ExtensiveAutomation Documentation

Version 20.0

ExtensiveAutomation

août 10, 2019

A propos de

1 Extensive Automation **1**
- 1.1 Concepts . . . 1
- 1.2 Usages . . . 2
- 1.3 Licence . . . 2
- 1.4 Auteur . . . 2
- 1.5 Contact . . . 3

2 Téléchargement **5**

3 Notes de version **7**
- 3.1 Version courante . . . 7
- 3.2 Versions précédentes . . . 7

4 Client lourd **13**
- 4.1 L'espace de travail . . . 14
- 4.2 L'analyseur . . . 21
- 4.3 L'explorateur . . . 23
- 4.4 Préférences de configuration . . . 24
- 4.5 Compléments . . . 25

5 Boite à outils **29**
- 5.1 Déploiement . . . 30
- 5.2 Compléments . . . 31

6 Interface Web **33**
- 6.1 Partie tests . . . 33
- 6.2 Partie administration . . . 33

7 Première utilisation **35**
- 7.1 Connexion du client au serveur . . . 35
- 7.2 Écriture d'un test (script) . . . 36
- 7.3 Ecriture d'un scénario (conditionnel) . . . 36
- 7.4 Exécution d'un test . . . 37
- 7.5 Analyse des résultats . . . 38
- 7.6 Les bonnes pratiques . . . 39

8 Exemples de tests **41**

8.1 Cas de test (unit) . . . 41
8.2 Cas de test (suite) . . . 41
8.3 Variables de test . . . 42
8.4 Scénario . . . 43
8.5 Campagne de tests . . . 44
8.6 Rest API . . . 44
8.7 Contrôles SSH . . . 46
8.8 Navigateurs Internet . . . 46
8.9 Mobile Android . . . 49

9 Tests réutilisables **53**
9.1 Données partagées . . . 53
9.2 Actions basiques . . . 55
9.3 Générateurs . . . 58
9.4 Protocoles réseaux . . . 59
9.5 Interface utilisateur . . . 63
9.6 Vérifications . . . 64

10 Variables globales **67**
10.1 Ajout/suppression d'une variable . . . 67
10.2 Description environnement de test . . . 67
10.3 Import/export des variables . . . 68

11 Conception assistée **69**
11.1 Onglet Framework . . . 70
11.2 Onglet Système . . . 70
11.3 Onglet Application . . . 71
11.4 Onglet Navigateur . . . 73
11.5 Onglet Android . . . 75

12 Dépannage **79**
12.1 Code erreurs . . . 79
12.2 FAQ . . . 79

13 Installation **81**
13.1 Serveur . . . 81
13.2 Plugins pour le serveur . . . 81
13.3 Client Web . . . 81
13.4 Client lourd . . . 81
13.5 Boîte à outils . . . 81

14 Administration **83**
14.1 Arrêt/relance du serveur . . . 83
14.2 Status du serveur . . . 83
14.3 Configuration du serveur . . . 84
14.4 Scripts crontab . . . 84

15 Projets **85**
15.1 Ajout d'un projet . . . 85
15.2 Suppression d'un projet . . . 85
15.3 Associer un projet à un utilisateur . . . 85

16 Utilisateurs **87**
16.1 Ajout d'un utilisateur . . . 87
16.2 Suppression d'un utilisateur . . . 87

17 Dépannage **89**
17.1 Récupération des logs . . . 89
17.2 Foire aux questions . . . 91

18 Les types de tests **93**
18.1 Test Unit . . . 93
18.2 Test Suite . . . 94
18.3 Test Plan . . . 94
18.4 Test Global . . . 94

19 Les fondamentaux **95**
19.1 Cas de test . . . 95
19.2 Etapes de test . . . 95
19.3 Annulation d'un test . . . 96
19.4 Ajout de trace . . . 97
19.5 Données . . . 97
19.6 Mettre en attente . . . 99
19.7 Interaction avec le testeur . . . 99
19.8 Paramètres d'un test . . . 100

20 La traçabilité **105**
20.1 Les évènements . . . 105
20.2 Les rapports de tests . . . 108
20.3 Les logs . . . 112

21 L'interopérabilité **113**
21.1 Adaptateurs . . . 113
21.2 Librairies . . . 115
21.3 Outils tiers . . . 118
21.4 Agents . . . 119

22 Le moteur d'exécutions **121**
22.1 L'ordonnanceur . . . 121
22.2 Exécutions parallélisées . . . 123
22.3 Exécutions synchronisées . . . 124
22.4 Exécutions distribuées . . . 125

23 Exemples avancés **127**
23.1 Adaptateur SSH . . . 127
23.2 Adaptateur HTTP . . . 128
23.3 Adaptateur Telnet . . . 129
23.4 Adaptateur MySQL . . . 130
23.5 Adaptateur SNMP . . . 131
23.6 Adaptateur FTP(s) . . . 132
23.7 Adaptateur SFTP . . . 133
23.8 Librairie ChartJS . . . 134
23.9 Paramètre de tests « text » . . . 135
23.10 Paramètre de tests « json » . . . 136
23.11 Paramètre de tests « alias » . . . 136
23.12 Paramètre de tests « global » . . . 137
23.13 Paramètre de tests « dataset » . . . 138
23.14 Utilisation d'un agent . . . 139

24 Prérequis systèmes **141**
24.1 Serveur . . . 141

24.2 Client . . . 141
24.3 Boite à outils . . . 142

25 Architecture **143**
25.1 Serveur . . . 143
25.2 Client Graphique . . . 144
25.3 Agents . . . 144

26 Spécifications **145**
26.1 Cycle des versions . . . 145
26.2 Arborescence du serveur . . . 145
26.3 Modèle de données . . . 146
26.4 Gestion des mots de passes . . . 146
26.5 Format des fichiers . . . 146
26.6 Stockage des résultats de tests . . . 148
26.7 Contrôle Agents . . . 149
26.8 Les logs serveurs . . . 150

27 Contributions **151**
27.1 Développement client lourd . . . 151
27.2 Plugins côté serveur . . . 151
27.3 Plugins côté client et agents . . . 151
27.4 Documentations . . . 152

28 API **153**
28.1 Authentification . . . 153
28.2 Exemple d'utilisation . . . 154
28.3 Ressources . . . 154

CHAPITRE 1

Extensive Automation

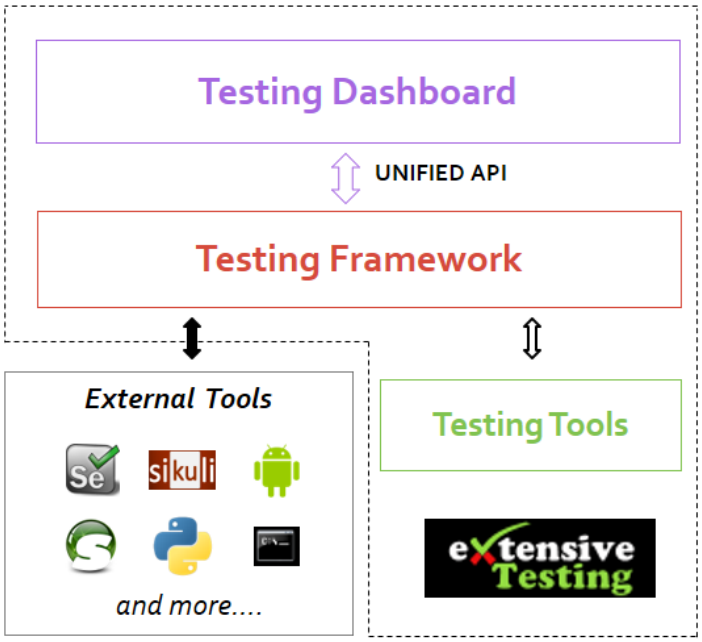
Bienvenue sur le projet Extensive Automation.

1.1 Concepts

L'ojectif de la solution est de fournir un environnement de création et d'exécution de tâches automatiques

Le projet a plusieurs objectifs depuis sa création :

- Unifier les différents outils de test dans un seul environnement
- Simplifier l'écriture, l'exécution et l'analyse des tâches automatiques
- Partager les tests automatiques et permettre l'exécution depuis n'importe où
- Fournir un environnement de travail agréable
- Fournir un tableau de bord



1.2 Usages

La solution couvre plusieurs usages et peut être utilisée pour les cas suivants :

- automatiser les tests dans les environnements d'intégration
- automatiser les tests de non-régression et fonctionnels
- automatiser les tests de bout-en-bout
- automatiser le déploiement de serveurs
- automatiser le déploiement d'applications

Note : La solution est développée en `Python` ainsi que l'ensemble des tests.

Note : Il est possible d'intégrer facilement vos scripts existants en python (ou autre) dans la solution.

1.3 Licence

L'environnement est open source et sous licence `LGPL 2.1`. Le code source est disponible sur github (https://github.com/ExtensiveAutomation).

1.4 Auteur

Le projet a été initialisé en 2010. Il est dévelopé et maintenu par `Denis Machard`.

1.5 Contact

Pour entrer en contact :

- par email d.machard@gmail.com
- depuis twitter https://twitter.com/Extensive_Auto
- depuis le forum google https://groups.google.com/forum/#!forum/extensive-automation-users
- depuis github https://github.com//ExtensiveAutomation

CHAPITRE 2

Téléchargement

Une version complète de la solution est générée tous les 2 à 3 mois environ. Elle contient l'ensemble des composants nécessaires au bon fonctionnement du serveur. La solution est faite de plusieurs composants pouvant être téléchargés séparément ou de manière intégrée.

La solution peut être téléchargée depuis le site internet https://www.extensiveautomation.org

Le serveur est disponible sour plusieurs formes :
- version depuis les sources directement
- image docker
- fichier tar.gz

Le client web est disponible sous deux formes :
- version compilée
- version depuis les sources directement

Le client lourd est disponible sous deux formes :
- version portable
- version en mode installation

La boîte à outils est disponible sous trois formes :
- version portable
- version en mode installation
- version non compilée, en mode ligne de commande

Note : Le client et la boîte à outils sont compatibles Windows et Linux, en mode 64bits seulement.

Avertissement : Le serveur doit être exécuté sur un serveur Linux avec Python.

Avertissement : Le client ou la boîte à outils en mode installation peuvent nécessiter des droits d'administration.

CHAPITRE 3

Notes de version

3.1 Version courante

Note : Version 20.0.0 disponible depuis le 20/07/2019

— Image docker disponible
— Rest API : CORS support
— Procédure d'installation automatique du produit supprimée
— Suppression d'un maximum de dépendances
— Plus de plugins embarqués par défaut
— Les sondes sont supprimés du produit, utilisation des agents à la place
— Supression de la base de donnée MySQL, remplacer par une base Sqlite
— Optimisation du framework de test pour réduire la consommation CPU
— Nouveau version majeure du client lourd, simplification de l'interface
— Nouvelle version majeure de la boite à outils
— Nouvelle version majeure de l'interface web totalement indépendante, non fournie par défaut.
— Corrections de bugs divers

La release notes détaillée est disponible dans le paquet d'installation.

3.2 Versions précédentes

Version 19.0.1 sortie le 09/08/2018

— Changement du nom de produit de la solution par ExtensiveAutomation
— Fichiers stockés en mode XML par défaut (plus de compression)
— Améliorations et corrections diverses au niveau de l'API REST
— Support initial docker
— Python 2.6 n'est plus supporté côté serveur
— Prévisualisation du cache durant l'écriture des tests dans le client
— Simplification des paramètres de tests avec les types « text » et « json »

— Optimisation du nombre de requête SQL sur le serveur
— Support initial de python 3.5 côté serveur
— Le client lourd n'est plus embarqué sur le serveur par défaut
— Nouvelle fonction permettant d'ajouter un message de sécurité sur la page de connexion
— Mise à jour de selenium en 3.13.0 dans la boite à outils
— Nouvelle version majeure pour le client lourd
— Nouvelle version majeure pour la boite à outils
— Correction bug deploiement serveur, utilisation de la commande pip

Version 18.0.0 sortie le 11/02/2018
— Suppression de l'API XmlRPC sur le serveur
— Refonte complète de l'API REST
— Nouveau client majeur utilisant l'api rest exclusivement
— Support de Qt5.9 pour le client et la boite à outils
— Support de python 3.6 pour le client et la boite à outils
— Nettoyage important de l'ensemble du code source
— Corrections de bugs divers
— Mise à jour de selenium en 3.9.0 dans la boite à outils
— La boite à outils n'est plus embarquée par défaut sur le serveur

Version 17.1.0 sortie le 22/10/2017
— Amélioration de l'API REST
— Ajout de fonctionnalités majeures dans le framework de test
— Apport de corrections
— Amélioration de l'interface graphique sur le client
— Support expérimental du client sur Ubuntu

Version 17.0.0 sortie le 04/06/2017
— Passage en mode 64bits par défaut pour le client et la boite à outils
— Ajout de fonctionnalités majeures dans le framework de test
— Mise à disposition d'un swagger pour l'API rest
— Mise à jour de selenium 3 et 2 dans la boite à outils
— Apport de corrections

Version 16.1.0 sortie le 30/03/2017
— Apport de corrections
— Amélioration de l'interface graphique sur le client
— Amélioration de l'installation

