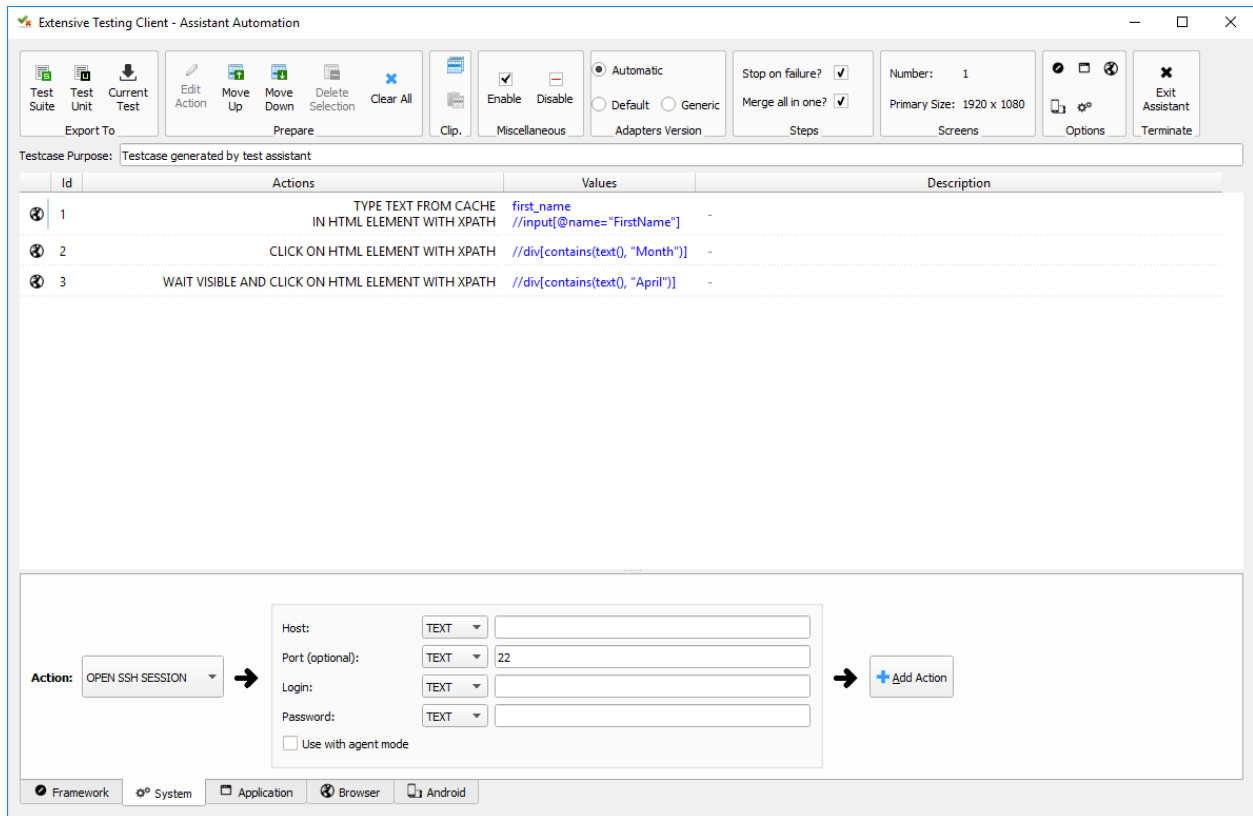
- to control browsers (firefox, internet explorer, chrome, edge)
- to simulate the keyboard

Warning: an agent selenium3-server or selenium2-server is needed to use the actions.

Tip: To click on an HTML element, it is advisable to use systematically the WAIT VISIBLE AND CLICK ON HTML ELEMENT function.

Example of a test done with the assistant:

1. Get the name from the cache and send it to the HTML element found by the xpath
2. Click on the HTML element found by the xpath
3. Find the HTML element found by the xpath and click on it as soon as it is visible on the screen.



Note: It is possible to open multiple browsers in parallel on the same extension to define a new session. The name of the session is defined by the OPEN BROWSER action. Then use the SWITCH TO SESSION action to switch sessions.

Available actions:

Note: In red, the essential actions.

Browser Control

OPEN BROWSER	Open the browser and load the specified url
CLOSE BROWSER	Closes the browser
MAXIMIZE BROWSER	Enlarges the browser window

Navigation actions

REFRESH PAGE	Refresh the page
GO BACK	Backspace
GO FORWARD	Go forward
ACCEPT ALERT	Validate the javascript alert
DISMISS ALERT	Dismiss the javascript alert
CLOSE CURRENT WINDOW	Closes the current window
SWITCH TO NEXT WINDOW	Toggle on next window
SWITCH TO FRAME	Toggle on the next frame
SWITCH TO SESSION	Toggles to another selenium session
SWITCH TO WINDOW	Toggle on the next frame

javascript actions

EXECUTE JAVASCRIPT ON HTML ELEMENT	Allows you to inject javascript script on an html element
------------------------------------	---

Actions on html elements

WAIT HTML ELEMENT	Wait for the appearance of a precise HTML element
WAIT AND CLICK ON HTML ELEMENT	Wait for the appearance of a precise HTML element and click on it
WAIT VISIBLE HTML ELEMENT	Wait for an HTML element to be visible to the user
WAIT NOT VISIBLE HTML ELEMENT	Wait until an HTML element is not visible to the user
WAIT VISIBLE AND CLICK ON HTML ELEMENT	Wait for an HTML element to be visible to the user and click on it
HOVER ON HTML ELEMENT	Move the mouse cursor over a specific HTML element
CLICK ON HTML ELEMENT	Click on a specific HTML element
DOUBLE CLICK ON HTML ELEMENT	Double click on a specific HTML element
CLEAR TEXT ON HTML ELEMENT	Empty the text on a specific HTML element
SELECT ITEM BY TEXT	Select item according to the text (for combolist or list)
SELECT ITEM BY VALUE	Select item according to the value attribute (for combolist or list)

Text Recovery

GET TEXT ALERT	Retrieves the text of an alert message javascript
GET TEXT FROM HTML ELEMENT	Retrieves the text an exact html element
GET PAGE TITLE	Retrieves the title of the page
GET PAGE URL	Get the URL of the page
GET PAGE SOURCE CODE	Get the source code page

Keyboard simulation

TYPE KEYBOARD SHORTCUT	Sends a keyboard shortcut to a specific HTML element
TYPE TEXT ON HTML ELEMENT	Sends text on a specific HTML element

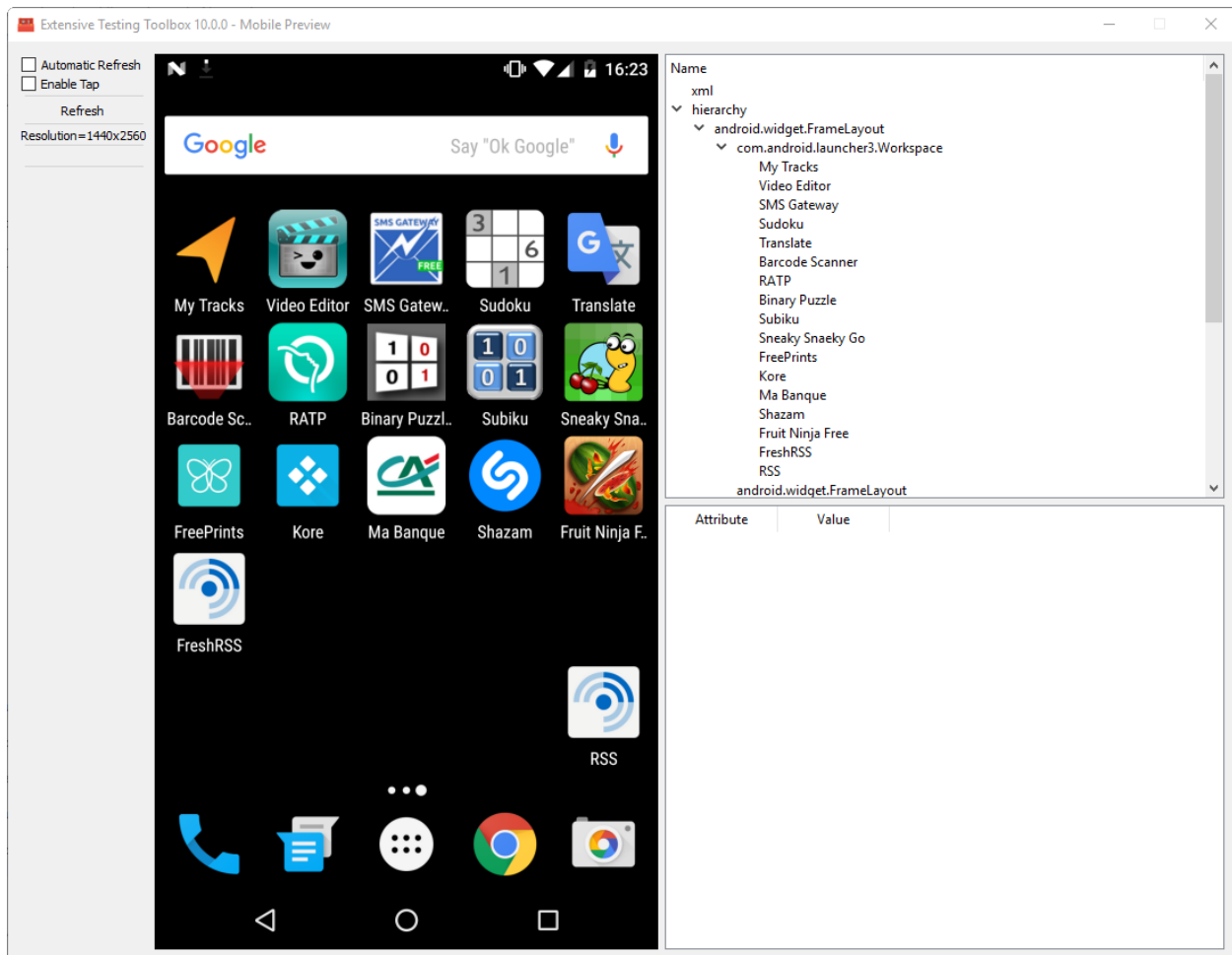
11.5 Android Tabulation

The android tab allows you to automate mobile applications by enabling:

- to simulate the keyboard
- to simulate the use of the fingers on the screen
- to control the system and the applications

Warning: an adb agent is needed to use the actions.

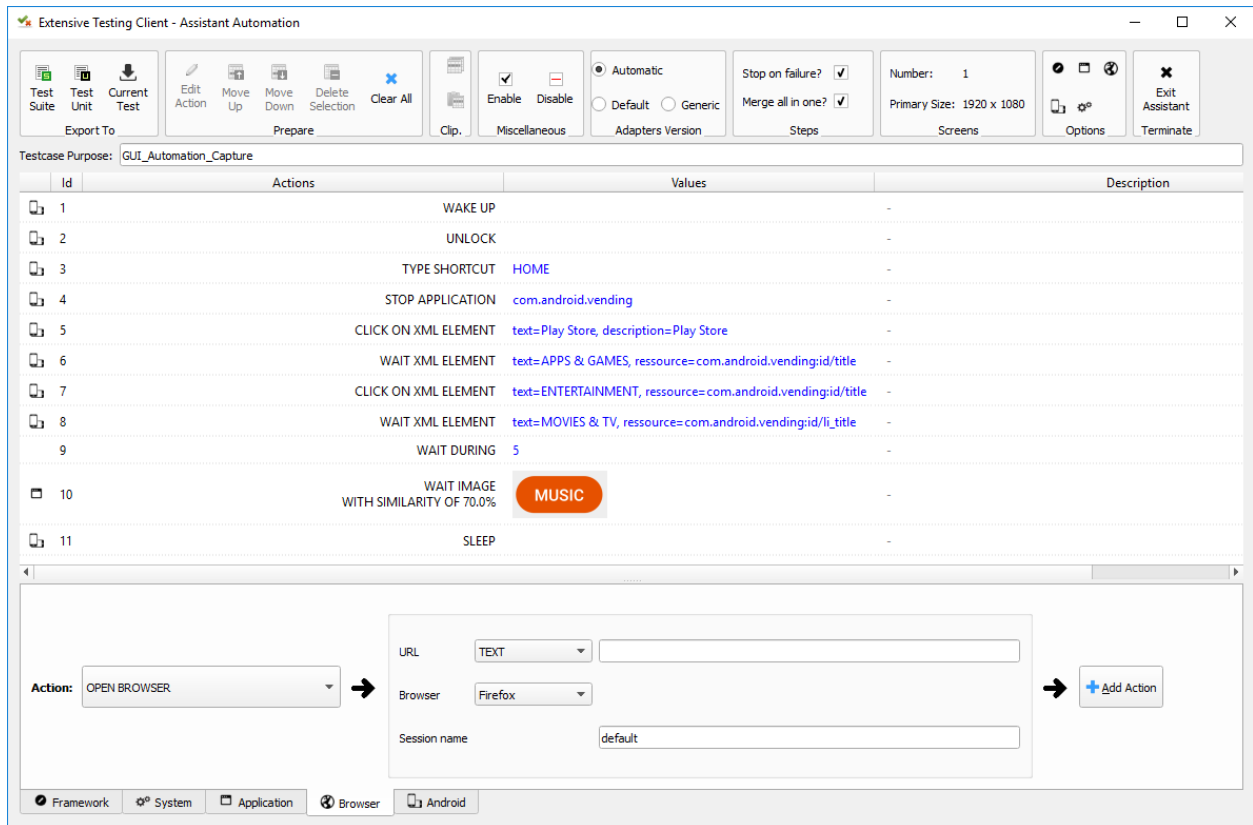
Overview of the agent



Example of a test done with the assistant:

1. Wake up the device
2. Unlock the device
3. Click on the *HOME* button
4. Stop the application
5. Click on the 'Play Store' app to open it
6. Wait for the application to open and search the *APPS & GAMES* menu
7. Click on the text *ENTERTAINMENT*
8. Click on the menu *MOVIES & TV*

9. Wait for 5 seconds
10. Research the image
11. Put the device to sleep.



Available actions:

Note: In red, mandatory actions.