Version 16.0.0 sortie le 25/02/2017
— Apport de corrections
— Amélioration de l'api REST
— Modifications diverses dans le coeur du serveur
— Ajout de fonctionnalités dans le framework de test
— Optimisation du nombre de requêtes SQL sur le serveur
— Amélioration de l'interface graphique sur le client
— Support 64bits du client et de la boite à outils

Version 15.0.3 sortie le 04/11/2016
— Apport de corrections
— Nouveau plugins pour le client
— Amélioration de l'api REST
— Ajout de fonctionnalités dans le framework de test
— Ajout du module intéropérabilité

Version 14.0.0 sortie le 27/08/2016
— Apport de corrections
— Ajout de fonctionnalités dans le framework de test
— Amélioration majeures de l'API REST
— Plus d'évolutions dans l'API XmlRPC côté serveur, correction de bugs seulement
— Ajout de fonctionnalités dans l'interface web
— Python2.7 n'est plus supporté sur windows pour le client et la boite à outils
— Utilisation de l'api REST au niveau du client
— Amélioration de l'interface graphique sur le client
— Nouveau plugin HP QC ALM

Version 13.0.0 sortie le 23/06/2016
— Apport de corrections
— Ajout API REST sur le serveur
— Ajout de fonctionnalités dans le framework de test
— Améliorations diverses dans le coeur du serveur
— Support des plugins pour le client et à la boite à outils
— Amélioration de l'interface graphique sur le client

Version 12.1.0 sortie le 29/04/2016
— Apport de corrections
— Ajout de fonctionnalités dans le framework de test
— Quelques modifications au niveau l'API XmlRPC
— Amélioration de l'interface graphique sur le client

Version 12.0.0 sortie le 12/02/2016
— Apport de corrections
— Ajout de fonctionnalités au niveau l'API XmlRPC
— Ajout de fonctionnalités dans le framework de test
— Ajout de fonctionnalités dans l'interface web

Version 11.2.0 sortie le 22/11/2015
— Apport de corrections
— Ajout de fonctionnalités dans le framework de test
— Amélioration de l'ordonnanceur
— Ajout d'un dépôt public utilisé par le framework de test
— Support installation sans accès internet
— Modification mineures dans l'API XmlRPC

Version 11.1.0 sortie le 18/10/2015
— Apport de corrections
— Ajout de fonctionnalités au niveau l'API XmlRPC
— Ajout de fonctionnalités dans l'interface web

Version 11.0.0 sortie le 14/09/2015
— Apport de corrections
— Ajout de fonctionnalités dans l'interface web
— Fusion des agents et sondes dans la boite à outils
— Modifications au niveau de l'API XmlRPC
— Support de python 3.4 pour le client et la boite à outils

Version 10.1.0 sortie le 12/07/2015
— Apport de corrections
— CentOS 4 et 5 ne sont plus supportés officiellement

— Ajout de fonctionnalités dans le framework de test
— Ajout de fonctionnalités dans l'interface web

Version 10.0.0 sortie le 28/05/2015
— Apport de corrections
— Ajout de fonctionnalités dans l'interface web
— Modifications diverses dans le coeur du serveur
— Mise à jour des documentations
— Amélioration de l'interface graphique sur le client

Version 9.1.0 sortie le 22/03/2015
— Apport de corrections
— Ajout de fonctionnalités dans le framework de test
— Amélioration de l'installation du produit
— Amélioration de l'interface graphique sur le client

Version 9.0.0 sortie le 05/01/2015
— Apport de corrections
— Ajout de fonctionnalités dans le framework de test
— Python 2.4 n'est plus supporté
— Ajout de fonctionnalités dans l'interface web
— Amélioration de l'interface graphique sur le client

Version 8.0.0 sortie le 25/10/2014
— Apport de corrections
— Amélioration de l'interface graphique sur le client
— Ajout de fonctionnalités dans le framework de test
— Modifications mineures au niveau de l'API XmlRPC
— Ajout de fonctionnalités dans l'interface web

Version 7.1.0 sortie le 20/09/2014
— Apport de corrections
— Mise à jour documentations
— Optimisation pour réduire le temps de construction d'un test sur le serveur
— Ajout de fonctionnalités dans le coeur du serveur
— Ajout de fonctionnalités dans le framework de test
— Amélioration de l'interface graphique sur le client

Version 7.0.0 sortie le 08/08/2014
— Apport de corrections
— Amélioration de l'ordonnanceur
— Ajout d'apache en mode reverse sur le serveur
— Support des websockets activé par défaut
— Ajout de documentations
— Communication des composants unifiées sur le port tcp/443 ssl
— Support proxy SSL
— Utilisation SSL par défaut sur les agents et sondes
— Amélioration de l'interface graphique sur le client

Version 6.2.0 sortie le 02/06/2014
— Apport de corrections
— Mise à jour des agents
— Modifications mineures au niveau de l'API XmlRPC
— Ajout de fonctionnalités dans le framework de tests

— Modifications au niveau de l'ordonnanceur

Version 6.1.0 sortie le 25/04/2014
— Apport de corrections
— Ajout de fonctionnalités dans l'interface web
— Ajout de fonctionnalités dans le framework de tests
— Amélioration du module agents

Version 6.0.0 sortie le 23/03/2014
— Apport de corrections
— Nouveau mode de paquetage pour les adaptateurs et librairies
— Ajout de fonctions dans l'API XmlRPC
— Ajout de fonctionnalités dans le framework de tests
— Supression de la dépendance avec le projet twisted
— Support SSL activé par défaut pour l'API XmlRPC
— Support proxy socks4
— Support des agents

Version 5.2.0 sortie le 12/01/2014
— Apport de corrections
— Ajout de fonctionnalités mineures

Version 5.1.0 sortie le 08/12/2013
— Ajout de fonctionnalités dans l'interface web
— Apport de corrections
— Ajout de fonctionnalités dans le framework de tests

Version 5.0.0 sortie le 15/09/2013
— Apport de corrections
— Ajout majeurs de fonctionnalités dans le framework de tests
— Amélioration dans l'ordonnanceur

Version 4.2.0 sortie le 08/04/2013
— Apport de corrections
— Ajout de fonctionnalités dans l'interface web

Version 4.1.0 sortie le 10/03/2013
— Apport de corrections
— Ajout de fonctionnalités dans l'interface web
— Support de CentOS 6
— Amélioration dans l'ordonnanceur

Version 4.0.0 sortie le 30/01/2013
— Apport de corrections
— Ajout de fonctionnalités dans le framework de tests
— Support SSL pour l'interface web
— Nouveau mécanisme d'authentification avec salt et sha1
— Ajout de fonctions dans l'API XmlRPC

Version 3.2.0 sortie le 29/09/2012
— Apport de corrections
— Ajout de fonctionnalités dans le framework de tests

Version 3.1.0 sortie le 14/07/2012
— Apport de corrections

— Ajout de fonctionnalités dans le framework de tests
— Amélioration de l'ordonnanceur
— Ajout de fonctions dans l'API XmlRPC

Version 3.0.0 sortie le 09/06/2012
— Apport de corrections
— Ajout de fonctions dans l'API XmlRPC
— Amélioration de l'ordonnanceur
— Nouveau dépôt pour les adaptateurs et sauvegardes

Version 2.2.0 sortie le 28/03/2012
— Ajout de fonctions majeures dans l'API XmlRPC
— Apport de corrections
— Ajout de fonctionnalités dans le framework de tests

Version 2.0.0 sortie le 27/02/2012
— Ajout de fonctions dans l'API XmlRPC
— Ajout de la génération de la documentation du framework et adaptateurs
— Apport de corrections
— Support des sondes

Version 1.2.0 sortie le 14/01/2012
— Amélioration de l'ordonnanceur
— Ajout de fonctions dans l'API XmlRPC
— Ajout de fonctionnalités dans le framework de tests
— Ajout d'une interface web
— Apport de corrections

Version 1.0.0 sortie le 13/12/2011
— 1ière version officielle
— Support CentOS 5
— Apport de corrections

Version 0.1.0 sortie le 17/05/2010
— 1ière version beta

CHAPITRE 4

Client lourd

Le client lourd permet d'écrire et d'exécuter des tests automatiques mais aussi d'analyser les résultats en temps réel ou différé. Il permet aussi de partager les tests de manière simple et efficace. Pour utiliser le client, il faut obligatoirement un compte utilisateur et pouvoir se connecter au serveur de test (tcp/443).

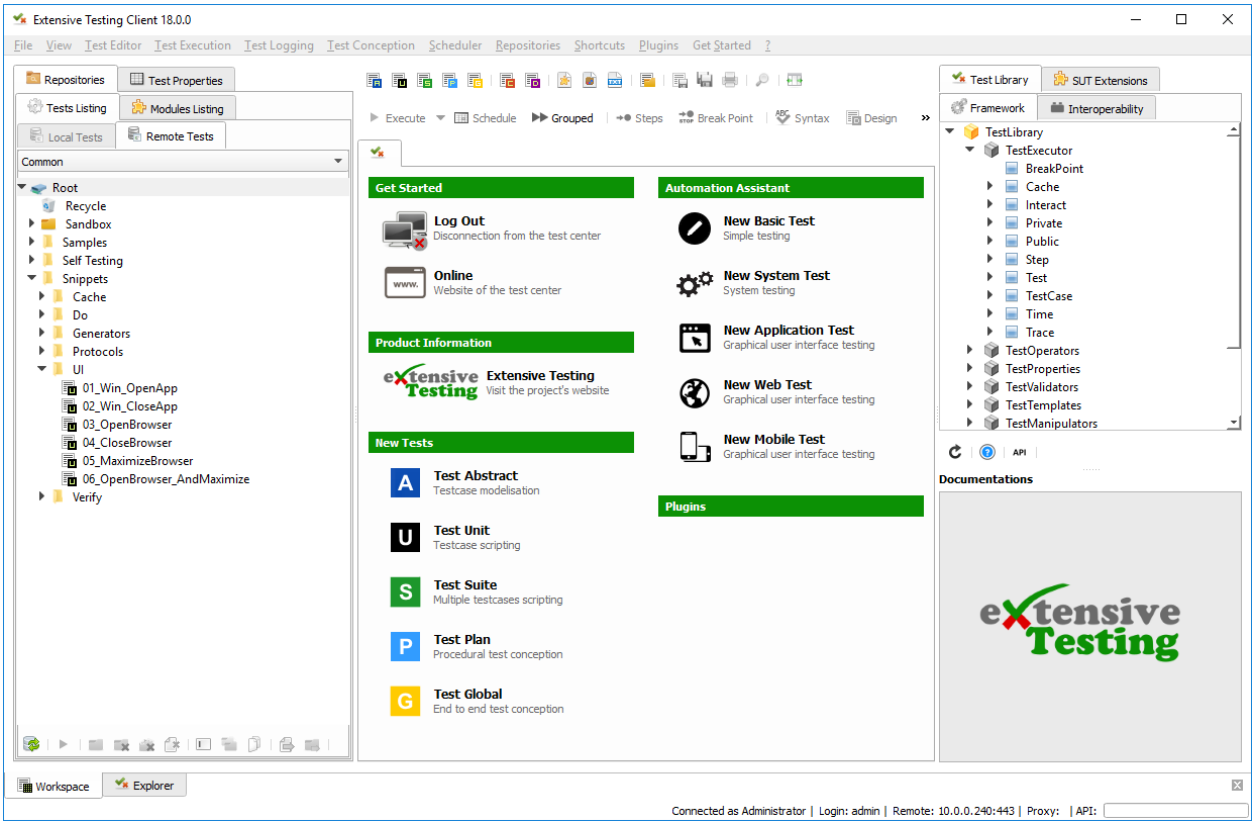
Le client peut aussi être utilisé pour effectuer le développement des extensions (adaptateurs et librairies) permettant de communiquer avec le système à tester ou piloter.