Mobile controls

WAKE UP AND UNLOCK	Wake up and unlock the device
REBOOT	Restarting the device
SLEEP	Paused

Texts

TYPE SHORTCUT	Simulates a shortcut
TYPE TEXT ON XML ELEMENT	Sends text on a specific element of the interface
GET TEXT FROM XML ELEMENT	Retrieves the text of a specific element of the interface

Contrôles des éléments XML

CLEAR XML ELEMENT	Removes text from a specific element of the interface
CLICK ON XML ELEMENT	Click on a specific element of the interface
LONG CLICK ON XML ELEMENT	Long-term click on a specific element of the interface
WAIT AND CLICK ON XML ELEMENT	Wait for the appearance of a specific element of the interface and click on it

Tap on screen

CLICK TO POSITION	Click on the position x, y
DRAG FROM POSITION	Drag from position x1, y1 to x2, y2
SWIPE FROM POSITION	Swipe from position x1, y1 to x2, y2

12.1 Errors codes

Framework error code

Error code	Description
ERR_TE_000	Generic error during test execution
ERR_TE_001	Generic error during testcase execution
ERR_TE_500	Generic error during script execution

Steps errors

Error code	Description
ERR_STP_001	The step is already started, the function <i>start()</i> is called several time in the test.
ERR_STP_005	The test try to set the result bu the step is not started

Test properties errors

Error code	Description
ERR_PRO_004	The parameter name for <i>input</i> function does not exists

12.2 How to fix

12.2.1 Unable to generate the test design of a test

The generation of the description of a test does not works in some cases:

- bad version for adapters and libraries
- syntax error in the test

- the cache is used in the step definition

13.1 Server

Please refer to the README in [github](#)

13.2 Plugins for server

Please refer to the README in [github](#)

13.3 Web Client

Please refer to the README in [github](#)

13.4 Qt Client

Please refer to the README in [github](#)

13.5 Toolbox

Please refer to the README in [github](#)

13.6 Plugins for client

Please refer to the README in [github](#)

14.1 Start/Stop of the server

The server can be controlled with the following command `./extensiveautomation`. This command enables to

- start or stop the server
- check the status
- display the version

Use the following command to start the server `./extensiveautomation start`.

```
# ./extensiveautomation start
```

Use the following command to stop the server `./extensiveautomation stop`.

```
# ./extensiveautomation stop
```

Tip:

More details in logs about the start or stop procedure.

```
# tailf [...]Var/Log/output.log
2014-12-06 11:00:54,092 - INFO - Extensive Automation successfully started (in 14 sec.)
...
2014-12-06 10:58:51,810 - INFO - Stopping server
2014-12-06 10:58:51,911 - INFO - Extensive Automation successfully stopped!
```

14.2 Server status's

`./extensiveautomation --status` enable to check the status of the server, 3 states exists:

- starting: the server is starting
- running: the server is running properly
- stopped: the server is stopped

Tip: Don't forget to check the status of your reverse proxy.

14.3 Server settings

The file `settings.ini` contains all parameters to configure the server. Parameters are separated in several sections:

- Boot
- Notifications
- Client_Channel
- Agent_Channel
- WebServices
- TaskManager
- Network
- Paths
- Bin
- Server
- Bind
- Misc
- MySql
- Trace
- Backups
- Default
- Supervision
- Users_Session

CHAPTER 15

Projects

The solution is multi-project. It is therefore possible to organize the tests by projects and grant access rights for users.

Note: The Common project exists by default, it is accessible by all users and can not be deleted.

15.1 Add a project

Only an administrator can add a new project. Creating a project requires specifying its name and can be done via the web interface or the API

15.2 Delete a project

Only an administrator can remove a project. This action can be done through the web interface or the web api.

Note: If the project is associated with a user, deletion is not allowed.

15.3 Link a project to a user

A user can access to several projects. From the profile of a user, you can select all the projects authorized. It's possible to define the default one.

CHAPTER 16

Users

The solution is multi-user, 3 users exists by default:

- Admin
- Tester
- Monitor

Note: Don't forget to disable default accounts in a production environment.

16.1 Add user

Only an administrator can add a new user. The creation of a user requires at least the following parameters and can be done via the web interface or the API

- username
- password

Note: Email is used by the solution to send test reports and results.

Note: It is possible to configure multiple email addresses for a user by separating them with ;

16.2 Delete a user

Only an administrator can remove a user. This action can be done through the web interface or the web api.

17.1 Logs

17.1.1 Server

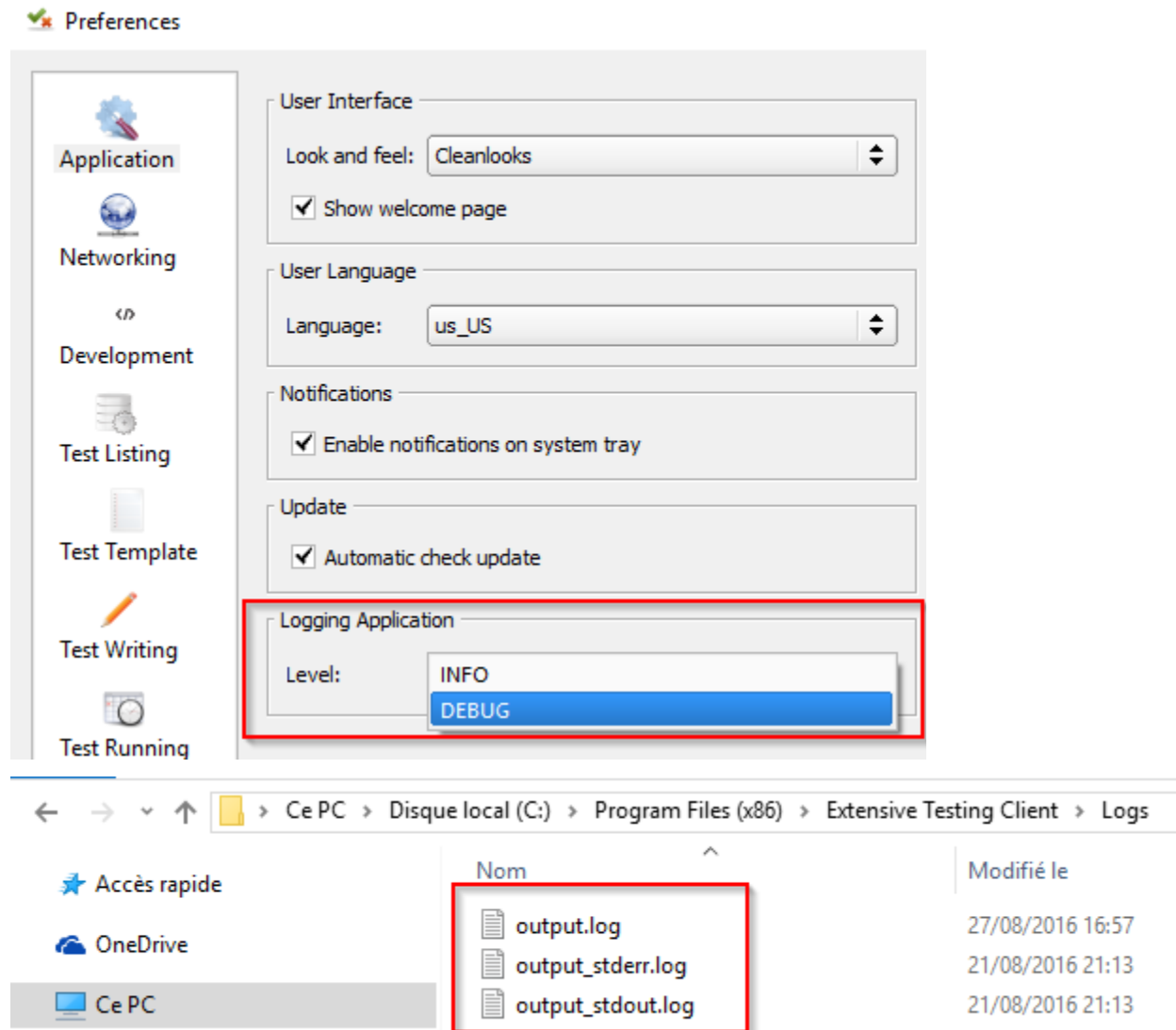
The server logs are stored on [...]Var/Logs/. The logs are set in INFO mode by default. The DEBUG level can be activated from the settings.ini file.

Note: It is possible to change the level of logs by doing an `xtcl reload`

17.1.2 Client

The client logs are available in <Program Files> /Extensive Automation Client/Logs/ The logs are set in INFO mode by default.

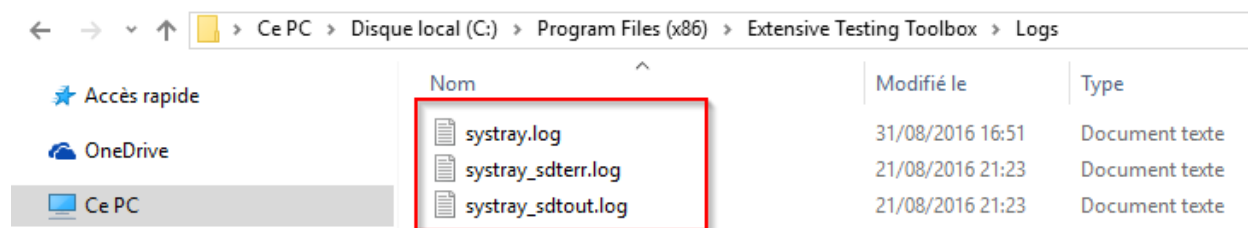
The DEBUG level can be activated from the client preferences.



17.1.3 Toolbox

The logs in the toolbox are available in <Program Files>/Extensive Automation Toolbox/Logs/. The logs are set in INFO mode by default.

The DEBUG level can be activated from the settings.ini file.

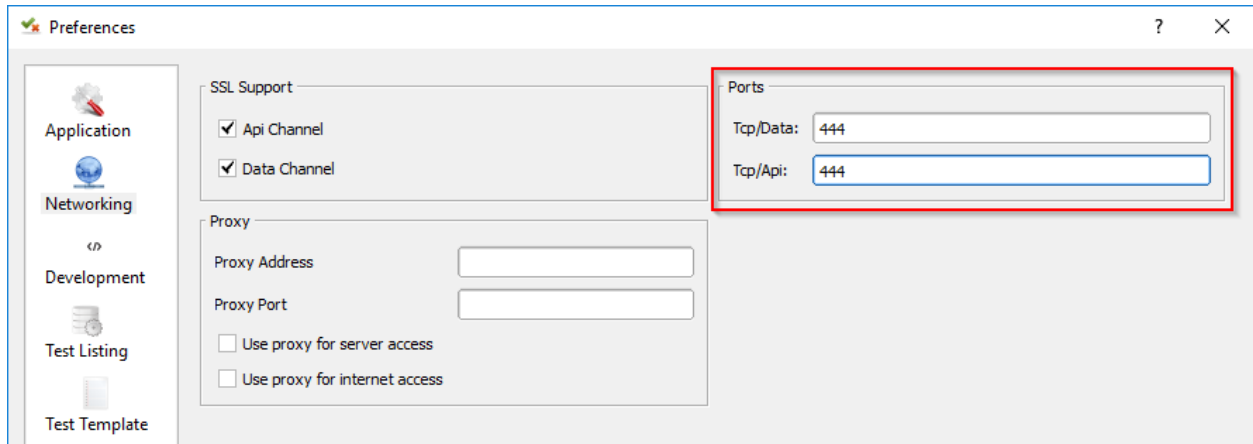


Note: A restart of the toolbox is necessary to take into account the change

17.2 Frequently Asked Questions

17.2.1 How to change the connection port of the client?

The destination port can be changed from the client preferences. Or directly from the `settings.ini` file.



17.2.2 View the server version?

```
./extensiveautomation version
Server version: 20.0.0
```

What if my connection to the server does not work? ~~~~~

If the connection from the client to the server does not work, an analysis is necessary.

The first thing to do is to connect to the server in SSH and execute the `extensiveautomation status` command to check if the server is running.