Enfin l'interface graphique change en fonction du niveau d'accès :

- niveau testeur : écriture/exécution de tests, et analyse des résultats
- niveau admin : accès à l'ensemble des fonctionnalités
- niveau monitor : accès en lecture seule

L'interface se divise en 3 parties principales :

- l'espace de travail
- l'analyseur
- l'explorateur

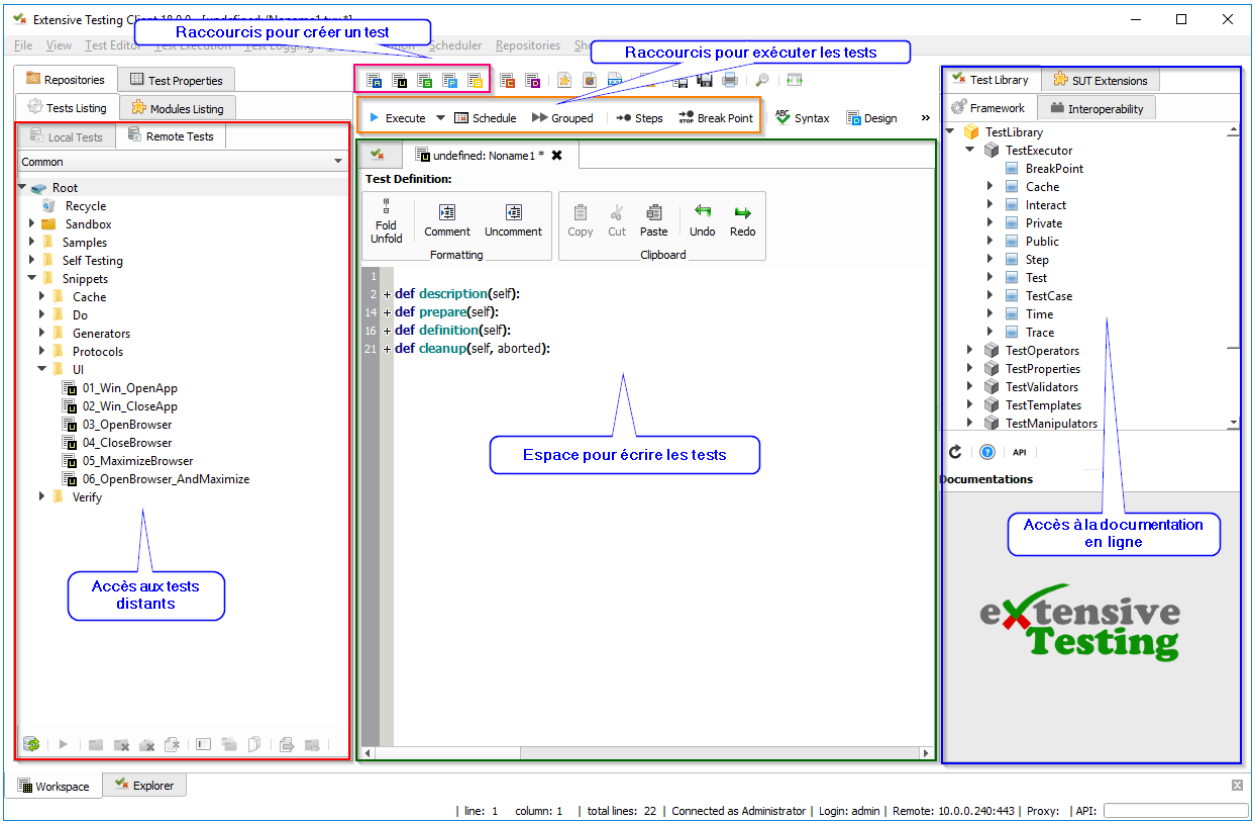


Note : Le client est disponible sur Windows et Linux, en mode 64bits

4.1 L'espace de travail

L'espace de travail se décompose en 3 parties principales :

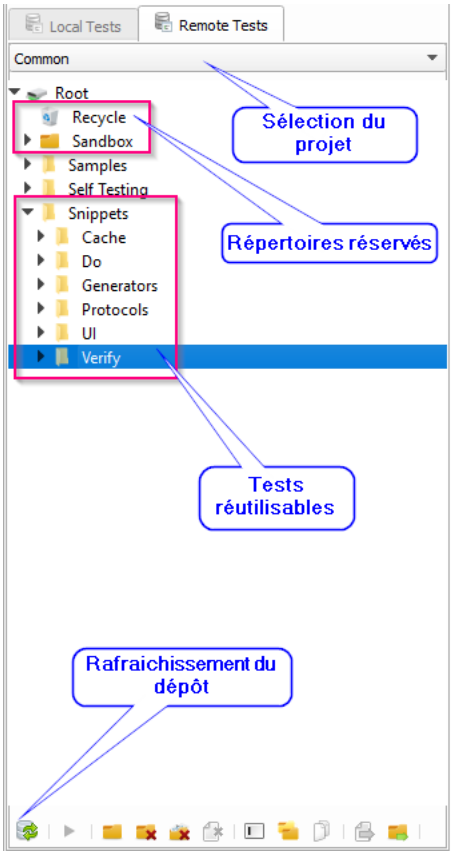
- l'accès à l'ensemble des dépôts de fichiers
- l'accès à la conception des tests
- la documentation en ligne



4.1.1 Dépôt des tests

Le client permet d'accéder aux deux dépôts de tests : distant et local.

Le dépôt `distant` permet de stocker ses tests sur le serveur de tests, donc de les partager avec les autres utilisateurs. L'arborescence se compose de fichiers et répertoires. La gestion des tests peut se faire depuis le client. Les tests peuvent être organisés par projet si nécessaire.



Note : Le projet `Common` contient les tests réutisables et divers exemples.

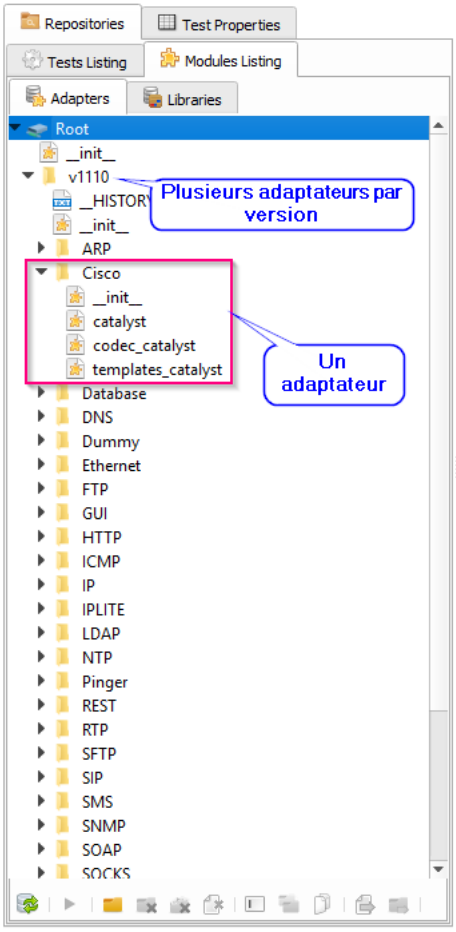
Note : Les répertoires `Recycle` et `Sandbox` sont des répertoires réservés, les supprimer est impossible.

Note : Il est possible d'ouvrir un test en faisant un drag and drop du fichier vers l'espace d'écriture.

Avertissement : Certaines fonctionnalités sont manquantes dans le dépôt local, son utilisation n'est pas conseillée !

4.1.2 Dépôt des extensions

Le client permet l'accès aux dépôts des extensions (adaptateurs) et peut aussi être utilisé pour en développer des nouvelles, qui seront stockées là aussi. Ces extensions sont organisées par version.



Note : Les extensions sont développées en `Python`.

4.1.3 Propriétés d'un test

Les tests peuvent être enrichis avec un certain nombre de propriétés. Les propriétés disponibles sont :

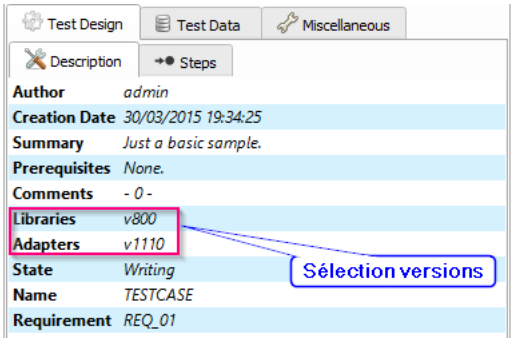
- la description du test (auteur, date de création, etc. . .)
- les variables du test
- la définition des agents et sondes utilisées par le test

La fenêtre `Test properties > Test Data > Inputs` contient la liste des variables accessibles depuis le test. L'ajout de variable peut se faire en faisant un clic droit "Add parameter".



Note : Pour insérer un paramètre dans un test, il suffit de faire un drag & drop.

Note : Il est possible de choisir la version des adaptateurs et librairies à utiliser pour le test

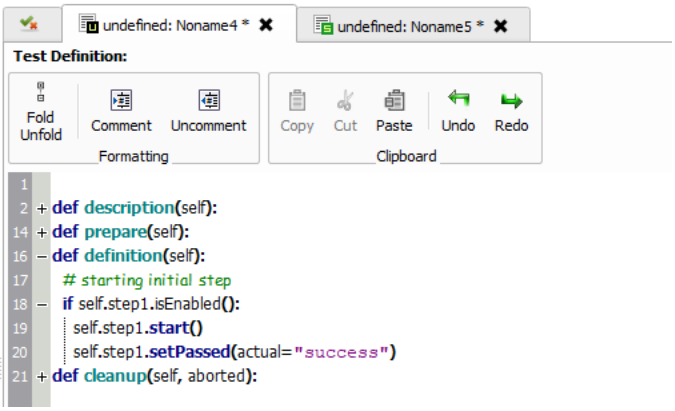


4.1.4 Conception textuelle

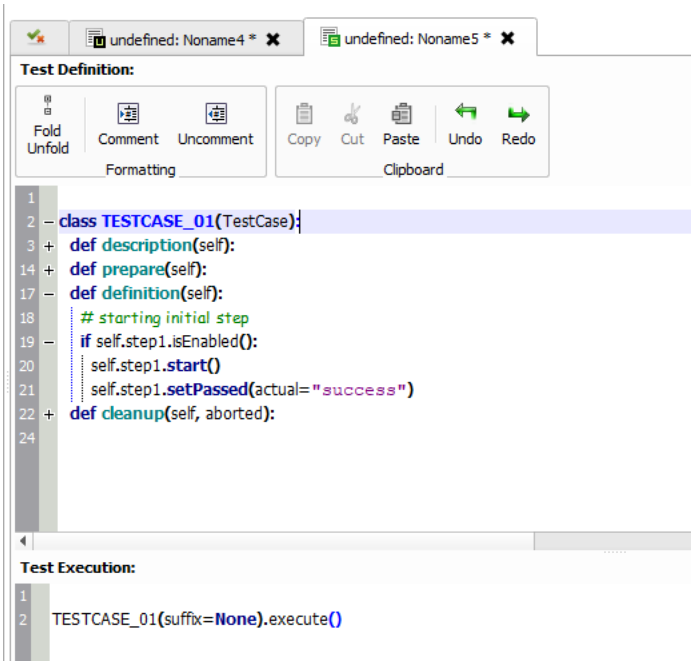
La conception d'un test en mode `scripting` est possible avec led testd de type `unit` et `suite`. Ce mode de conception nécessite des connaissances en développement, i.e. python.



Le test de type `unit` représente un cas de test. Il se découpe en 4 sections appelées automatiquement par le framework.



Le test de type « suite » représente un ou plusieurs cas de test. Ce type de test permet d'exécuter plusieurs fois le même cas de test en changeant les paramètres d'entrées.



Note : Le raccourci `Ctrl+F` permet de rechercher du texte dans vos tests.

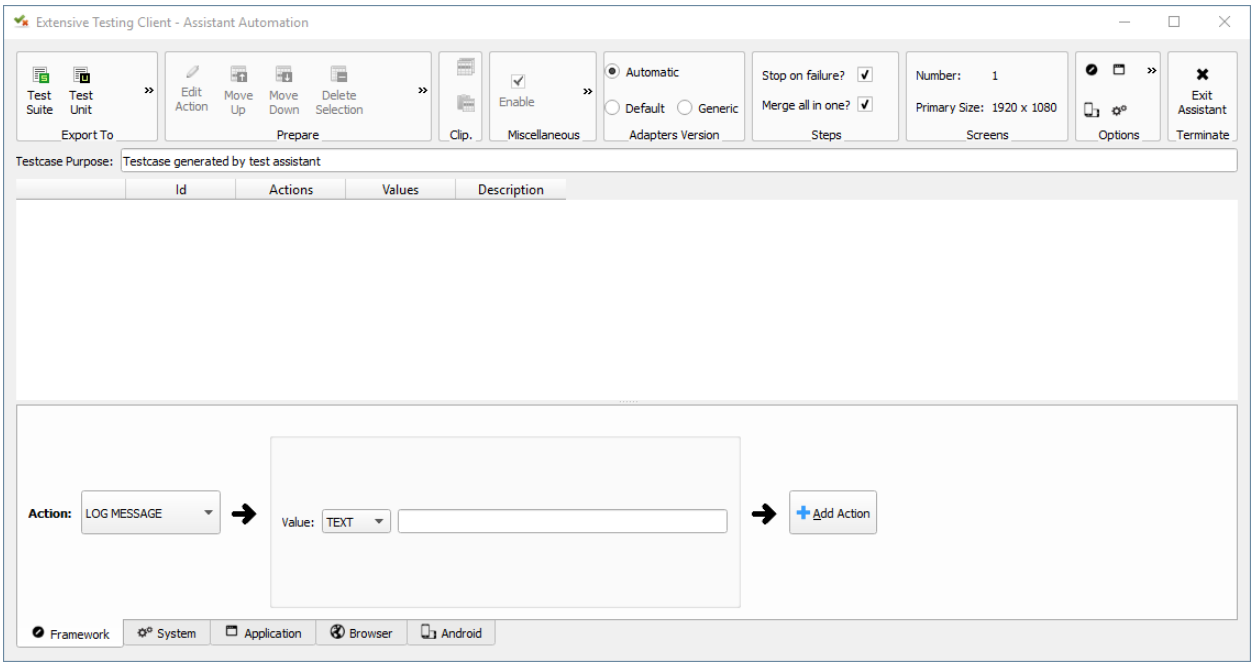
4.1.5 Conception assistée

L'assistant de conception permet d'écrire des tests sans connaissances en développement. Il couvre les différentes actions suivantes :

— Appel aux fonctions de base du framework de test
— Test SSH

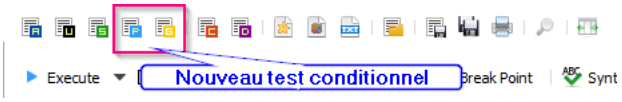
— Test d'application avec capture d'écran (basé sur le projet Sikuli)
— Test de site internet (basé sur le projet Selenium)
— Test d'application mobile Android

L'assistant consiste à décrire les actions à effectuer, et si désiré les exporter vers un test unit ou suite.

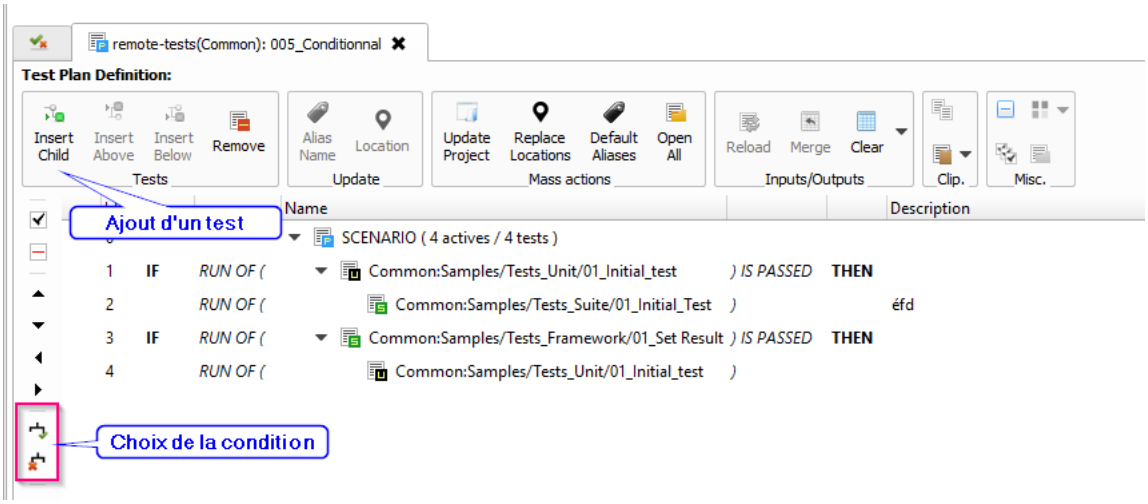


4.1.6 Conception conditionnelle

La conception conditionnelle permet de construire des scénarios ou des campagnes de tests. Cette approche ne nécessite pas de connaissances en développement. Pour réaliser ce type de test, il est nécessaire de créer un nouveau test `plan` ou `global`.



Le test « plan » permet d'écrire des scénarios de test en incluant des tests de type « unit » ou « suite ».

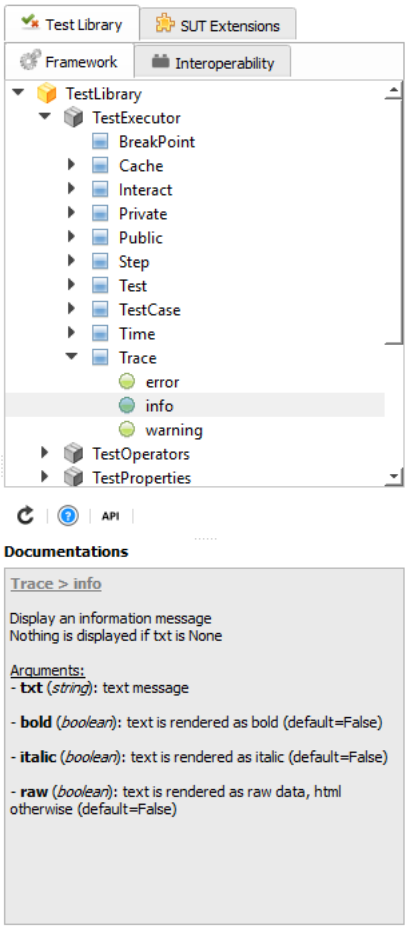


Le test « global » permet de décrire des campagnes de tests en incluant des tests « plan », « unit » ou « suite ».

Note : Il est possible de surcharger les paramètres de tests.

4.1.7 Documentations en ligne

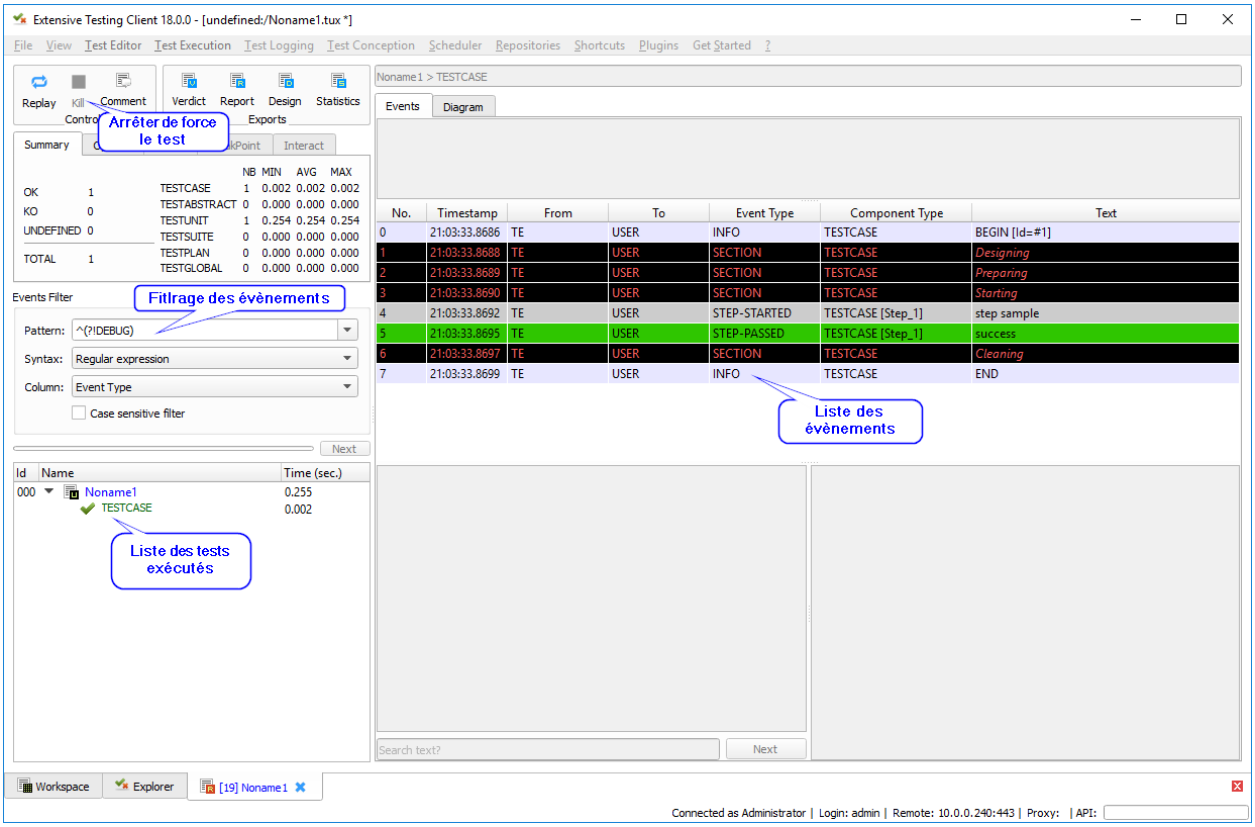
La documentation en ligne est générée par le serveur, elle décrit l'ensemble des fonctions disponibles dans le framework de test et les différentes extensions.



Note : Un drag & drop depuis la documentation sur un test insère automatiquement le squelette de la fonction.

4.2 L'analyseur

L'analyseur permet de suivre l'exécution d'un test en temps réél ou différé. Il permet d'afficher l'ensemble des évènements du test et de faciliter l'analyse du bon déroulement ou des erreurs.



4.2.1 Visualisation des évènements

Différents types d'évènements sont possibles (colonne event type) :

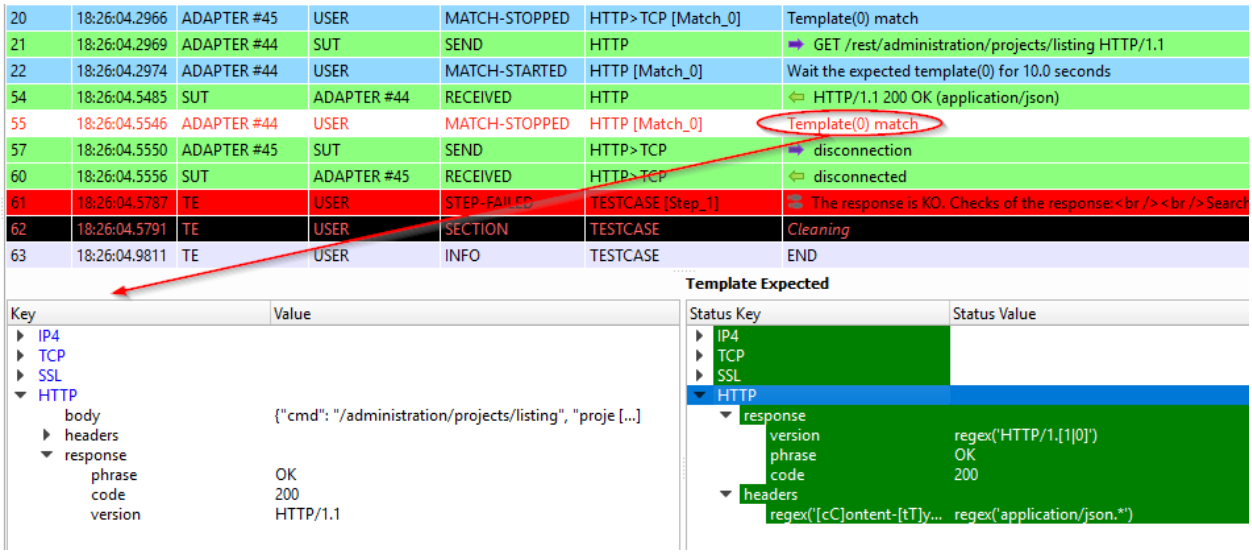
- DEBUG
- INFO
- WARNING
- ERROR
- SEND
- RECEIVED
- STEP-STARTED
- STEP-PASSED
- STEP-FAILED
- MATCH-STARTED
- MATCH-INFO
- MATCH-STOPPED
- MATCH-EXCEEDED

Note : Filtrer sur l'évènement `ERROR` permet de voir rapidement pourquoi le test est en erreur.

Note : Le filtre `SEND|RECEIVED` permet d'afficher les messages envoyés ou reçus par le système à tester/piloter.

4.2.2 Vue détaillée

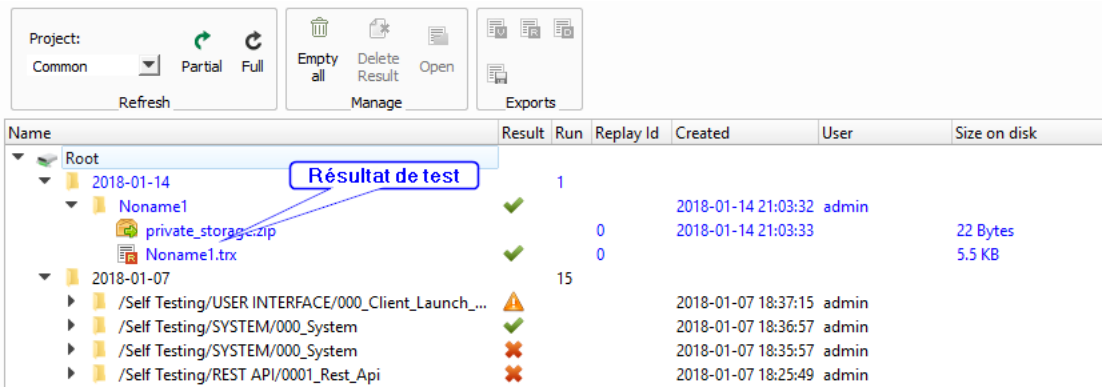
Sélectionner un évènement dans la liste permet d'afficher la vue détaillée. La vue détaillée affiche le contenu de l'évènement et plus encore.



4.3 L'explorateur

4.3.1 Visualisation des résultats

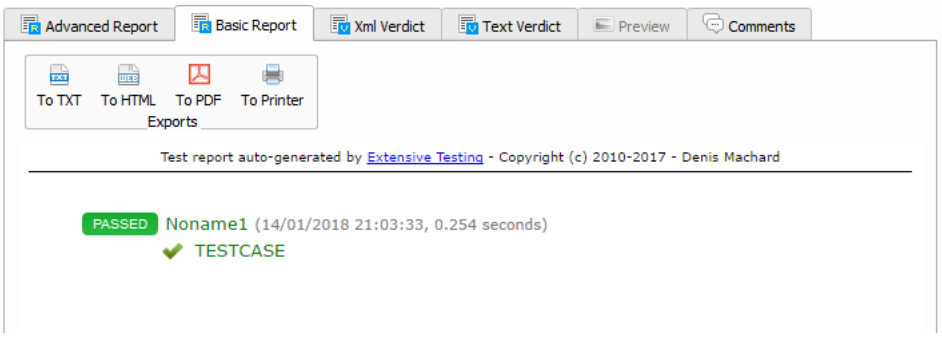
L'historique complet des résultats de tests est disponible depuis le client. Ils sont triés par date et heure d'exécution. Le client permet d'afficher les rapports et télécharger les logs générés durant l'exécution du test.