1. If the server is running then check:

- network connectivity in the client and the server
- a firewall blocking the https flow (443)

2. If the network connectivity is good and the server is working (or not), check the logs. The file is available in the `[...]/Var/Logs/output.log` directory. You must look for messages of type ERROR

How to fix the error “hping3 is not installed”? ~~~~~

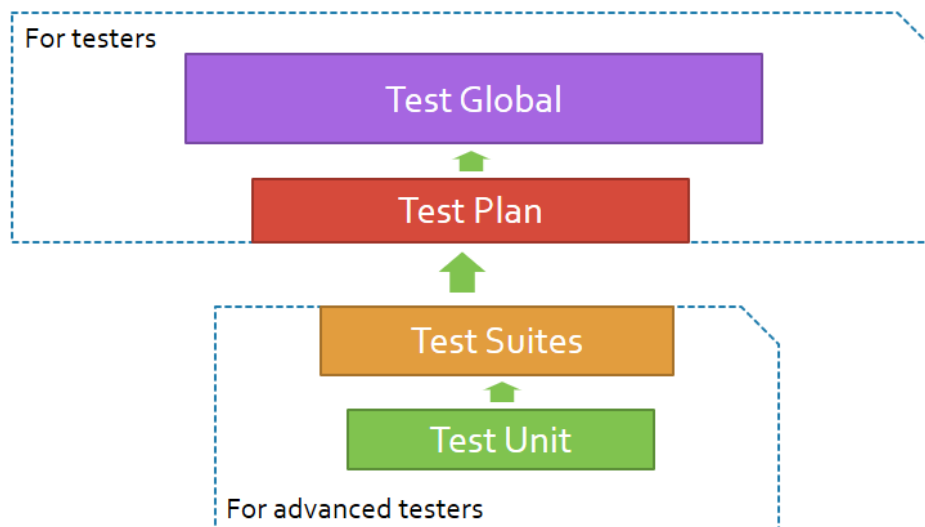
This error occurs while running a test when the Pinger adapter is used. Indeed requires to have the hping3 system library installed on the server.

You have to retrieve the sources from <https://github.com/antirez/hping> and compile them:

```
cd hping-master
yum install libpcap-devel-1.5.3-9.el7.x86_64
ln -s /usr/include/pcap/bpf.h /usr/include/net/bpf.h
./configure
make
make install
```


The solution is based on different types of tests for:

- enable the construction of advanced tests
- decrease the use of script



18.1 Test Unit

The test unit (tux) allows you to write a test case with several steps. This format is oriented development.



Test Unit
Testcase scripting

18.2 Test Suite

The test suite (tsx) allows you to write several test cases with several steps. This format is oriented development.



Test Suite
Multiple testcases scripting

18.3 Test Plan

The plan test (tpx) allows you to write test cases. The design is realized by nesting the tests *abstract*, *unit* and *suite*. This test format requires no knowledge in development.



Test Plan
Procedural test conception

18.4 Test Global

The global test (tgx) allows you to write test campaigns. The preparation of the campaigns is carried out by importing the tests *plans*.



Test Global
End to end test conception

Note: It is also possible to import other types of tests.

The test framework provides a framework for standardizing the creation of test cases.

The main features are:

- the support of test cases with steps
- Support extensions to communicate with the system to test or control
- automatic generation of test reports.

19.1 Test case

The creation of a test case in the solution is standardized.

A test case is divided into 4 sections:

- description: description of the different stages of the test
- prepare: preparation of adapters and libraries for communicating with the system to be tested or piloted
- definition: test flow
- cleanup: cleaning phase

The result of a test case is automatically calculated by the framework when the test is completed according to the different stages defined.

There are 3 possible results:

- PASS: all the steps of the tests have been successfully executed
- FAILED: at least one step is in error after execution
- UNDEFINED: at least one step of the test has not been executed

Note: The cleanup section is always called, even if there is an error.

19.2 Test steps

A test case is divided into sub-steps.

A step is defined by:

- a summary of the action to be carried out
- the detailed description of the action to be carried out
- the description of the expected result to validate the step.

The definition of the test steps must be done in the description section:

```
self.step1 = self.addStep(  
    expected="Logged in",  
    description="Login through the rest api",  
    summary="Login through the rest api",  
    enabled=True  
)
```

The result of a step is to be specified in the definition section

Example to set the result to PASS or FAILED

```
self.step1.setPassed(actual="step executed as expected")  
self.step1.setFailed(actual="error to run the step")
```

Warning: Do not forget to start a step with the function start otherwise it is not possible to put the result.

Note: Do not forget to specify the result of a step, otherwise it will be considered as UNDEFINED.

Important: A step set to FAILED can not become PASS thereafter in a test.

19.3 Cancellation of a test

It is possible to force the execution of a test case by using the stop function in the description section of your test.

```
Test(self).interrupt(err="aborted by tester")
```

Using the stop feature will stop the test and automatically call the cleanup section of the test case. In this case, the aborted argument is set to True by the framework to indicate the cancellation of the test.

```
def definition(self):
    Test(self).interrupt("bad response received")

def cleanup(self, aborted):
    if aborted: self.step1.setFailed(actual="%s" % aborted)
```

19.4 Adding trace

The framework provides some functions to add messages during the execution of a test.

The following levels are available:

- Example to display a message of type info

```
Trace(self).info(txt="hello world")
```

- Example to display a warning message

```
Trace(self).warning(txt="hello world")
```

- Example to display an error message

```
Trace(self).error(txt="hello world")
```

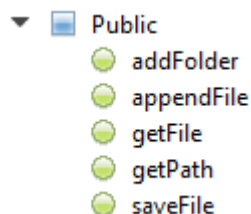
Note: If an error message is displayed then the result will automatically be set to FAILED.

Note: Messages appear automatically in the basic report.

19.5 Data

19.5.1 Public

A public space is available on the test server. This space makes it possible to provide files that are necessary during the execution of a test.

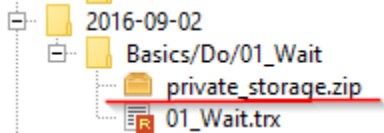


The files are stored in the `/opt/xtc/current/Var/Public/` directory on the server.

Warning: This space is common to all projects configured on the server.

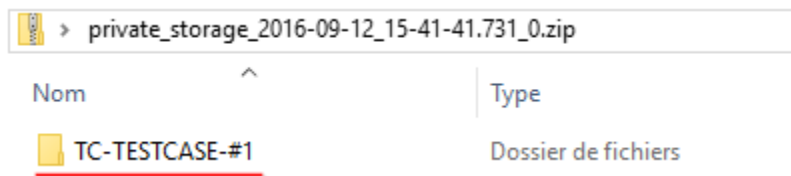
19.5.2 Private

Private vault only exists while running a test. It can save logs generated or recovered during the execution of the test. These logs are automatically made available to the user in a zip file when the test is completed. They can be retrieved from the client or from the server API.



The logs are organized by directory:

- TC-TESTCASE directory - # <id_tc>: contains the logs generated by the test case
- ADP directory - # <id_id>: contains the logs generated by the different adapters used during the test



Example to save the text *hello world* in a 'my_logs' file from the test case

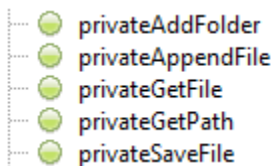
```
Private(self).saveFile(destname="my_logs", data="hello world")
```

Example to add text to an already existing log file

```
Private(self).appendFile(destname="my_logs", data="hello world2")
```

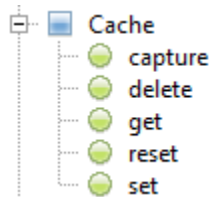
Note:

It is also possible to save files from an adapter. They will be automatically stored in a directory with the name of the adapter.



19.5.3 Cache

The test framework allows caching of data in the key/value form. This function may be necessary to share data between tests when writing a scenario for example.



Example to save a value in the cache

```
Cache().set(name="my_data", data="hello")
```

Read a value from the cache

```
my_data= Cache().get(name="my_data")
Trace(self).warning(my_data)
```

Example to capture a data with a regular expression and with record in the cache

```
my_data="March, 25 2017 07:38:58 AM"

Cache().capture(data=my_data, regexp=".* (?P<TIME>\d{2}:\d{2}:\d{2}) .*")

Trace(self).info( txt=Cache().get(name="TIME") )
```

STCASE	Preparing
STCASE	Starting
STCASE [Step_1]	step sample
STCASE	07:38:58
STCASE [Step_1]	Success
STCASE	Cleaning
STCASE	FND

It is also possible to rely on a custom parameter to supply the regular expression.

```
.*session_id=[!CAPTURE:SESSIONID:];expires.*
```

or in greedy mode

```
.*session_id=[!CAPTURE:SESSIONID:.*?];.*
```

Important: The cache exists only during the execution of a test.

19.6 Put on hold

This function allows you to pause while running a test.

Example of holding for 10 seconds:

```
Time(self).wait(timeout=10)
```

Standby example until the current date and time match the specified date:

```
Time(self).waitUntil(dt='2016-09-12 02:00:00', fmt='%Y-%m-%d %H:%M:%S', delta=0)
```

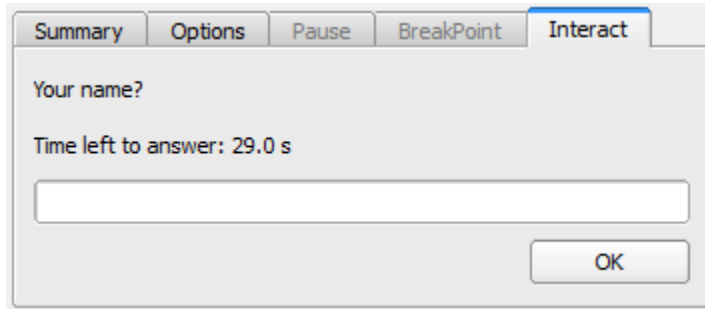
19.7 Interaction with the tester

The framework makes it possible to write semi-automatic tests, ie in interaction mode. This function can be interesting for a test in interactive mode (eg configuration of a device)

Example asking the name of the person:

```
user_rsp = Interact(self).interact(ask="Your name?", timeout=30.0, default=None)
```

From the client, the Interact tab automatically appears to answer the question asked during the execution of the test. This window is available from the analysis window.



Note: If no response is provided within the interval, it is possible to provide a default value with the default argument.

19.8 Parameters of a test

19.8.1 Inputs

Input parameters are used to add variables to a test. They are configurable from the client.

There are several types of parameters:

The variables are accessible from a test with the input (...) function

```
input('DEBUG')
```

The text parameter

The text type is used to construct parameters that use other parameters or the cache. It is therefore possible to use keywords that will be interpreted by the test framework at the time of execution.

List of available keywords:

Keywords	Description
[! INPUT: :]	Retrieves the value of a parameter present in the test
[! CACHE: :]	Retrieves a value present in the cache

Note: The name of a parameter is unique and must be capitalized.

19.8.2 The agents

Inputs				
Id	Name	Type	Value	Description
5	SUPPORT_AGENT	bool	False	

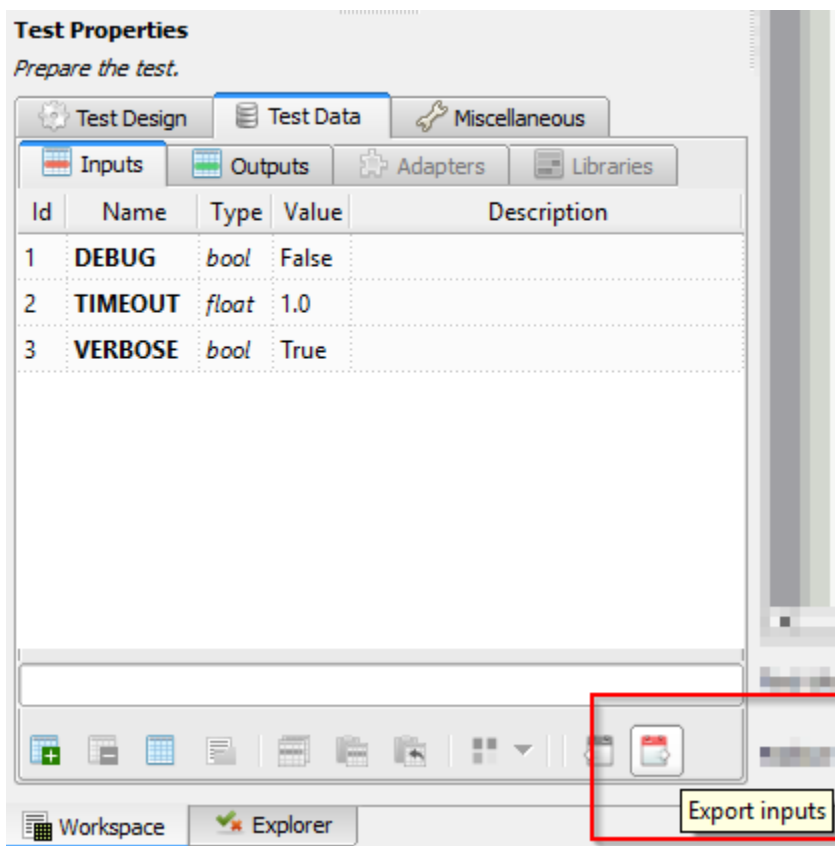
The list of agents can be accessed from a test using the () key mode.

```
self.ADP_REST= SutAdapters.REST.Client(
    parent=self,
    destinationIp=input('HOST'),
    destinationPort=input('PORT'),
    debug=input('DEBUG'),
    sslSupport=input('USE_SSL'),
    agentSupport=input('SUPPORT_AGENT'),
    agent=input('AGENT_SOCKET')
)
```

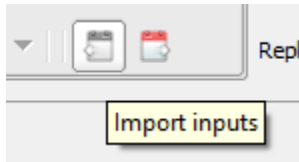
Define the agent parameter with json like that {'name': 'agent1', 'type': 'socket'}

19.8.3 Import / export settings

The test parameters can be exported to a dedicated testconfig (tcx) file type. It is therefore possible to prepare the parameters without having the test.