4.3.2 Visualisation des rapports de tests

Les rapports de tests sont visibles directement depuis le client. Deux types de rapports sont disponibles :

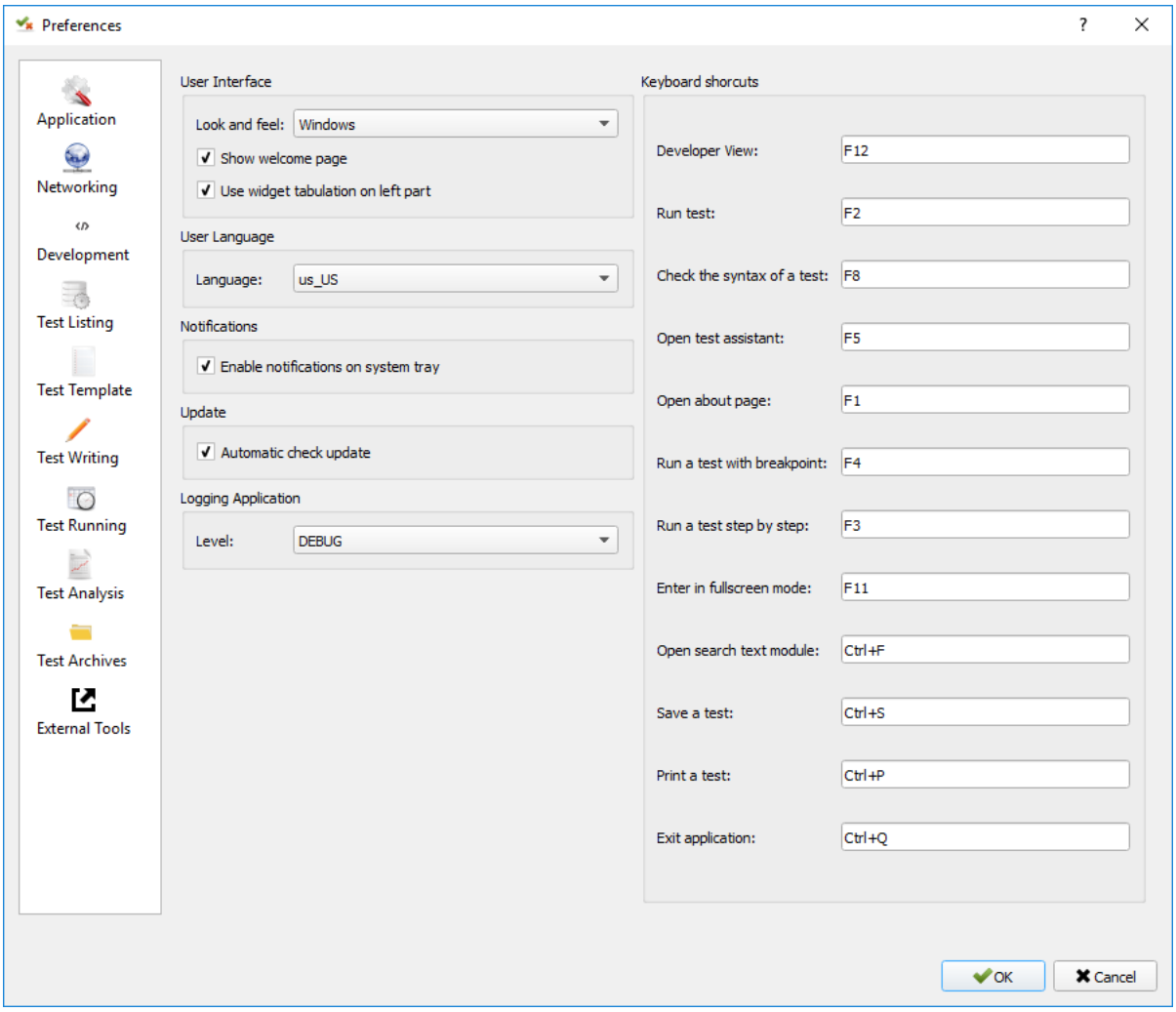
— rapport avancé
— rapport simple



Note : Les rapports sont exportables aux formats html, xml et csv.

4.4 Préférences de configuration

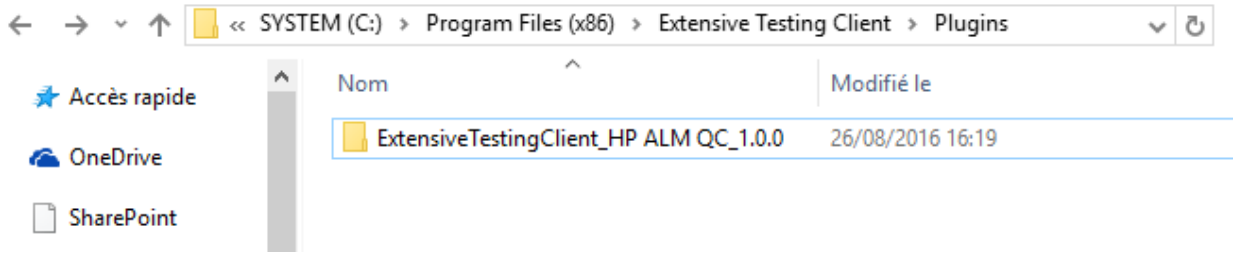
Le comportement du client peut être modifié à travers les préférences du client.



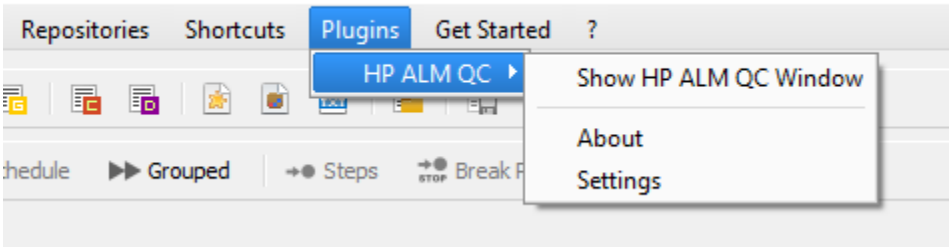
Note : Les préférences sont stockées dans le fichier `settings.ini` .

4.5 Compléments

Il est possible d'ajouter des plugins dans le client. Les plugins sont à ajouter dans le répertoire `Plugins`.



Les plugins sont accessibles dans le menu `Plugins` après redémarrage du client.



Note : Il est nécessaire de redémarrer le client pour prendre en compte les plugins déployés.

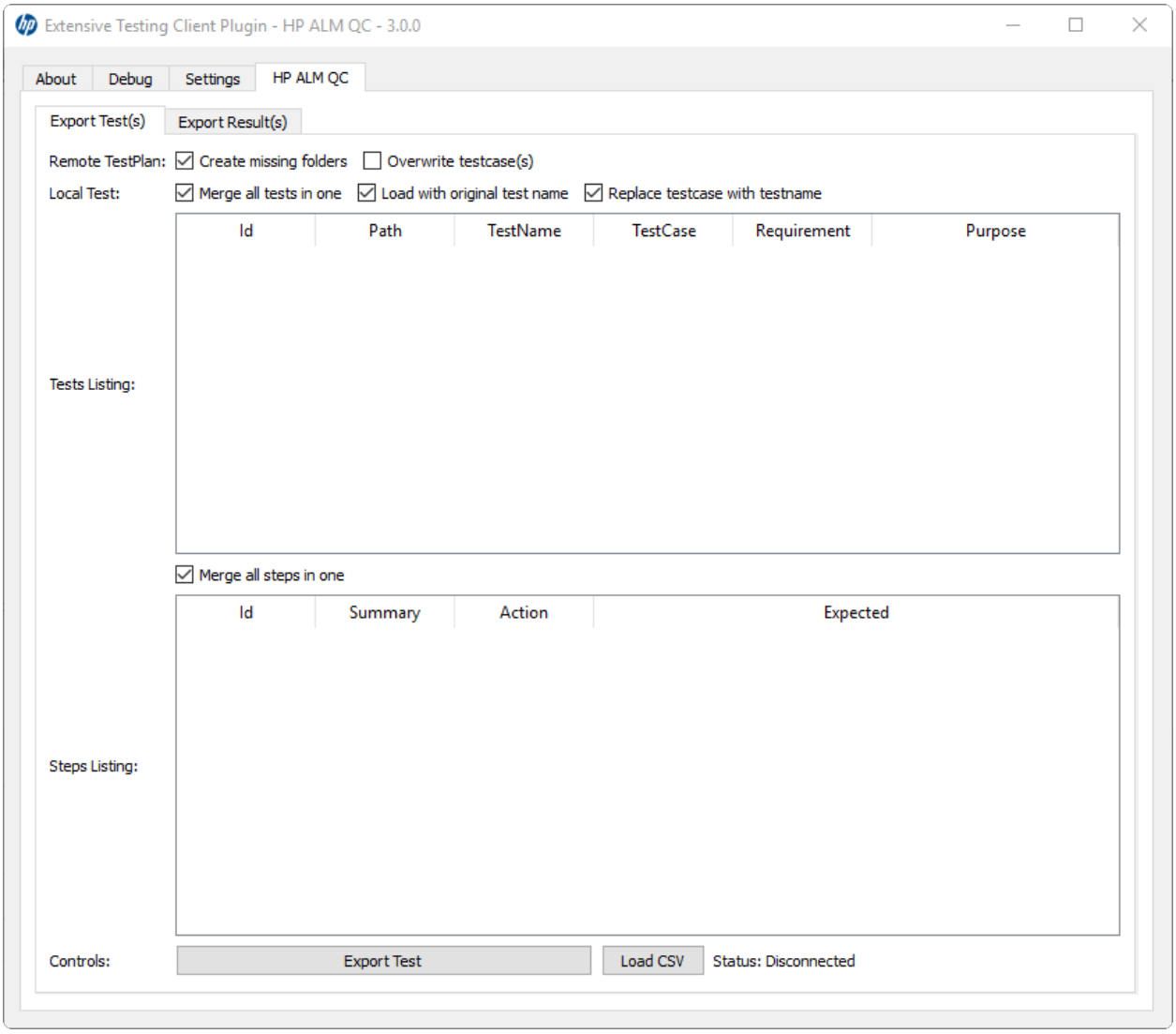
4.5.1 Plugin HP ALM

Le plugin `HP ALM` permet d'exporter les tests et résultats depuis le client Extensive vers HP ALM QualityCenter. Cette approche permet d'être autonome vis à vis de QC.

La configuration du plugin se fait dans la page `Settings`, il faut configurer à minima :

- nom d'utilisateur
- le mot de passe
- le domaine
- le projet

Pour exporter un test, il faut générer le design d'un test depuis le client et cliquer sur le plugin HP ALM disponible dans la barre d'outils.



L'export des résultats peut se faire depuis la fenêtre exploration des archives, Le plugin doit être disponible dans la barre d'outil lors qu'un rapport de test est chargé.

Note : Le plugin est compatible avec un HP ALM QC >= 12, l'api REST est utilisée.

4.5.2 Plugin Jenkins

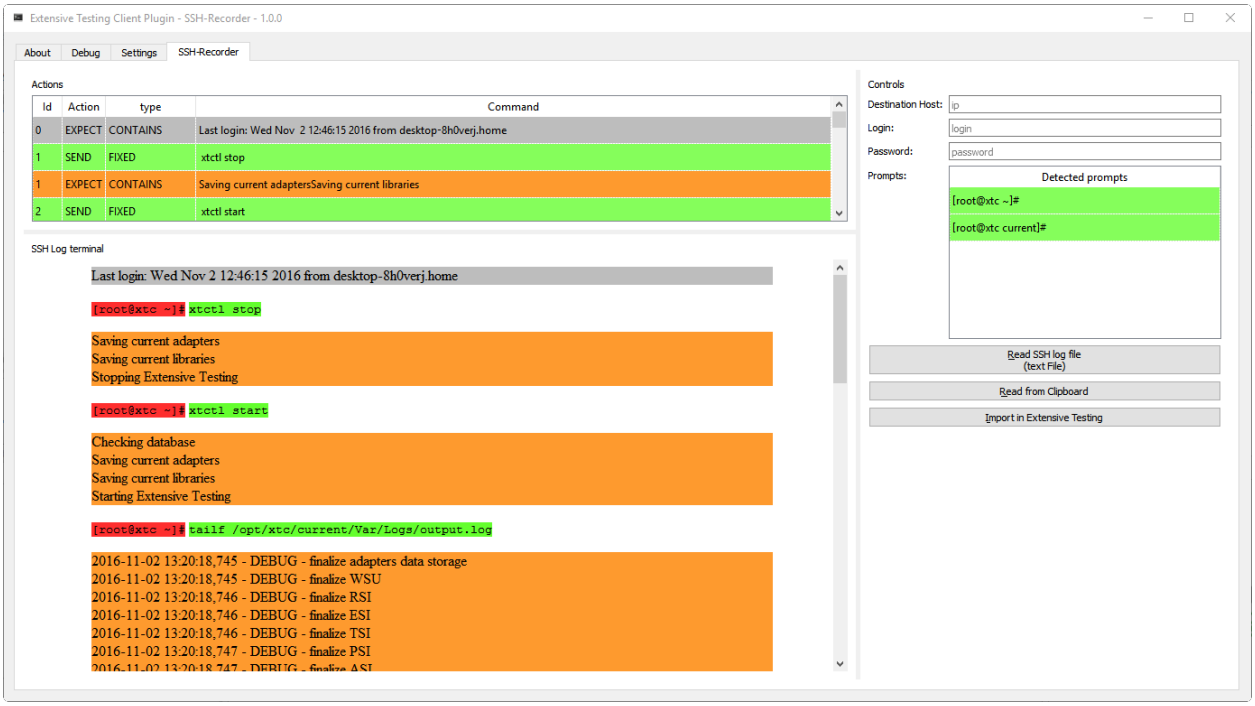
Le plugin `Jenkins` ne fait pas grand chose dans cette version... Il fournit juste un lien vers l'interface web de son Jenkins préféré.