It is possible to import a configuration file into a test. The import will overwrite all the parameters if the name is the same.



20.1 The events

The execution of a test is divided into events, all of these events are stored and can be viewed afterwards. An event can represent:

- an action performed by the test framework
- an action performed by the test
- a data item received by the system to be tested or checked.
- data to send to the system to test or control.

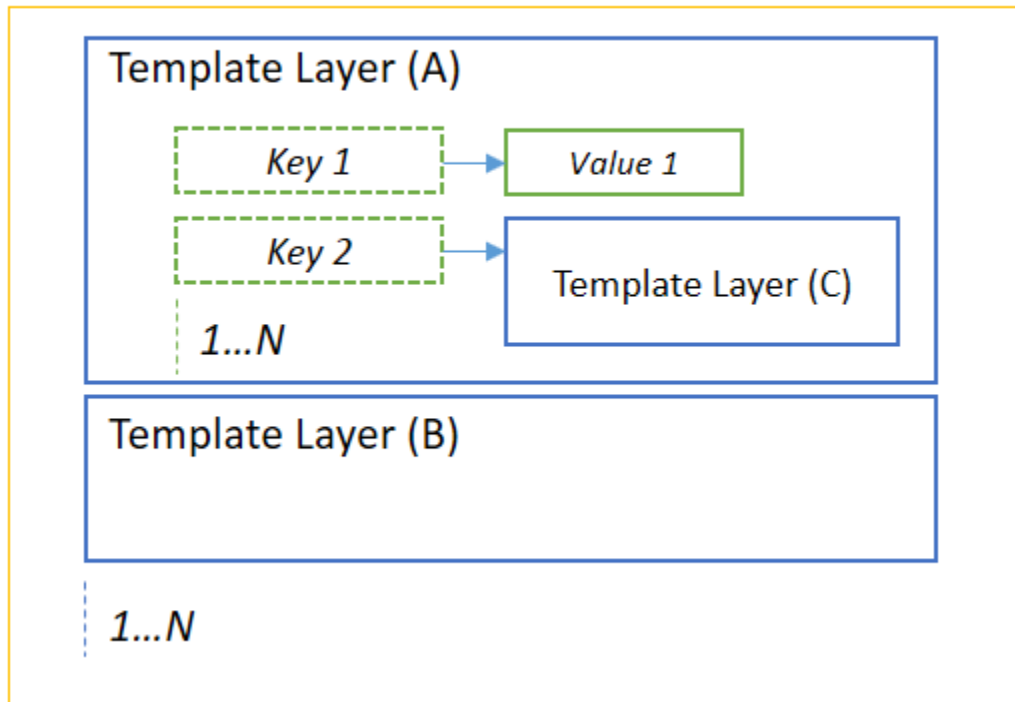
Event execution makes it possible to have robust tests thanks to the definition of observation intervals. The approach is to write the tests with the following formalism:

- I perform an action in my test.
- During a given interval, I look and compare all the events received with an expected.
- **I decide on the next action**
 - after receiving the event I was waiting for
 - or when the observation interval is over.

During the execution of a test, the framework captures all the events generated by the tested or piloted system. The events are then converted and stored in a message called `template`.

A `template` is split into one or more layers. A layer is defined by a set of key/value. The value of a layer can be another layer too.

Template Message



20.1.1 Creating a template

Creating a template can be done using the TestTemplates test framework.

```
tpl = TestTemplates.TemplateMessage ()
layer = TestTemplates.TemplateLayer (name = 'response')
layer.addKey (name = 'code', data = '200')
layer.addKey (name = 'msg', data = 'hello world')
tpl.addLayer (layer = layer)
```

This template indicates that the event should contain:

- a layer called *response* and containing the key 'code' and *msg*
- the key code must be strictly equal to the value 500
- the msg key must be strictly equal to the text hello world.

Example of an expected message visible from the graphical client:

Key	Value
[-] any()	
notcontains('source2')	any()
endswith('source-ip')	127.0.0.1
[-] contains('TCP')	
destination-port	greaterthan(70.0)
contains('source')	60246
tcp-event	startswith('conn')
[-] HTTP	
[-] contains('request')	endswith('HTTP 1.1')
version	contains('1.1')
[-] endswith('PP')	
contains('raw')	xxxxxxxxxx

20.1.2 The operators

Operators are available to facilitate comparison of the models received.

Sample template using comparison operators:

```
tpl = TestTemplates.TemplateMessage()
layer = TestTemplates.TemplateLayer(name='response')
layer.addKey(name='code', data=TestOperators.LowerThan(x=500))
layer.addKey(name='msg', data=TestOperators.Contains(x="hello"))
tpl.addLayer(layer=layer)
```

This template indicates that the event should contain:

- a layer called *response* and containing the key 'code' and *msg*
- the code key must be less than 500
- the msg key must contain the text hello.

20.1.3 Visualization

The client can graphically display the comparison made by the framework.

24	19:27:01.6247	ADAPTER #4	USER	MATCH-STARTED	HTTP [Match_0]	Wait the expected template(0) for 10.0 seconds
56	19:27:01.8739	SUT	ADAPTER #4	RECEIVED	HTTP	⇐ HTTP/1.1 200 OK (application/json)
57	19:27:01.8955	ADAPTER #4	USER	MATCH-INFO	HTTP [Match_0]	Template(0) mismatch, continues waiting
59	19:27:01.9276	SUT	ADAPTER #5	RECEIVED	HTTP>TCP	⇐ disconnected by peer
61	19:27:11.6350	ADAPTER #4	USER	MATCH-EXCEEDED	HTTP [Match_0]	Waiting time exceeded
62	19:27:11.6352	TE	USER	SECTION	TESTCASE	Cleaning
63	19:27:11.6354	TE	USER	STEP-FAILED	TESTCASE [Step_1]	Abort reason: expected rest response not received.
64	19:27:11.9406	TE	USER	INFO	TESTCASE	END

Key	Value	Status Key	Status Value
▶ IP4		▶ IP4	
▶ TCP		▶ TCP	
▶ SSL		▶ SSL	
▼ HTTP		▼ HTTP	
body	{ "expires": 86400, "user_id": 2, "cmd": "/session/ [...]" }	▶ response	
▶ headers		▼ headers	
response		regex('[s]et-[cC]ookie')	regex('session_id=(?P<CAPTURED_SESSION_ID>.*):expir [...]
phrase	OK		
code	200		
version	HTTP/1.1		

Definition of the color code:

Green	Perfect match between the value received and expected
Red	The value received does not correspond to the expected value
Yellow	The expected value has not been verified

20.2 Test reports

After each run of a test, the framework automatically generates the associated test reports.

There are 2 type reports:

- An advanced report
- A basic report (accessible by default from the graphical client)

The reports are accessible from the client, the web interface or from the API.

Note: Reports can be exported in html, csv, xml and pdf format.

20.2.1 Advanced report

The advanced report displays information such as:

- the execution time of each test case
- the complete description of the test steps.
- performance statistics.
- the test parameters.

Advanced Report

Basic Report

Xml Verdict

Text Verdict

Preview

Comments

To TXT

To HTML

To PDF

To Printer

Exports

Test report auto-generated by [Extensive Testing](#) - Copyright (c) 2010-2017 - Denis Machard

General Test Description

Global Result FAILED

Duration 10.859 seconds

	PASS	FAIL	UNDEF	TOTAL
Statistics				
TESTCASE:	3 (75%)	1 (25%)	0	4
TESTABSTRACT:	0	0	0	0
TESTUNIT:	3 (100%)	0	0	3
TESTSUITE:	0	1 (100%)	0	1
TESTPLAN:	1 (50%)	1 (50%)	0	2
TESTGLOBAL:	0	1 (100%)	0	1

Terminated At 21/01/2018 19:27:11

Run At 21/01/2018 19:27:01

Run By admin

Test Name 0001_Rest_Api

Test Path //Self Testing/REST API/

Project Name Common

Adapters v1110

Libraries v800

Test Global - 0001_Rest_Api

Summary Just a basic sample.

Started At 21/01/2018 19:27:01

Duration 10.858 seconds

Test Plan - PREPARE TEST

Test Plan - LOGIN TO THE REST API

It is possible to display variables in the test report by prefixing the variables:

- SUT_ Variables describing the version of the system to be tested or piloted
- DATA_ Variables describing specific data
- USER_ User variables

This feature can be useful for increasing the level of traceability in reports.

Test Design Test Data Miscellaneous				
Inputs Outputs Adapters Libraries				
Id	Name	Type	Value	Description
1	DEBUG	bool	False	
2	SUT_TEST	float	1.0	
3	VERBOSE	bool	True	

Test Suite: Noname2

Summary Just a basic sample.

Started At 07/01/2017 06:19:49

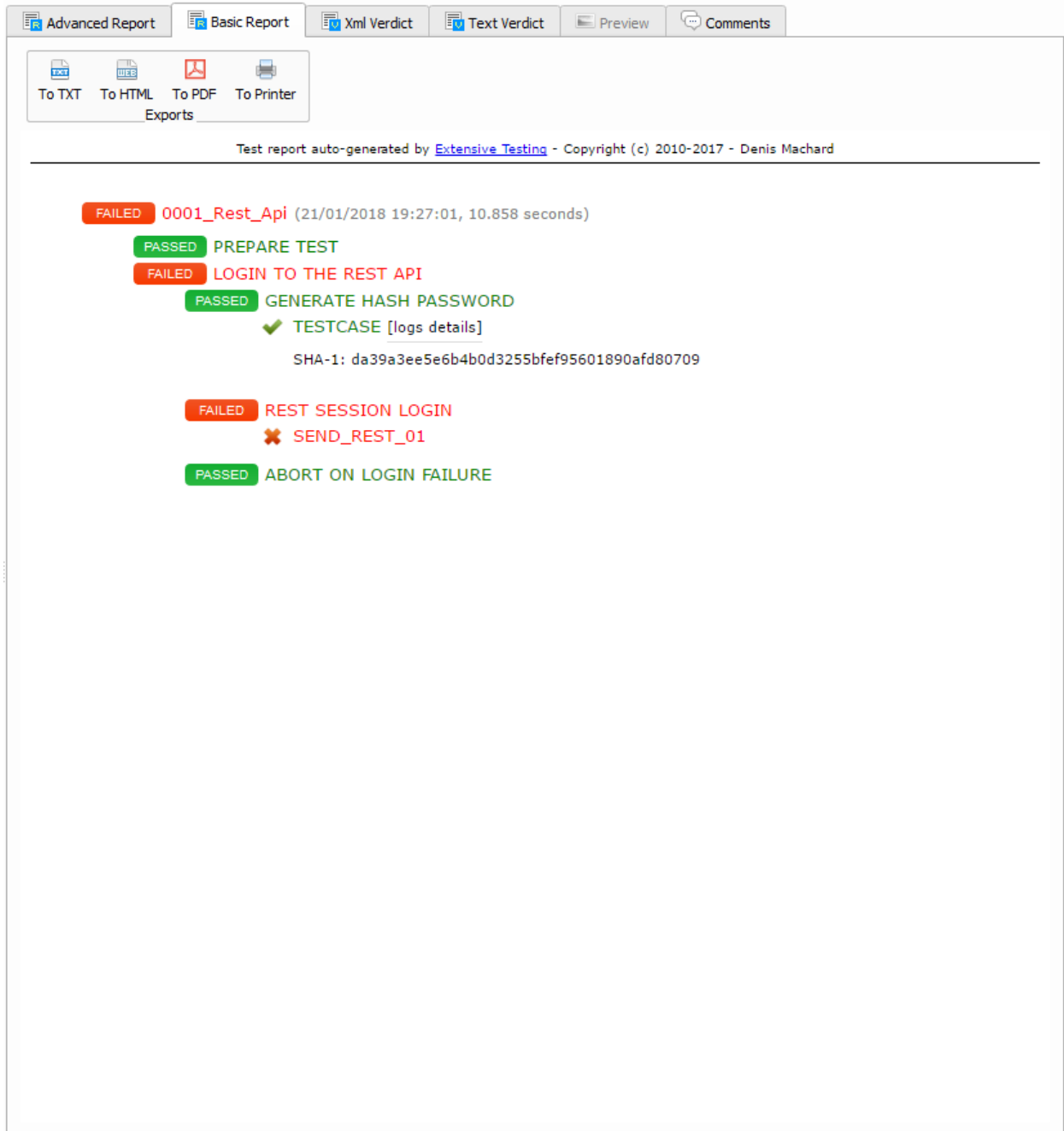
Duration 0.127 seconds

SUT Inputs SUT_TEST (float) = 1.0

TestCase: TESTCASE_01

20.2.2 Basic report

The basic report summarizes the result of all test cases and reports.



Test report auto-generated by [Extensive Testing](#) - Copyright (c) 2010-2017 - Denis Machard

FAILED 0001_Rest_Api (21/01/2018 19:27:01, 10.858 seconds)

- PASSED** PREPARE TEST
- FAILED** LOGIN TO THE REST API
 - PASSED** GENERATE HASH PASSWORD
 - ✓** TESTCASE [logs details]
SHA-1: da39a3ee5e6b4b0d3255bfef95601890afd80709
- FAILED** REST SESSION LOGIN
 - ✗** SEND_REST_01
- PASSED** ABORT ON LOGIN FAILURE

Color code:

Green	The test case is valid
Red	The test case is in error
Orange	The result of the test case is not determined
Gray	The test case was not executed

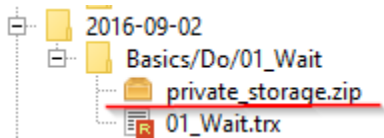
Tip: You must click on the test cases to display the steps.

Note: The messages displayed by the test with the `Trace (self) .info ()` function are available in the report by clicking on the [logs details] link.

Errors are also displayed by clicking on the [errors details] link.

20.3 The logs

The framework allows you to save logs while running a test and make them available quickly to the uses. All additional logs are zipped and accessible from the client or the API.



Note: For more details, read the chapter *The fundamentals >> Data*.

21.1 Adapters

The adapters allow communication with the system to be tested or piloted. The solution embeds several default adapters.

- network protocol support
- application level protocol support
- communication with databases
- systems interaction
- interaction with graphic interfaces
- telecom protocol support

Adapters have two modes of use:

- a direct mode: communication is done directly from the test server to the system to be controlled.
- an agent mode: the communication with the system to be controlled is done through an agent communicating with the test server.

Note: The Verbose mode is enabled by default on all adapters. This mode can be disabled to reduce the number of events during a test.

Note: The Debug mode is not enabled by default. It can be activated in case of problem.

Note: Examples are available in test samples /Samples/

List of adapters available by default:

21.1.1 Network Protocols

Adapters	Agents	Descriptions
ARP	socket	Sniffer to send and receive ARP packets
ICMP	socket	Sniffer to send and receive ICMP packets
Ethernet	socket	Sniffer for sending and receiving Ethernet frames
IP	socket	Sniffer for sending and receiving IPv4 packets
Pinger	not supported	Machine life tests via ICMP, TCP or URL
UDP / TCP	socket	Sniffer and UDP client and TCP
NTP	socket	Client to request an NTP server
DNS	not supported	Resolver Customer
SNMP	socket	Receiving SNMPv2 Alarms

21.1.2 Network Protocols Applications

21.1.3 System commands

Adapters	Agents	Descriptions
Dig		Customer dig
Curl		Customer curl
Nmap		Nmap client
Ncat		Customer ncat
Openssl		Openssl client

21.1.4 User Interfaces

Adapters	Agents	Descriptions
Adb	adb	Integration with the Android Gateway
Selenium	selenium2-server or selenium3-server	Integration with the Selenium project
Sikuli	sikulix-server	Integration with the SikuliX project

21.1.5 Data base

Adapters	Agents	Descriptions
Microsoft SQL	database	Communication with a base of type Microsoft SQL
MySQL	database	Communication with a MySQL/MariaDB database
PostgreSQL	database	Communication with a PostgreSQL database

21.1.6 System controls

21.1.7 Telecom Protocols

Adapters	Agents	Descriptions
SMS Gateway	gateway-sms	Receive or send SMS using an Android smartphone
SIP	socket	SIP Phone
RTP	socket	Module for sending and receiving audio and video streams

21.2 Bookstores

A library makes it possible to quickly make available functions for

- support data encryption methods
- support existing compression formats
- support authentication functions
- manipulate the different format of date, time and units
- support codecs (XML, JSON, etc ...)
- support data hash functions

A library does not communicate directly with the system to be tested or piloted. It is used:

- directly from the tests
- from the adapters.

Tip: If several adapters need the same functions, it is advisable to factor them in a library.

List of libraries available by default:

21.2.1 Encryption

AES	Encryption or decryption support
Blowfish	Encryption or decryption support
OpenSSL	Execute SSL command
RC4	Encryption or decryption support
XOR	Encryption or decryption support
RSA	RSA Key Generator

Note: An example is available in test samples /Samples/Tests_Libraries/02_Ciphers

21.2.2 Codecs

Base64	Encode or decode in base64 format
Excel	Excel file reading
G711A	Encode or decode the audio codec
G711U	Encode or decode the audio codec
JSON	Encode or decode text in JSON format
XML	Encode or decode text in XML format

Note: An example is available in test samples /Samples/Tests_Libraries/03_Codecs

21.2.3 Compression

GZIP	Compression or decompression in GZIP format
------	---

Note: An example is available in test samples /Samples/Tests_Libraries/09_Compression

21.2.4 Hashing

Checksum	Checksum Generator
HMAC	Creating a hash md5, sha1 and sha256
MD5	Creating a md5 hash
SHA	Creating a hash sha1, sha256 and sha512
CRC32	Checksum Generator

Note: An example is available in test samples /Samples/Tests_Libraries/05_Hashing

21.2.5 Identifiant

SessionID	Session Builder ID
UUIDS	UUID Generator (Universally Unique Identifier)

Note: An example is available in test samples /Samples/Tests_Libraries/07_Identifiers

21.2.6 Média

ChartsJS	Visible graph generator in test reports
DialTones	Tone generator
Image	Manipulation of images
Noise	Noise generator
SDP	Decodes or encodes SDP messages
WavContainer	Creating audio file type WAV
Waves	Simple wave generator

Note: An example is available in test samples `/Samples/Tests_Libraries/04_Media`

21.2.7 Date

Today	Retrieves today's date
-------	------------------------

Note: An example is available in test samples `/Samples/Tests_Libraries/11_Date`

21.2.8 Security

Basic	Decode or encode the authorization
Digest	Decode or encode the authorization
Hmac	Decode or encode the authorization
Oauth	Decode or encode the authorization
Wsse	Decode or encode the authorization
Certificate	Decodes certificates in a readable format
JWT	Decode or encode tokens

Note: An example is available in test samples `/Samples/Tests_Libraries/01_Security`

21.2.9 Time

Timestamp	Generate a timestamp or convert to a readable value
-----------	---

Note: An example is available in test samples `/Samples/Tests_Libraries/06_Time`

21.2.10 Units

Bytes	Convert fromtes to readable
-------	-----------------------------

Note: An example is available in test samples /Samples/Tests_Libraries/08_Units

21.3 Third party tools

The product comes at the base with a number of plugins to interface with other existing tools (defect tracking, test management, etc.).

These plugins can be used directly from a test.

List of supported tools:

Git	Clone / commit file on remote repository
Jira	Ticket creation
HP ALM QC	Test run, ticket creation. Version 12 minimum
ExtensiveAutomation	Test execution, variable creation
Jenkins	Running tests before or after a build
VSphere	VM creation or supression on VMware

Note:

The solution has a REST API, it can be driven also by these tools.

- Jenkins Plugin: <https://wiki.jenkins.io/display/JENKINS/ExtensiveTesting+Plugin>
-

21.3.1 HP ALM

This plugin allows you to export test results in the HP ALM tool. It can be used from an etst to export results without user intervention.

Example of use:

::

HP ALM —> Call REST API —> AND ^ | | v | Execution of the requested test | v + <——
Push the result ——+

Note: An example is available in the test samples /Samples/Tests_Interop/02_HP_QC

21.3.2 Jenkins

This plugin allows to launch a build from the Extensive solution.

Note: An example is available in test samples /Samples/Tests_Interop/06_Jenkins

21.3.3 VSphere

This plugin allows you to control a VMware virtual environment. It can be used for:

- create virtual machines automatically
- remove machines

Note: An example is available in test samples /Samples/Tests_Interop/05_VSphere

21.3.4 ExtensiveTesting

This plugin makes it possible to make a link between several environment (dev, integration, qualification) by allowing to run tests from one environment to another.

Note: An example is available in test samples /Samples/Tests_Interop/03_ExtensiveTesting

21.3.5 Jira

This plugin makes it possible to create tickets following the execution of a test in the tool Jira.

Note: An example is available in test samples /Samples/Tests_Interop/01_Jira

21.3.6 Git

This plugin allows you to recover or push files from a source repository. It can be used as a prerequisite for a test.

Note: An example is available in test samples /Samples/Tests_Interop/04_Git

21.4 Agents

Agents are available from the toolbox. They are to be used together with the adapters

- to communicate with the system to test or control when it is not accessible live by the test server (ex: a web page)
- run a test on several different environments.

Note: The dummy agent is to be used as a basis for developing a new agent.

21.4.1 Network Protocols

socket	Lets you start TCP / UDP sockets
ftp	Connect to an FTP server(s)
database	Queries databases (MySQL, Microsoft SQL and PostgreSQL)
ssh	Connect to machines via SSH or SFTP

21.4.2 Systems

command	Execute system commands on Windows or Linux
file	Allows you to recover files on Windows or Linux systems

21.4.3 Third party tools

sikulix-server	Interactions with heavy applications
selenium3-server	Allows you to control the latest generation web browsers
selenium2-server	Allows you to control web browsers
soapui	Allows you to run SoapUI tests
adb	Allows you to control Android smartphones
gateway-sms	Send or receive SMS

Note: Using the Selenium3-Server agent requires at least Java 8 on the machine.

22.1 The scheduler

22.1.1 Programming of performances

The scheduler present in the server makes it possible to program the execution of the tests in several ways.

- Run the test once in `x_seconds` or `date_time`
- Run the test several times at `date_time`.
- Run the test at each `start_time` at `finish_time`
- Run the test every hour at a specific hour
- Run the test every day at a precise hour
- Run the test once a week the `day` of the week at a specific hour

Schedule the test:

☒ One Time
☐ Successive
☐ Every
☐ Hourly
☐ Daily
☐ Weekly

☒ Start in x second(s):
☐ Start at:

☐ Run without probes
☐ Run without notifications
☒ Do not keep test result on success

Schedule

Cancel

Note: A recursive task will be automatically restarted by the server after a reboot.

22.1.2 Task Management

The following actions are available to manage tasks scheduled by users:

- cancel one or more tasks
- stop one or more tasks
- reprogram one or more tasks
- view the history of the performances.

All of its actions can be done from the heavy client or from the API.

The screenshot displays the Extensive Testing Client 18.0.0 interface. The main window is titled "Extensive Testing Client 18.0.0 - [remote-tests(Common)://Snippets/Do/01_Wait.tux]". The interface includes a menu bar (File, View, Test Editor, Test Execution, Test Logging, Test Conception, Scheduler, Repositories, Shortcuts, Plugins, Get Started, ?) and a toolbar with icons for Tests Results, Repositories, Task Manager, Agents, Probes, Miscellaneous, Counters, and Release Notes.

The main content area is divided into three sections:

- Scheduled:** A table showing scheduled tests. The table has columns: No., Group, Scheduling Type, Project, Name, Next run, Repeat, and Pr. One test is listed: No. 9, Group 0, Scheduling Type "One time at 2019-01-23 21:54:28", Project "Common", Name "Noname7", Next run "2019-01-23 21:54:28", Repeat "Tr", and Pr "Tr".
- Running:** A table showing tests currently running. The table has columns: No., Project, Name, Started at, Author, and Recursive. One test is listed: No. 10, Project "Common", Name "/Snippets/Do/01_Wait", Started at "2018-01-23 21:54:41", Author "admin", and Recursive "False".
- History:** A table showing the history of test runs. The table has columns: Id, Scheduling Type, Project, Name, Sched at, Run start, Run end, Author, Duration (in sec.), and Run Result. Six tests are listed, all with a "COMPLETE" result.

On the right side, there is a **Summary** section with a table showing the status of tests:

Status	Count
Running	1
Waiting	1
History	6
- COMPLETE	6
- ERROR	0
- KILLED	0
- CANCELLED	0

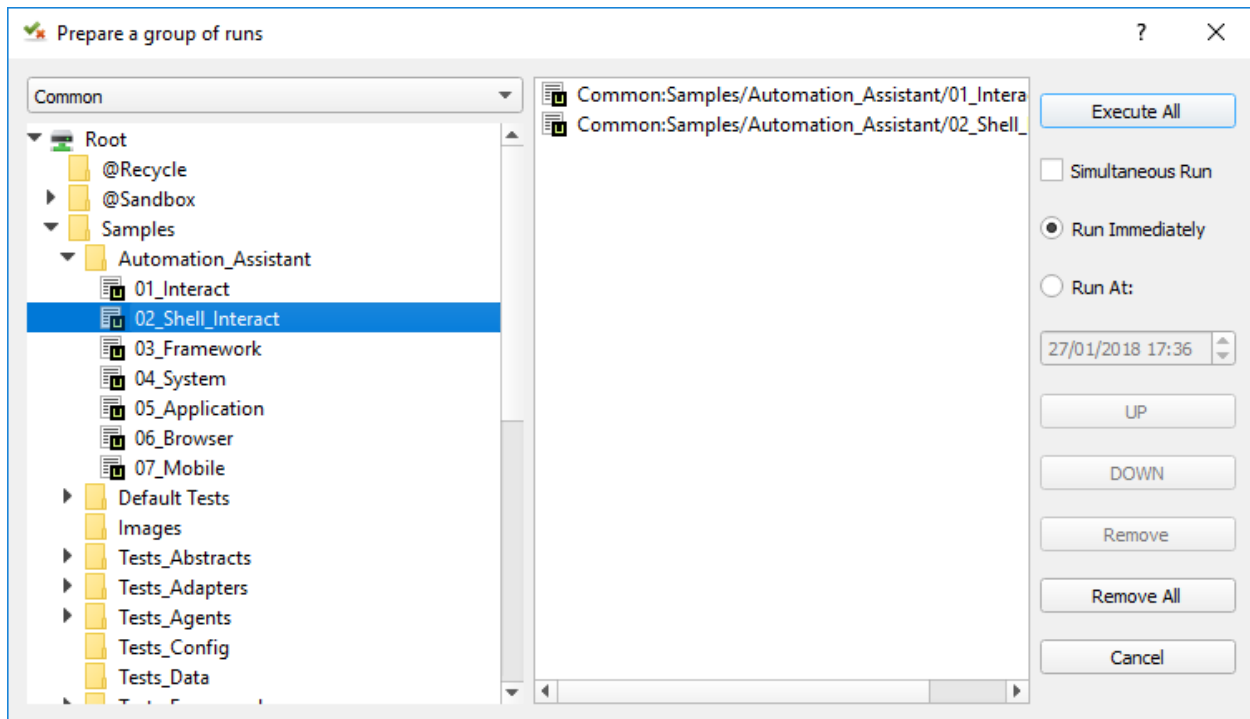
At the bottom, there is a status bar showing "Connected as Administrator | Login: admin | Remote: 10.0.0.240:443 | Proxy: | API: ". The bottom toolbar includes icons for Workspace, Explorer, and two test files: "0001_Rest_Api_0_FAIL_0" and "[10] 01_Wait".