4.5.3 Plugin Shell Recorder

Le plugin `Shell Recorder` permet d'importer une séquence de commandes shell dans l'assistant de conception et de générer le test associé. Il permet donc de rejouer facilement une séquence de commandes.

La 1ière étape consiste à importer une session ssh (depuis un terminal putty par exemple) depuis le presse papier ou en important directement un fichier texte contenant la séquence des commandes shell.

Le plugin détecte automatiquement le prompt dans la séquence pour parser les commandes et résultats associés. Si le prompt n'est pas détecté, il est possible de le modifier manuellement.



4.5.4 Plugin SeleniumIDE

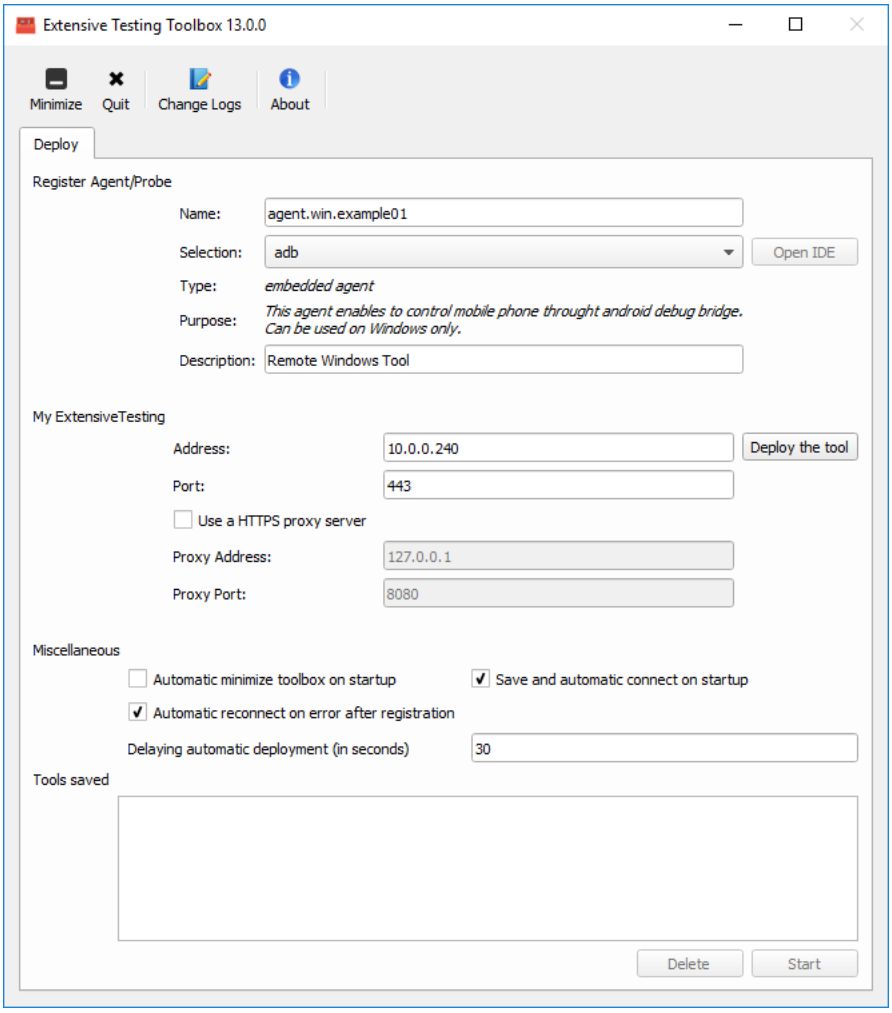
L'utilisation du plugin `SeleniumIDE` implique une utilisation basique. Il permet de convertir un fichier enregistré avec le plugin SeleniumIDE de firefox dans l'assistant de conception.

Astuce : Il est plus efficace d'utiliser l'assistant en direct pour être en phase avec la philosophie de la solution.

CHAPITRE 5

Boite à outils

La boite à outils permet de démarrer des agents sur des postes dédiés. Les agents sont indispensables pour exécuter des tests avec Selenium sur des postes dédiés ou bien pour déporter l'exécution d'un test.



5.1 Déploiement

Cette fenêtre permet de choisir l'agent à démarrer. Le type d'agent à démarrer peut être choisi dans la liste déroulante. Enfin un agent nécessite d'être enregistré auprès du serveur de test pour pouvoir l'utiliser.

Un agent va permettre de faire une exécution distribuée de vos tests. Par exemple, un agent déployé sur plusieurs machines va permettre d'exécuter le même test sur différent environnement à tester ou piloter.

La liste complète des agents disponibles sont décrits dans le chapitre *Compléments Serveur > Agents*.

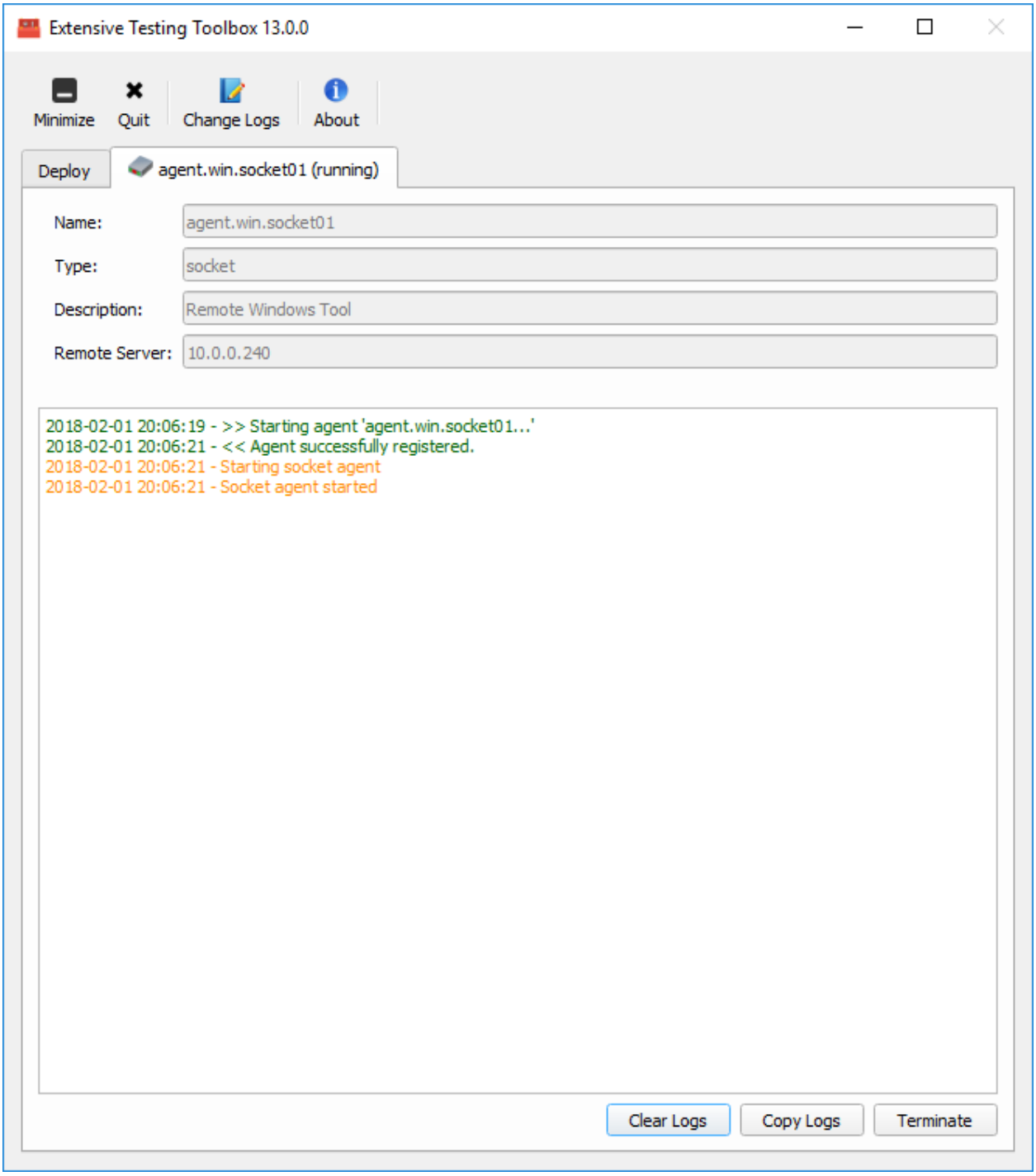
Note : Le nom de l'agent doit être unique pour réussir l'enregistrement.

Astuce : Pour une meilleure visibilité des agents disponibles, il est conseillé de respecter le formalisme suivant pour les noms :

[agent].[environnement].[prénom_testeur].[nom][numéro_instance]

Exemple : agent.win.denis.socket01

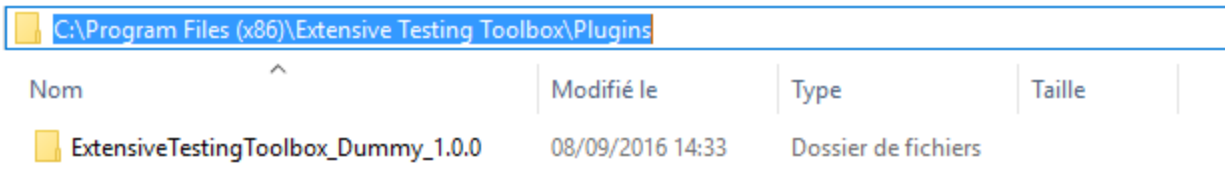
Exemple d'un agent déployé et en cours d'exécution :



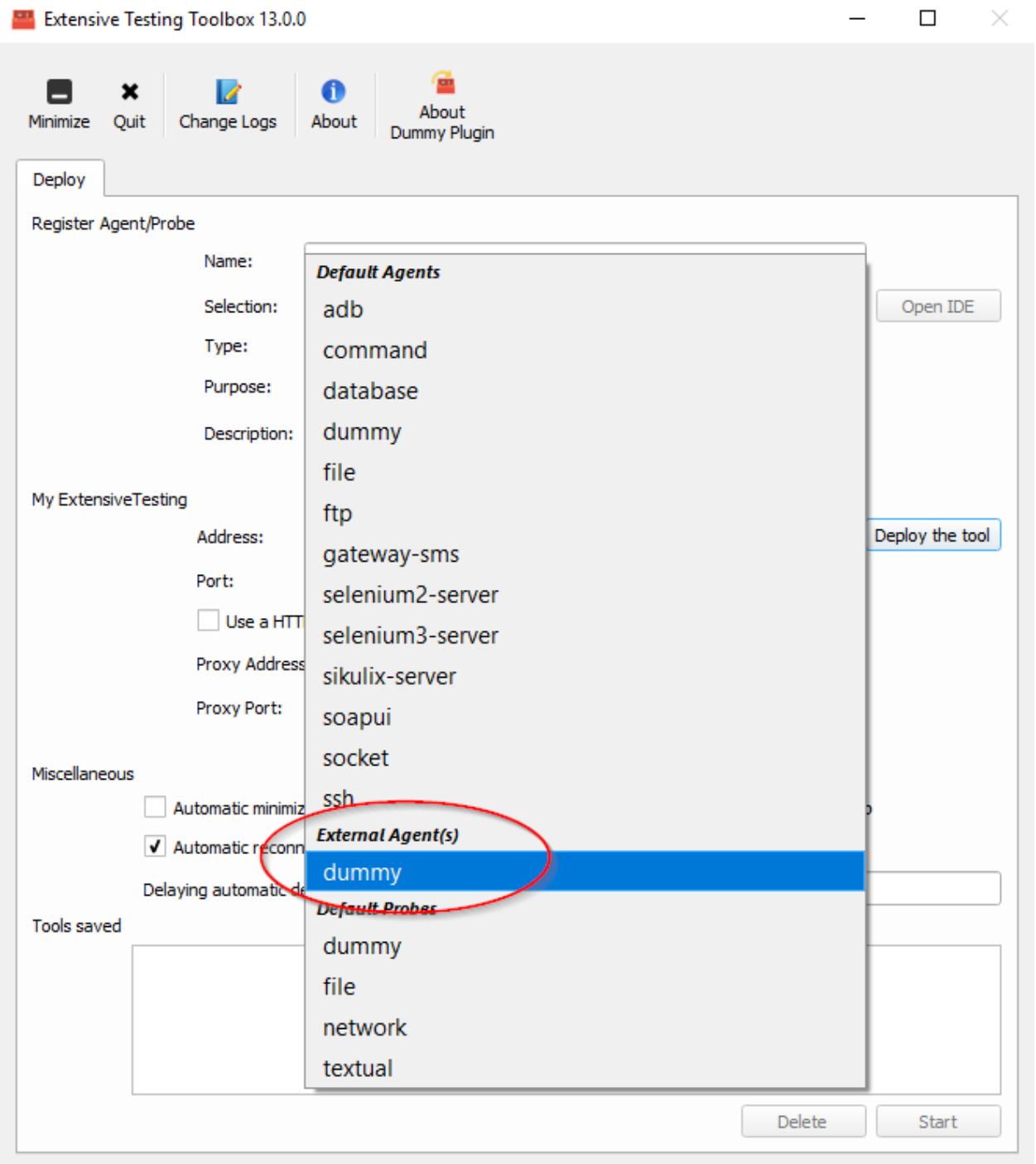
5.2 Compléments

La boite à outils peut être enrichie avec de nouveaux plugins.

Pour ce faire il faut suivre la procédure décrite dans le chapitre *Contributions* > *Développement plugins* > *Boites à outils*. Les plugins sont à déposer dans le répertoire `Plugins`.



Après redémarrage de la boite à outils, le complément apparait dans la liste des agents `external`



CHAPITRE 6

Interface Web

6.1 Partie tests

6.1.1 Variables globales

Les variables globales permettent de décrire un jeu de données pour l'ensemble d'un projet. Ils sont automatiquement accessibles au niveau de chaque test depuis les propriétés.

Le format `JSON` doit être utilisé.

6.2 Partie administration

6.2.1 Utilisateurs

La solution nécessite de créer des comptes utilisateurs. La création peut se faire à travers l'interface Web ou bien directement depuis l'API.

La création d'un utilisateur nécessite à minima de préciser :

- un nom d'utilisateur
- un mot de passe
- son niveau d'accès (administrateur, testeur)
- les projets autorisés

Note : Si une adresse email est précisée, alors il est possible de recevoir les résultats des tests automatiquement dans cette boite mail.

6.2.2 Projets

Les tests peuvent être organisés par projet. L'ajout ou la suppression de projet peut se faire depuis l'interface web ou directement depuis l'api REST.

Note : Le projet `Common` existe par défaut et est accessible par l'ensemble des utilisateurs, il ne peut pas être supprimé.

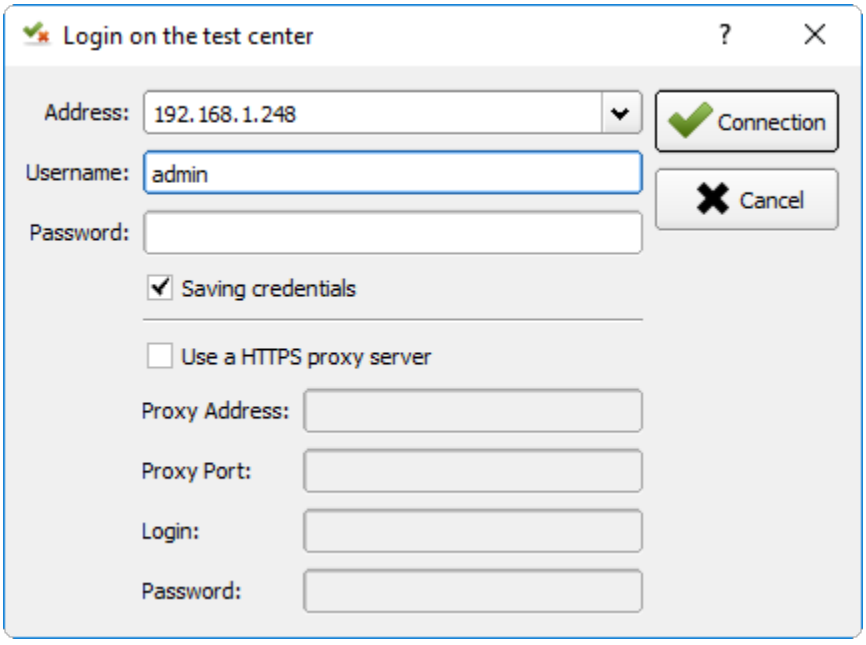
CHAPITRE 7

Première utilisation

7.1 Connexion du client au serveur

Après avoir ouvert le client, la première étape consiste à se connecter au serveur de test. Pour ce faire il faut avoir à disposition son compte utilisateur ainsi que l'adresse du serveur.

La fenêtre de connexion est disponible depuis le menu `Get Started > Connect` ou bien directement sur la page d'accueil. Une fois la connexion réussie, l'utilisateur peut accéder à l'ensemble des tests automatiques existant sur le serveur.



Note : L'utilisateur `admin` peut être utilisé dans le cadre de la découverte de la solution.

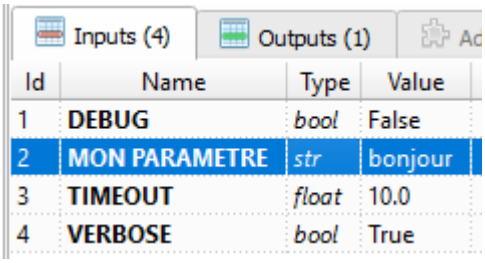
7.2 Écriture d'un test (script)

La première utilisation consiste à créer un très simple premier cas de test en affichant la valeur d'un paramètre du test.

1. Créer un test de type `Unit`



2. Ajouter le paramètre MON_PARAMETRE de type `str` avec la valeur « bonjour »



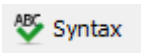
3. Modifier le test au niveau de la section `definition` pour afficher la valeur du paramètre.

```
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)

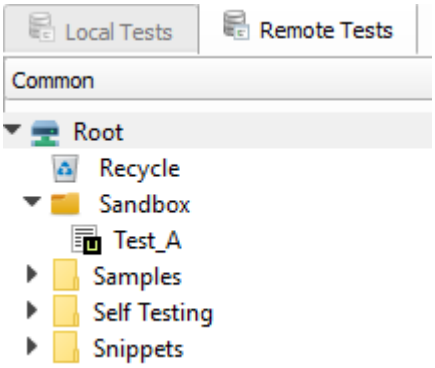
  self.step1.setPassed(actual="success")
```

Note : Il est possible de vérifier la syntaxe du test avant exécution en cliquant sur le bouton `Syntax`.



---

4. Enregistrer le test dans le dépôt avec le nom « Test_A » dans le répertoire `Sandbox`



7.3 Ecriture d'un scénario (conditionnel)

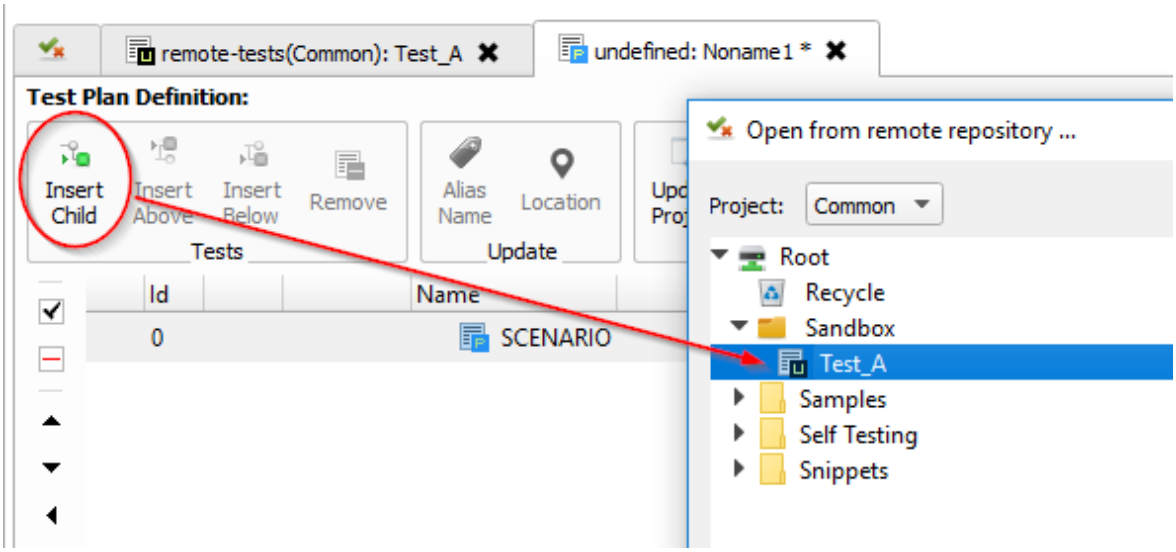
Note : Ce mini guide part du principe que vous avez suivi le chapitre `Ecriture d'un test script`.

L'exemple suivant explique comment créer son premier scénario avec une surcharge des variables de tests.

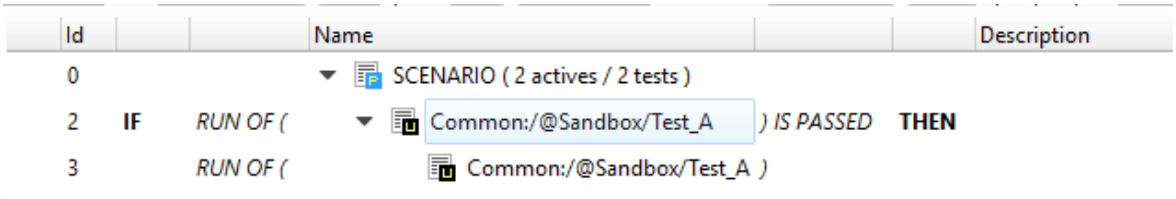
1. Créer un test de type `Plan`.



2. Insérer le test « Test_A » dans le scénario. Cliquer sur le bouton `Insert Child` et sélectionner le test *Test_A*.

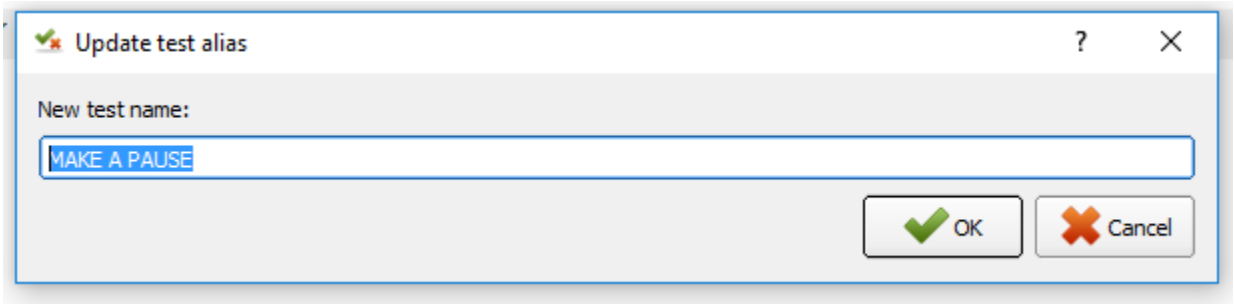


3. Après insertion, cliquer sur le test *Test_A* et insérer de nouveau le même test.



4. Enregistrer le scénario dans le dépôt de test avec le nom « Scenario_A » dans le répertoire `Sandbox`.
5. Ajouter le paramètre MON_PARAMETRE avec la valeur « au revoir » au niveau du scénario.

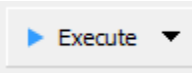
Astuce : Ne pas hésiter à définir un alias pour le nom du test pour rendre le scénario plus lisible.



7.4 Exécution d'un test

Note : Ce mini guide part du principe que vous avez suivi les chapitres *Ecriture d'un test script* et *Ecriture d'un scénario*.