22.2 Parallelized executions

It is possible to run multiple tests in parallel using the Grouped function. This function is available from the heavy client or from the API.

There are 2 options of executions:

- run tests one after the other (no link)
- or parallel execution



Note: From the API, use the function `/rest/tests/schedule/group`.

```
{
  "test": [
    "Common:/Samples/Tests_Unit/02_A.tux",
    "Common:/Samples/Tests_Unit/03_B.tux"
  ],
  "postpone-at": [],
  "parallel-mode": False,
  "postpone-mode": False
}
```

Important: There is no guarantee that the tests will start at the same time with this mode of execution.

22.3 Synchronized executions

22.3.1 Sharing adapters

The “shared mode” feature allows you to reuse the same adapter in several test cases. This mode is to be used in a scenario (test plan) or a test suite with several test cases.

Here is an example of possible use:

- the scenario tests an application
- in the background, the scenario also checks the logs generated by the application
- It is therefore possible to influence the result of the test based on what is found in the logs.

To enable shared mode, set the “shared” parameter to True and give the adapter a name:

```
self.ADP_EXAMPLE = SutAdapters.Dummy.Adapter(  
    parent=self,  
    debug=False,  
    name="MY_ADAPTER",  
    shared=True  
)
```

Note: It is important to give a name to its adapter because it makes finding it easier. If no name is given, the framework configures the adapter with a random name.

After initialization of the adapter it is possible to recover an adapter from another test case by searching for it by name.

```
self.ADP_EXAMPLE = self.findAdapter(name="MY_ADAPTER")  
if self.ADP_EXAMPLE is None: Test(self).interrupt("unable to find the adapter")
```

22.3.2 Sharing data

Since the cache is unique when a test (no matter the type) is performed, it is possible to exchange data between several test cases.

A first test can record data in the cache and a 2nd test can retrieve the value stored by the 1st test.

22.3.3 Synchronization

Synchronized execution of several test cases is possible using a testplan. This scenario should contain:

- an observer test case
- one or more test cases running actions in the background

The observer test must be used to make the connection between the different adapters.

Important: The use of adapters in shared mode is mandatory.

Note: An example is available in the /Samples/Tests_Non_Sequential test samples.

22.4 Distributed executions

The solution allows for distributed executions using distributed agents across the networks.

23.1 SSH adapter

The SSH adapter allows you to connect to remote servers using the SSH protocol.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the remote server port (default 22)
- the user account

The adapter supports the following features:

- authentication by username and password
- key exchange authentication

Example of configuring the adapter in the prepared section of the test.

```
self.ADP_SSH = SutAdapters.SSH.Client(  
    parent=self,  
    login=input('LOGIN'),  
    password=input('PWD'),  
    destIp=input('DEST_IP'),  
    destPort=input('DEST_PORT'),  
    debug=input('DEBUG'),  
    agentSupport=input('SUPPORT_AGENT')  
)
```

Example to connect, to authenticate on a remote server and to disconnect:

```
connected = self.ADP_SSH.doConnect(  
    timeout=input('TIMEOUT'),  
    prompt='~]#'  
)
```

(continues on next page)

(continued from previous page)

```
if not connected: self.abort("ssh connect failed")
self.info("SSH connection OK" )

disconnected = self.ADP.doDisconnect(timeout=input('TIMEOUT'))
if not disconnected: self.abort("disconnect failed")
self.info("SSH disconnection OK" )
```

Example to send a command on a remote machine:

```
rsp = self.ADP_SSH. doSendCommand(
    command='date',
    timeout=input('TIMEOUT'),
    expectedData=None,
    prompt='~]#'
)
if rsp is None: self.abort("run command failed")
self.warning( rsp )
```

Warning: SSH replies can be split into several events (this depends on the network). We must be careful when waiting for a specific response, the use of a buffer may be necessary in this case.

Note: Examples are available in the /Samples/Tests_Adapters/05_SSH.tsx sample.

23.2 HTTP adapter

The HTTP adapter is used to send requests and inspect associated responses to a web server.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the remote server port (default 80)

The adapter supports the following features:

- encryption tls of the communication
- the use of socks4, 5 proxy and http
- digest or basic authentication
- reassembly of responses chunked

Example of configuring the adapter in the prepared section of the test.

```
self.ADP_HTTP = SutAdapters.HTTP.Client(
    parent=self,
    debug=input('TRACE'),
    destinationIp=input('DST_IP'),
    destinationPort=input('DST_PORT'),
    sslSupport = input('SSL_SUPPORT'),
    agent=input('AGENT_SOCKET'),
```

(continues on next page)

(continued from previous page)

```

        agentSupport=input('SUPPORT_AGENT')
    )

```

Example to send a GET type query and a response with the 200 code.

```

rsp = self.ADP_HTTP.GET(
    uri="/",
    host=input('HOST'),
    timeout=input('TIMEOUT'),
    codeExpected=200
)

if rsp is None:
    self.step1.setFailed(actual="bad response received")
else:
    self.step1.setPassed(actual="http response OK")

```

Example to send a GET type query and wait for a response that meets the following criteria:

- the version must end with 1.1
- the code must not contain the value 200
- the sentence must not contain the text *Testing*
- the body of the answer must contain the text *google*
- the response must contain a header containing the text *server*, regardless of the value

```

headersExpected = { TestOperators.Contains(needle='server'): TestOperators.Any() }

rsp = self.ADP_HTTP.GET(
    uri="/",
    host=input('HOST'),
    timeout=input('TIMEOUT'),
    versionExpected=TestOperators.EndsWith(needle='1.1') ,
    codeExpected=TestOperators.NotContains(needle='200') ,
    phraseExpected=TestOperators.NotContains(needle='Testing') ,
    bodyExpected=TestOperators.Contains(needle='google') )
    headersExpected=headersExpected
)

if rsp is None:
    self.step1.setFailed(actual="bad response received")
else:
    self.step1.setPassed(actual="http response OK")

```

23.3 Telnet adapter

The Telnet adapter is used to connect to machines with a telnet interface.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the remote server port (default 23)

Example of configuring the adapter in the prepared section of the test.

```
self.ADP_TELNET = SutAdapters.Telnet.Client(
    parent=self,
    destIp=input('TELNET_IP'),
    destPort=input('TELNET_PORT'),
    debug=input('DEBUG'),
    agentSupport=input('SUPPORT_AGENT')
)
```

Example to connect or disconnect from the remote server

```
self.ADP_TELNET.connect()
connected = self.ADP_TELNET.isConnected( timeout=input('TIMEOUT') )
if not connected: Test(self).interrupt( 'unable to connect' )

self.ADP_TELNET.disconnect()
disconnected = self.ADP_TELNET.isDisconnected( timeout=input('TIMEOUT') )
if not disconnected: Test(self).interrupt( 'unable to disconnect' )
```

Example showing how to wait for the receipt of a particular text.

```
rsp = self.ADP_TELNET.hasReceivedData(
    timeout=input('TIMEOUT'),
    dataExpected=TestOperators.Contains(needle='Password:') )
if rsp is None: Test(self).interrupt( 'Password prompt not found' )
```

Example to send data to the remote server

```
tpl = self.ADP_TELNET.sendData(dataRaw="example")
```

search for a particular text. To guard against this problem, we must add an intermediary buffer, there is a complete example with the Catalyst adapter.

Note: An example is available in the test samples /Samples/Tests_Adapters/12_Telnet.tsx.

23.4 MySQL adapter

The MySQL adapter allows you to connect to a remote database.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the remote server port (by default 3306)
- the user name
- the associated password

Example of configuring the adapter in the prepared section of the test.

```
self.ADP_MYSQL = SutAdapters.Database.MySQL(
    parent=self,
    host=input('HOST_DST'),
    user=input('MYSQL_LOGIN'),
```

(continues on next page)

(continued from previous page)

```

password=input('MYSQL_PWD'),
debug=input('DEBUG'),
verbose=input('VERBOSE'),
agent=input('AGENT_DB'),
agentSupport=input('SUPPORT_AGENT')
)

```

Example to connect or disconnect from the remote server:

```

self.ADP_MYSQL.connect(dbName=input('MYSQL_DB'), timeout=input('TIMEOUT'))

self.ADP_MYSQL.disconnect()

```

Example to execute an SQL query in the database:

```

query = 'SELECT id FROM `%s-users` WHERE login="admin"' % input('TABLE_PREFIX')
self.ADP_MYSQL.query(query=query)
rsp = self.ADP_MYSQL.hasReceivedRow(timeout=input('TIMEOUT'))

```

Note: An example is available in the /Samples/Tests_Adapters/15_Database.tsx test samples.

23.5 SNMP adapter

The SNMP adapter allows you to receive SNMP v1 or v2 alarms.

The configuration of the adapter consists of indicating at least:

- the listening address
- the listening port

Example of configuring the adapter in the prepared section of the test.

```

self.ADP_SNMP = SutAdapters.SNMP.TrapReceiver(
    parent=self,
    bindIp=get('SRC_IP'),
    bindPort=get('SRC_PORT'),
    debug=get('DEBUG'),
    agent=input('AGENT_SOCKET'),
    agentSupport=input('SUPPORT_AGENT')
)

```

Example to start listening to the server

```

self.ADP_SNMP.startListening()
listening = self.ADP_SNMP.udp().isListening( timeout=get('TIMEOUT') )
if not listening: Test(self).interrupt( 'UDP not listening' )

```

Example to wait for the reception of an alarm:

```

trap = self.UDP_ADP.hasReceivedTrap(
    timeout=input('TIMEOUT'),
    version=SutAdapters.SNMP.TRAP_V1,

```

(continues on next page)

(continued from previous page)

```

        community=None,
        agentAddr=None,
        enterprise=None,
        genericTrap=None,
        specificTrap="17",
        uptime=None,
        requestId=None,
        errorStatus=None,
        errorIndex=None
    )
    if trap is None: Test(self).interrupt("trap expected not received")

```

Note: An example is available in the /Samples/Tests_Adapters/18_SNMP.tsx test samples.

23.6 FTP adapter (s)

The FTP adapter allows you to connect to remote servers and supports the following functions:

- TLS connection
- Download or recover files or directories
- Add / delete and rename files or directories
- List the contents of a directory
- Detect the appearance of a file or directory with the support of regular expressions.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the username to login
- the password

Example of configuring the adapter in the prepared section of the test.

```

self.ADP_FTP = SutAdapters.FTP.Client(
    parent=self,
    debug=input('DEBUG'),
    destinationIp=input('FTP_HOST'),
    user=input('FTP_USER'),
    password=input('FTP_PWD') ,
    agentSupport=input('SUPPORT_AGENT')
)

```

Example to connect or disconnect from the FTP server:

```

self.ADP_FTP.connect(passiveMode=True)
if self.ADP_FTP.isConnected(timeout=input('TIMEOUT')) is None:
    Test(self).interrupt("unable to connect")

self.ADP_FTP.login()
if self.ADP_FTP.isLogged(timeout=input('TIMEOUT')) is None:

```

(continues on next page)

(continued from previous page)

```
Test(self).interrupt("unable to login")
Trace(self).info("SFTP connection OK" )
```

```
self.ADP_FTP.disconnect()
if self.ADP_FTP.isDisconnected(timeout=input('TIMEOUT')) is not None:
    Test(self).interrupt("disconnect failed")
Trace(self).info("FTP disconnection OK" )
```

Example to list the contents of a directory:

```
self.ADP_FTP.listingFolder()
if self.ADP_FTP.hasFolderListing(timeout=input('TIMEOUT')) is not None:
    Trace(self).error("unable to get listing folder")
```

Example to detect a file in a directory with a regular expression:

```
self.ADP_FTP.waitForFile(
    path='/var/log/',
    filename='^messages-.*$',
    timeout=input('TIMEOUT')
)

found = self.ADP_FTP.hasDetectedFile(
    path=None,
    filename=None,
    timeout=input('TIMEOUT')
)
if found is None: Trace(self).error("file not found")
```

Note: An example is available in the test samples /Samples/Tests_Adapters/21_Ftp.tsx.

23.7 SFTP adapter

The SFTP adapter allows you to connect to servers with an SSH interface. The following features are supported:

- Download or recover files or directories
- Add / delete and rename files or directories
- List the contents of a directory
- Detect the appearance of a file or directory with the support of regular expressions.

The configuration of the adapter consists of indicating at least:

- the ip address of the remote server
- the username to login
- the password

Example of configuring the adapter in the prepared section of the test.

```
self.ADP_SFTP = SutAdapters.SFTP.Client(
    parent=self,
    login=input('LOGIN'),
    password=input('PWD'),
    destIp=input('DEST_IP'),
    destPort=input('DEST_PORT'),
    debug=input('DEBUG'),
    agentSupport=input('SUPPORT_AGENT')
)
```

Example to connect and disconnect from the server:

```
connected = self.ADP_SFTP.doConnect(timeout=input('TIMEOUT'))
if not connected: Test(self).interrupt("sftp connect failed")
self.info("SFTP connection OK" )

disconnected = self.ADP_SFTP.doDisconnect(timeout=input('TIMEOUT'))
if not disconnected: Test(self).interrupt("disconnect failed")
self.info("SFTP disconnection OK" )
```

Example to list the contents of a directory:

```
self.ADP_SFTP.listingFolder(
    path="/var/log/",
    extended=False
)

rsp = self.ADP_SFTP.hasFolderListing(timeout=input('TIMEOUT'))
if rsp is None: Trace(self).error("unable to get listing folder")
self.warning( rsp.get("SFTP", "result") )
```

Example to detect a file in a directory with a regular expression:

```
self.ADP_SFTP.waitForFile(
    path='/var/log/',
    filename='^messages-.*$',
    timeout=input('TIMEOUT')
)

found = self.ADP_SFTP.hasDetectedFile(
    path=None,
    filename=None,
    timeout=input('TIMEOUT')
)
if found is None: Trace(self).error("file not found")
```

Note: An example is available in the test samples /Samples/Tests_Adapters/22_Sftp.tsx.

23.8 ChartJS librairies

The ChartJs adapter, based on the javascript library of the same name, allows you to generate graphics that can be integrated into an html page. The main interest of this library is to be able to integrate graphs in the

test report.

Example configuration of the library in the prepared section of the test.

```
self.LIB_CHART = SutLibraries.Media.ChartJS(parent=self, name=None, debug=False)
```

Example to generate a bar chart and integrate it into the report

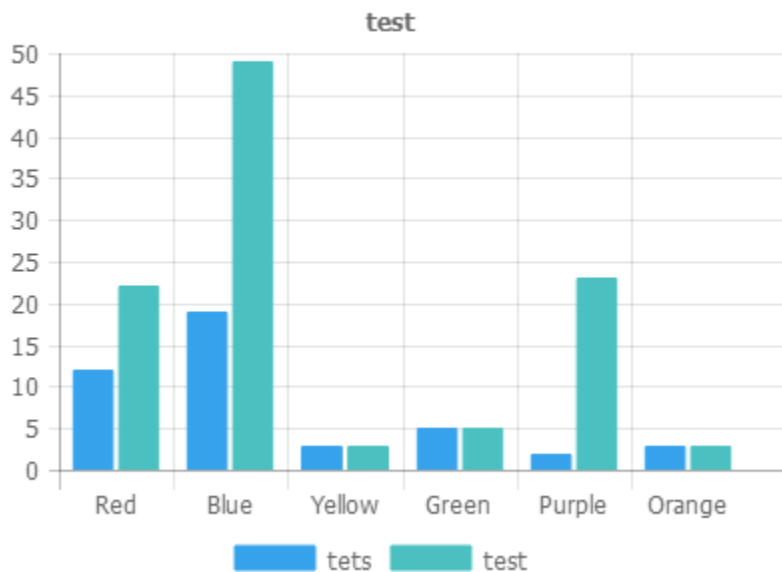
```
# génération de données
labelsAxes = ["Red", "Blue", "Yellow", "Green", "Purple", "Orange"]
dataA = [12, 19, 3, 5, 2, 3]
dataB = [22, 49, 3, 5, 23, 3]
legendDatas = ["tets", "test"]
backgroundColor = '#4BC0C0'
borderColor = '#36A2EB'

# génération du graphique
myChart = self.LIB_CHART.barChart(
    labelsAxes=labelsAxes,
    datas=[dataA, dataB],
    legendDatas=legendDatas,
    width=400,
    height=300,
    backgroundColors=[borderColor, backgroundColor],
    borderColors=[borderColor, backgroundColor],
    chartTitle="test"
)

# ajout du graphique dans le résultat de l'étape
self.step1.setPassed(actual="chart", chart=myChart)
```

The chart is automatically inserted into the advanced report.

1. PASS - chart



23.9 “Text” parameter

The text parameter is used to construct values calling other variables.

For example, consider a test containing the following 2 variables:

- DEST_IP with the value 192.168.1.1
- DEST_PORT with the value 8080

2	DEST_IP	str	192.168.1.1
3	DEST_PORT	int	8080
4	DEST_URL	custom	

The text type will allow us to build a 3rd variable

- DEST_URL with the value

✓✱ Test Config > Custom Values

```
1 https://[!INPUT:DEST_IP:]:[!INPUT:DEST_PORT:]/welcome
```

The keyword [! INPUT: <VARIABLE_NAME:] allows calling another incoming variable. The framework will replace at the time of execution of the test the various keywords with the associated value. We will obtain the value <https://192.168.1.1:8080/welcome> for the variable DEST_URL.

	SECTION	TESTCASE	Preparing
	SECTION	TESTCASE	Starting
	STEP-STARTED	TESTCASE [Step_1]	step sample
	INFO	TESTCASE	https://192.168.1.1:8080/welcome
	STEP-PASSED	TESTCASE [Step_1]	success
	SECTION	TESTCASE	Cleaning
INFO	TESTCASE	END	

To go further, it is also possible to add a value available from the cache. Assuming that the value “welcome? User = hello” is in the cache and accessible via the key “url_params”. It is possible to integrate in the parameter as below

✓✱ Test Config > Custom Values

```
1 https://[!INPUT:DEST_IP:]:[!INPUT:DEST_PORT:]/welcome?[!CACHE:url_params:]
```

Example of result after execution:

SECTION	TESTCASE	Preparing
SECTION	TESTCASE	Starting
STEP-STARTED	TESTCASE [Step_1]	step sample
INFO	TESTCASE	https://192.168.1.1:8080/welcome?user=hello
STEP-PASSED	TESTCASE [Step_1]	success
SECTION	TESTCASE	Cleaning
INFO	TESTCASE	END

23.10 “Json” parameter

todo

23.11 “alias” parameter

The alias parameter can be used to define a new name for an already existing parameter. This mechanism can be used in plan test to avoid overloading all parameters with the same name.

Example of use

1. Before execution

```
Scenario (TIMEOUT_A(int)=2 seconds)
---> Test 1 (TIMEOUT_A(int)=10 seconds)
---> Test 2 (TIMEOUT_A(int)=30 seconds)
---> Test 3 (TIMEOUT_A(int)=20 seconds)
```

2. After running the test

```
Scenario (TIMEOUT_A(int)=2 seconds)
---> Test 1 (TIMEOUT_A(int)=2 seconds)
---> Test 2 (TIMEOUT_A(int)=2 seconds)
---> Test 3 (TIMEOUT_A(int)=2 seconds)
```

When executing the above scenario, test 1, 2 and 3 are automatically set to 2 seconds for the TIMEOUT_A parameter. This is the behavior provided by the test framework.

How to do if you want the test 2 to keep the value 30 seconds against the test 1 and 2 inherit the value of the scenario?

You have to use an alias parameter, they are not overloaded by the framework.

1. Before execution

```
Scenario (TIMEOUT_A(int)=2 seconds et TIMEOUT_B(int)=30 seconds)
---> Test 1 (TIMEOUT_A(int)=10 seconds)
---> Test 2 (TIMEOUT_A(alias)=TIMEOUT_B et TIMEOUT_B(int) = 0 seconds)
---> Test 3 (TIMEOUT_A(int)=20 seconds)
```

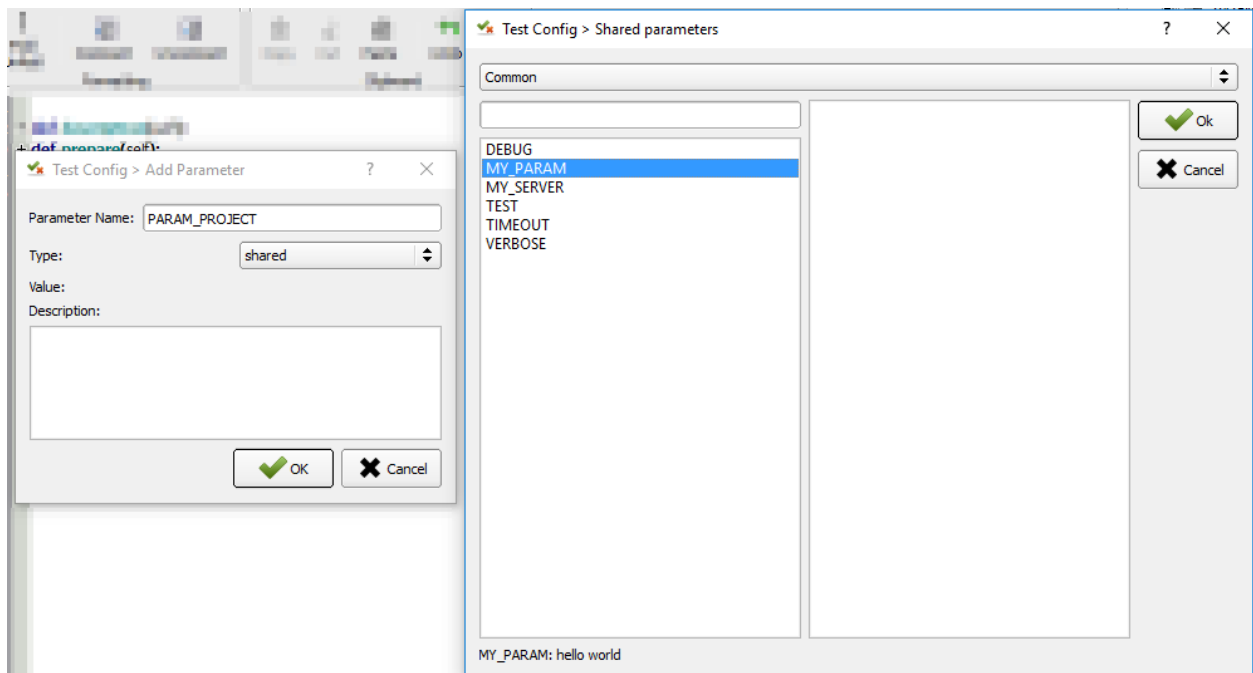
2. After running the test

```
Scenario (TIMEOUT_A(int)=2 seconds et TIMEOUT_B(int)=30 seconds)
--> Test 1 (TIMEOUT_A(int)=2 seconds)
--> Test 2 (TIMEOUT_A(alias)=TIMEOUT_B et TIMEOUT_B(int)= 30 seconds)
--> Test 3 (TIMEOUT_A(int)=2 seconds)
```

23.12 “global” parameter

The global parameters are added from the web interface or from the REST API. They are global and accessible by all the tests in the same project. The expected value for this parameter is of JSON type.

A selection window in the graphical client allows you to select the parameter to be used in the test.



In the example below, the MY_SERVER test parameter contains the value of the IP key present in the variable global MY_SERVER which is itself present in the Common project.

2	MY_SERVER	shared	Common>MY_SERVER>IP
---	-----------	--------	---------------------

Tip: To have a test parameter that contains a list of elements, use the list-global type.

23.13 “Dataset” parameter

The dataset parameter is used to import tdx files. A dataset file is just a text file, it can be created from the graphical client and saved to the remote test repository.



Sample content of a dataset file with the csv format

```
a;1;administrator
b;2;tester
```

This file can be used in a test that is important in the settings.

Inputs			
Outputs			
Adapters			
Id	Name	Type	Value
1	DATA	dataset	1_Dataset

Example to read the variable:

```
for d in input('DATA').splitlines():
    Trace(self).info( d )
```

23.14 Using an agent

To use an agent, you need two things:

- Deploy the toolbox and select the desired agent.
- Declare the agent in the test
- Configure the adapter to use the agent.

Agents are to be declared from the client in the tab Miscellaneous> Agents

Test Properties				
Prepare the test.				
Test Design				
Test Data				
Miscellaneous				
Agents				
Probes				
Id	Name	Type	Value	Description
1	AGENT_SOCKET	socket	agent.win.sock01	

Enabling agent mode on adapters is done with the agentSupport and agent arguments.

```
agentSupport=input('SUPPORT_AGENT'),
agent=input('AGENT_SOCKET')
```

```
self.ADP_REST= SutAdapters.REST.Client(
    parent=self,
    destinationIp=input('HOST'),
    destinationPort=input('PORT'),
    debug=input('DEBUG'),
    sslSupport=input('USE_SSL'),
    agentSupport=input('SUPPORT_AGENT'),
    agent=input('AGENT_SOCKET')
)
```

In the analysis window, it is possible to see the agent used for each event:

Arrival Time: 2016-09-09 18:13:48.945

Key	Value
AGENT	
type	selenium
name	agent.win.selenium01
GUI	

Note: It is advisable to put in test parameter the use of the agent mode.

Inputs				
Outputs				
Adapters				
Libraries				
Id	Name	Type	Value	Description
5	SUPPORT_AGENT	bool	False	

Requirements

24.1 Server

For now the server can only be run on a Linux environment. It can be run on a virtual or physical environment.

Features	Minimum	Recommended
OS	Linux	
Python	2.7	
Arch	x86_64	
Memory	4GB	8 GB
CPU	2 core	4 core
Disk	~10GB	~50GB
Network	100Mb/s	1 Gbit/s
User Number	2	5

Note: Python 3.x support is under development.

24.2 Customer

The client can be run on a Windows or Linux environment.

Features	Minimum	Recommended
OS	Windows 7+ Linux CentOS 6.5+	Windows 10+ CentOS7+ or Ubuntu 17+
Arch	x86_64	
Memory	4GB	8GB
Disk	~1GB	~2GB

Note: The 32-bit architecture is no longer supported since version 17.0.0. However it is still possible to compile the sources on a 32bits environment.

Important: The plugins for the client are only available for the Windows environment.

24.3 Toolbox

The toolbox can be run on a Windows or Linux environment.

Features	Minimum	Recommended
OS	Windows 7+ Linux	Windows 10+ Linux
Arch	x86_64	
Memory	4GB	8GB
Disk	~1GB	

Note: The 32-bit architecture is no longer supported since version 17.0.0. However it is still possible to compile the sources on a 32bits environment.

Important: The plugins for the client are only available for the Windows environment.

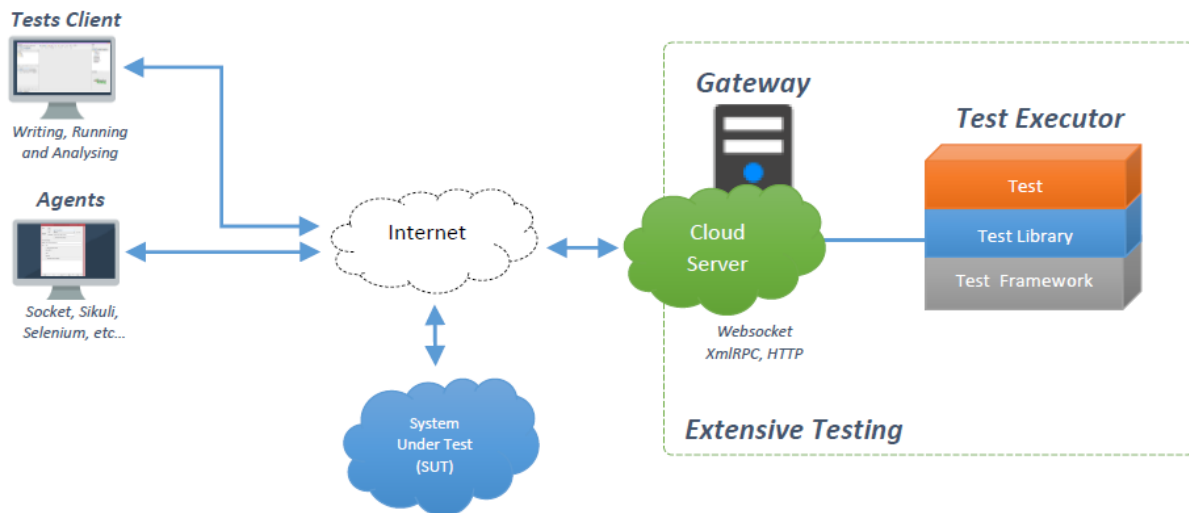
CHAPTER 25

Architecture

The solution is based on a client/server mode. The tests and adapters are centralized in a server that allows to quickly provide the same test environment to all users.

The solution consists of several components:

- A scheduler
- A graphical client
- Agents



25.1 Server

The server consists of:

- a reverse proxy (apache)
- a scheduler
- a REST API server
- the test framework
- adapters
- tool extensions
- a web interface

25.2 Graphic Client

The graphical client uses a single tcp/443 (https) stream between the client and the server. The stream is bidirectional and the client can:

- make calls to the server's REST API
- receive events from the server via WebSockets.

25.3 Agents

An agent is compulsorily controlled by an adapter via the test server intermediary. It allows to deport the communication point with the system to test or control. Agents use a single tcp/443 (https) stream to communicate with the server.

26.1 Version cycle

The set of software packages for the solution follows the following rules for naming versions.

The version is divided into 3 digits (A.B.C)

- **A: the 1st digit indicates the major version. Incrementation of this figure implies**
 - the addition of major features (with potentially a loss of compatibility with the previous version)
 - adding minor features
 - bug fix
- **B: The 2nd digit indicates a minor version. Incrementing this number indicates**
 - adding minor features
 - bug fix
- **C: the 3rd digit indicates a maintenance version. Incrementing this number indicates**
 - bug fix

26.2 Server tree

All files handled by the server are stored in the [...]Var/ directory.

```
Build/  
ServerEngine/  
ServerControls/  
ServerInterfaces/  
ServerRepositories/
```

(continues on next page)

(continued from previous page)

```

Libs/
TestCreatorLib/
TestExecutorLib/
TestInterop/
Var/
  Tests/
  TestsResults/
  SutAdapters/
  Logs/
  Backups/

```

The tests are stored in the [...]Var/Tests/ directory, they are organized by project ID.

26.3 Data model

A database is used by the server to store:

- the users of the solution
- the list of projects
- test data (project variables)
- statistics
- the history of executions

Tables	Description
xtc-config	Server Configuration
xtc-projects	List of projects
xtc-relations-projects	Relationship between projects and users
xtc-users	List of users
xtc-test-environment	List of variables in JSON format
xtc-tasks-history	History of tasks running on the server

26.4 Passwords management

No password (in plain text) is stored in the database. Using a hash is however used. The hash of the password is stored in the *xtc-users* table.

The algorithm used:

```
MariaDB [xtc112]> select id, login, password from 'xtc-users';
```

id	login	password
1	system	146aeec808258c699abf36fb9cb81697833a946f
2	admin	c9188bd805ea6c0018469b5bf106fb38c3cb9f88
3	leader	c9188bd805ea6c0018469b5bf106fb38c3cb9f88
4	developer	c9188bd805ea6c0018469b5bf106fb38c3cb9f88
5	tester	c9188bd805ea6c0018469b5bf106fb38c3cb9f88
6	automaton	c9188bd805ea6c0018469b5bf106fb38c3cb9f88

26.5 File format

The tests are in XML format. There are several test formats:

- Xml Test Unit
- Xml Test Suite
- Xml Test Plan
- Global Xml Test

Common XML Structure

```
<?xml version="1.0" encoding="utf-8" ?>
<file>
  <properties>
    <descriptions>...</descriptions>
    <inputs-parameters>...</inputs-parameters>
    <outputs-parameters>...</ outputs -parameters>
  </properties>
</file>
```

Test Unit Xml

```
<?xml version="1.0" encoding="utf-8" ?>
<file>
  <properties>...</properties>
  <testdefinition><![CDATA[pass]]></testdefinition>
  <testdevelopment>1448190694.813723</testdevelopment>
</file>
```

Test Suite Xml

```
<?xml version="1.0" encoding="utf-8" ?>
<file>
  <properties>...</properties>
  <testdefinition><![CDATA[pass]]></testdefinition>
  <testexecution><![CDATA[pass]]></testexecution>
  <testdevelopment>1448190717.236711</testdevelopment>
</file>
```

Test Plan Xml

```
<?xml version="1.0" encoding="utf-8" ?>
<file>
  <properties>...</properties>
  <testplan id="0">
    <testfile>
      <id>1</id>
      <color />
      <file>Common:Defaults/testunit.tux</file>
      <enable>2</enable>
      <extension>tux</extension>
      <alias />
      <type>remote</type>
      <parent>0</parent>
      <properties>...</properties>
      <description />
    </testfile>
  </testplan>
</file>
```

(continues on next page)

(continued from previous page)

```

    </testfile>
  </testplan>
  <testdevelopment>1448190725.096519</testdevelopment>
</file>

```

Test Global Xml

```

<?xml version="1.0" encoding="utf-8" ?>
<file>
  <properties>...</properties>
  <testplan id="0">
    <testfile>
      <id>1</id>
      <color />
      <file>Common:Defaults/testplan.tpx</file>
      <enable>2</enable>
      <extension>tpx</extension>
      <alias />
      <type>remote</type>
      <parent>0</parent>
      <properties>...</properties>
      <description />
    </testfile>
  </testplan>
  <testdevelopment>1448190733.690697</testdevelopment>
</file>

```

26.6 Storage of test results

The test results are stored on the server in the [...]Var/TestsResult directory.

The results are stored:

- by the id of the test projects
- by the date of the day of execution of the test
- and finally by the date and time of the tests.

Organization of the results:

```

Répertoire: <project_id>
- Répertoire: <yyyy-mm-dd>
- Répertoire: <yyyy-mm-dd_hh:mm:ss.testid.testname.username>
  - Fichier: TESTPATH
  - Fichier: test.out
  - Fichier: test.ini
  - Fichier: <testname>_<replayid>.hdr
  - Fichier: <testname>_<replayid>_<result>_<nbcomments>.trv
  - Fichier: <testname>_<replayid>.tbrp
  - Fichier: <testname>_<replayid>.tdsx
  - Fichier: <testname>_<replayid>.trd
  - Fichier: <testname>_<replayid>.trp
  - Fichier: <testname>_<replayid>.trpx
  - Fichier: <testname>_<replayid>.trv
  - Fichier: <testname>_<replayid>.trvx

```


Description of files:

- TESTPATH contains the full path for the test result
- test.out contains the internal logs of the test, to be used to debug the test framework
- test.ini contains test-specific parameters
- <testname>_<replayid>.hdr represents the header of the test result
- <testname>_<replayid>_<result>_<nbcomments>.trv contains all the events generated during the execution of the tests
- <testname>_<replayid>.tbrp contains the basic report in html format
- <testname>_<replayid>.trp contains the full report in html
- <testname>_<replayid>.trv contains the results report in csv format

26.7 Control Agents

The control of the agents since a test is carried out through:

- the adapters
- and the server

The communication takes place with the exchange of some specific messages:

- init: allows to initialize an agent
- notify: send a message to the agent without waiting for a response
- reset: allows to reset the agent
- error: allows the agent to send an error to the adapter
- data: allows the agent to send data to the adapter

Direction of available communications:

- Agent -> server -> adapter -> test
- Test -> adapter -> server -> agent

	Agent	
	Function	Callback
Send an error message	def sendError * request * data	
Send a “notify” message	def sendNotify * request * data	
Send a “data” message	def sendData * request * data	
Receiving an “init” message		def onAgentInit * customer * tid * request
Receiving a “reset” message		def onAgentNotify * customer * tid * request
Receiving a “notify” message		def onAgentReset * customer * tid * request

	Adapter	
	Function	Callback
Receiving an error message		def receivedErrorFromAgent * data
Receiving a “notify” message		def receivedNotifyFromAgent * data
Receiving a “data” message		def receivedDataFromAgent * data
Send an “init” message	def initAgent * data	
Send a “reset” message	def resetAgent	
Send a “notify” message	def sendNotifyToAgent * data	

26.8 The server logs

The server logs are located in the [...] /Var/logs/ directory.

output.log	server logs
------------	-------------

27.1 Solution development

27.1.1 Qt client application

Please refer to the `'README < https://github.com/ExtensiveAutomation/extensiveautomation-appclient/blob/master/README.md>'` _

27.1.2 Toolbox

Please refer to the `'README < https://github.com/ExtensiveAutomation/extensiveautomation-apptoolbox/blob/master/README.md>'` _

27.1.3 Server

Please refer to the [README](#)

27.2 Plugins development for servers

Please refer to the [README](#)

27.3 Plugins development for qt client and agents

Please refer to the [README](#)

27.4 Documentation

The documentation is stored on github in the *repository* <<https://github.com/ExtensiveAutomation/extensiveautomation.readthedocs.org>>. It is possible to contribute by applying for participation in the deposit.

The documentation is generated by the *readthedocs* <<https://readthedocs.org/>> _ service.

28.1 Authentication

Authentication in the API REST can be done with 2 methods:

- By using the function login to obtain a session cookie
- with basic auth

28.1.1 Basic

The basic auth must be used with the api key available through the web interface. The re-generation of the api key can be done for now only on the server.

```
./extensiveautomation --apikey admin  
API Key ID: admin  
API Key Secret: d30278d49e4845e45daa748873e2171b14a0c55a
```

After that, add the header Authorization in your HTTP request.

```
Authorization: Basic base64(key_id:key_secret)
```

Note: With the basic auth, it's not necessary to call the login function.

28.1.2 Session cookie

The authentication by cookie can be done by calling the function login to generate a cookie for the session. This cookie must be present on all next http request in the header Cookie.

Cookie: `session_id=NjQyOTVmOWNlMDgyNGQ2MjlkNzAzNDdjNTQ3ODU5MmU5M`

28.2 Usage example

The REST api is available through the tcp port 443 (https) with the uri /rest.

The following example show how to execute a test with the basic auth.

```
POST /rest/tests/schedule HTTP/1.1
[...]
Authorization: Basic YWRtaW46N2UwMDExY2I3Y2ZhMGQ1MjM4NGQ1YWYyM2QyODBiMjUyM2EzMTA3ZA==
Content-Type: application/json; charset=utf-8
[...]

{
  "project-id": 1,
  "test-extension": "tux",
  "test-name": "01_Wait",
  "test-path": "/Snippets/Do/",
  "test-inputs": [ {"name": "DURATION", "type": "int", "value": 5} ]
}
```

Received response:

```
{
  "cmd": "/tests/schedule",
  "message": "background"
  "test-id": "a3f19398-b463-41e8-9e43-af86aac44a59",
  "task-id": 17,
  "tab-id": 0
  "test-name": "01_Wait"
}
```

28.3 Ressources

Description of most important functions:

Authentication

Execute a test

Read the result of a test executed