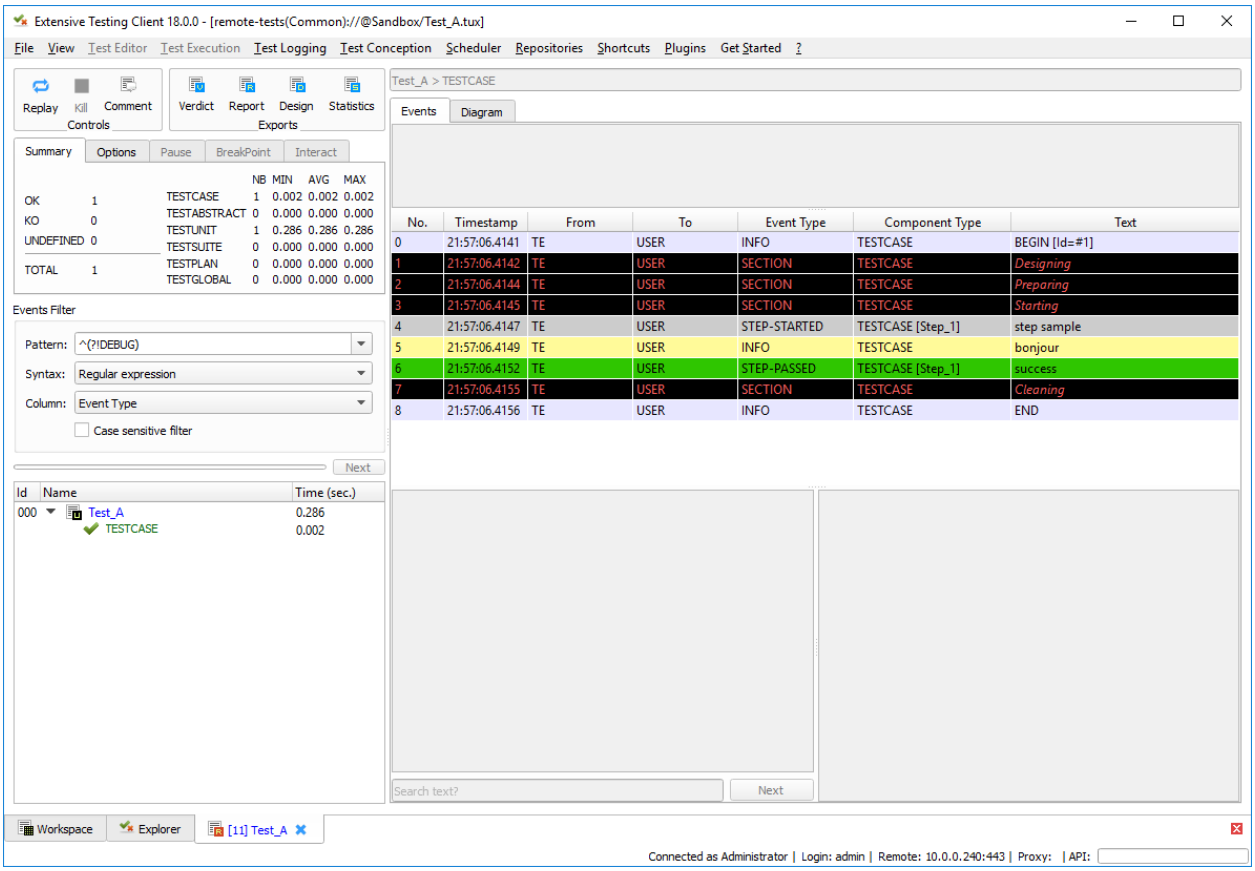
L'exécution d'un test peut se faire en cliquant sur le bouton `Execute`. Ouvrir les tests *Test_A* et *Scenario_A* et les exécuter.



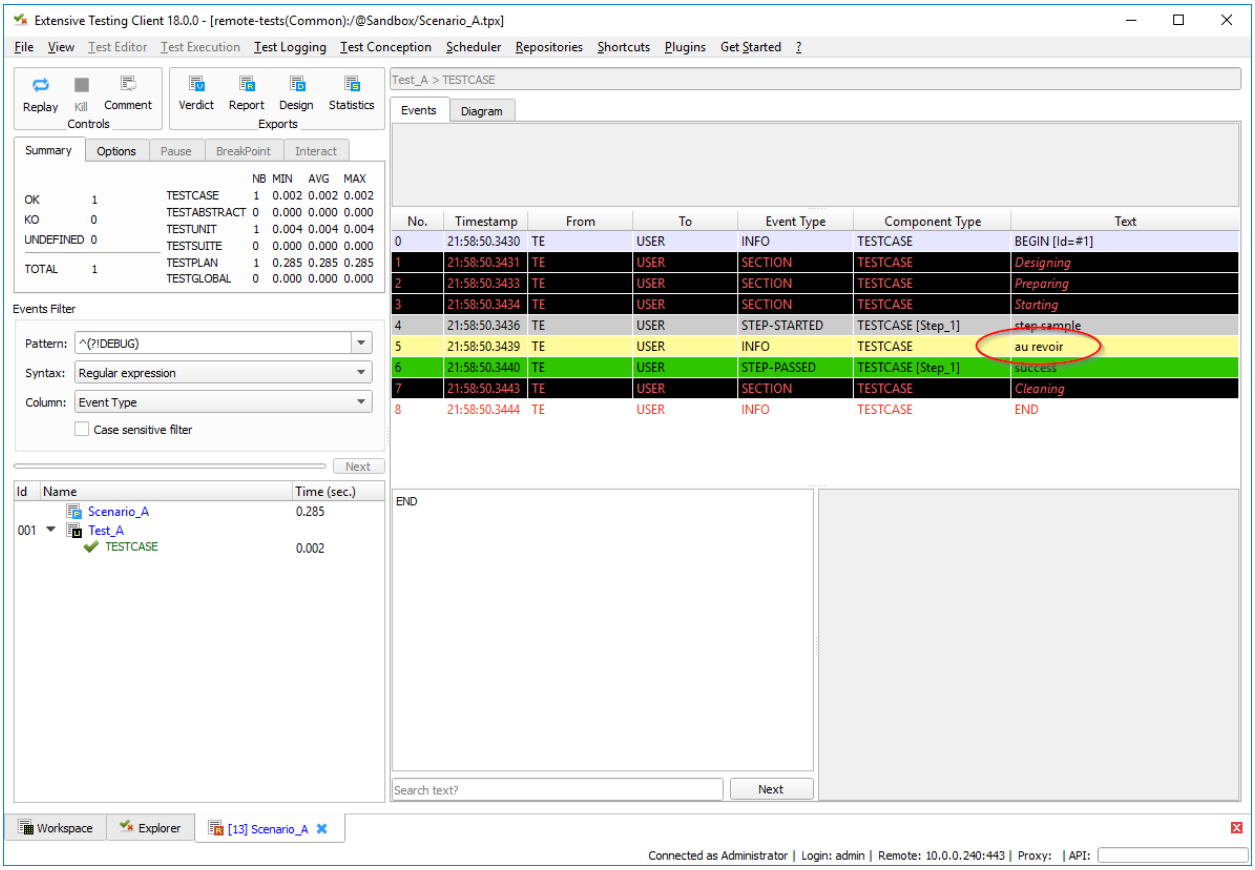
7.5 Analyse des résultats

Note : Ce mini guide part du principe que vous avez suivi les chapitres *Ecriture d'un test script* et *Ecriture d'un scénario*.

La 1ière fenêtre d'analyse montre l'exécution du test « Test_A » et notamment le message « bonjour ».



La 2ième fenêtre d'analyse montre l'exécution du test « Scenario_A » et notamment le message « au revoir ».



Ce premier usage montre comment exécuter un test et un scénario ansi que la surcharge des variables de tests.

7.6 Les bonnes pratiques

Astuce : Pour garder une bonne lisibilité dans les tests de type scripts, il ne faut pas utiliser de *try/except*. Le framework intercepte toutes les exceptions à son niveau.

Astuce :

Il faut absolument prendre le temps de déclarer les étapes de test car elles permettent

- de comprendre rapidement le test sans le script.
- d'avoir des rapports de test pertinents et compréhensibles.

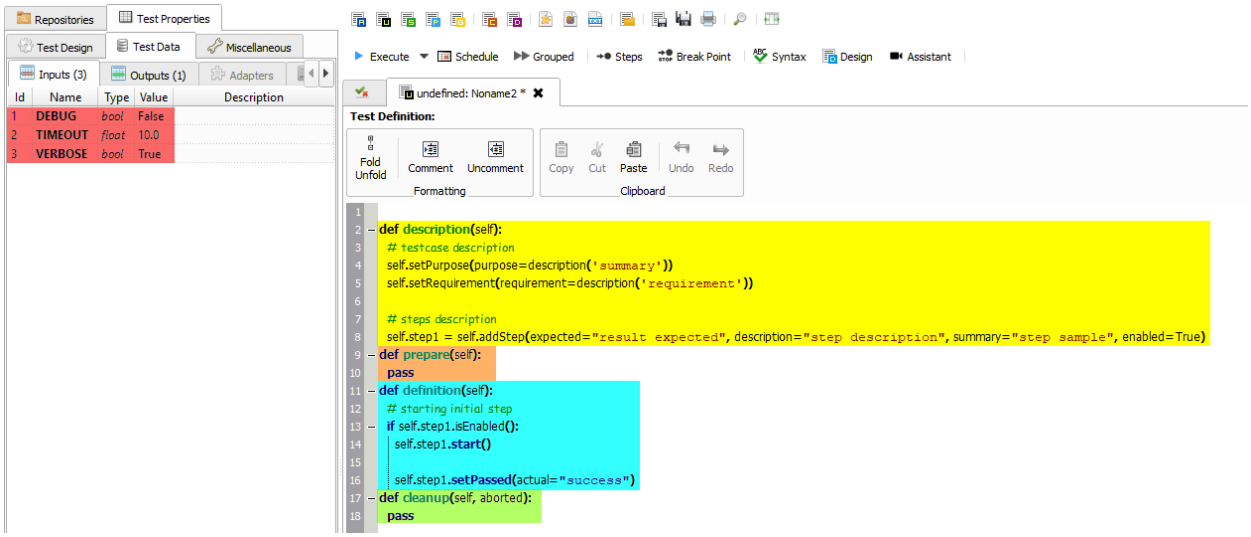
Astuce : Pour faciliter la maintenance de vos tests et les rendre réutilisables, il ne faut pas avoir de valeur en dur dans votre test. Il faut systématiquement les mettre en paramètres de test, c'est fait pour.

CHAPITRE 8

Exemples de tests

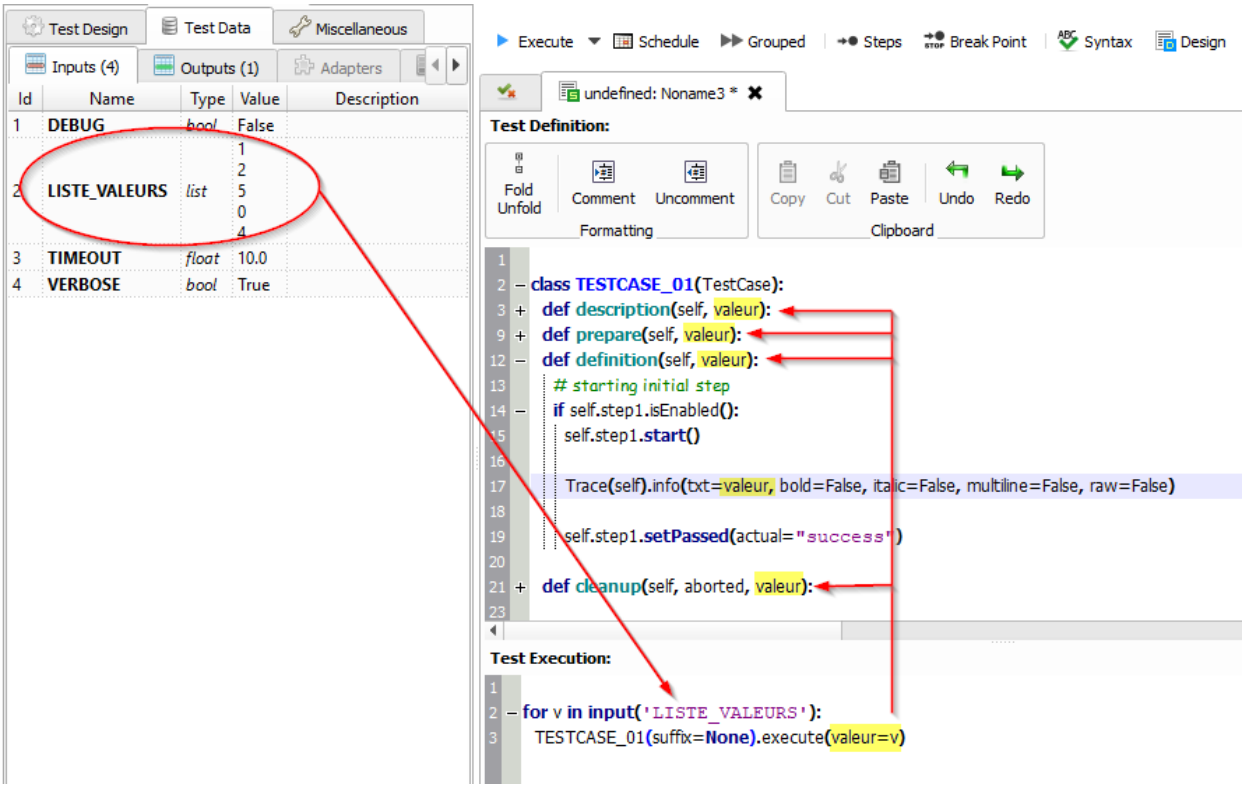
8.1 Cas de test (unit)

Cet exemple montre comment utiliser un cas de test simple. Un cas de test se compose de 4 sections exécutées automatiquement par le framework de test ainsi que les paramètres de tests associés.



8.2 Cas de test (suite)

Une suite de tests permet d'éxécuter à la suite plusieurs cas de test. L'exemple montre comment boucler sur un cas de test tout en modifiant les données entrantes.



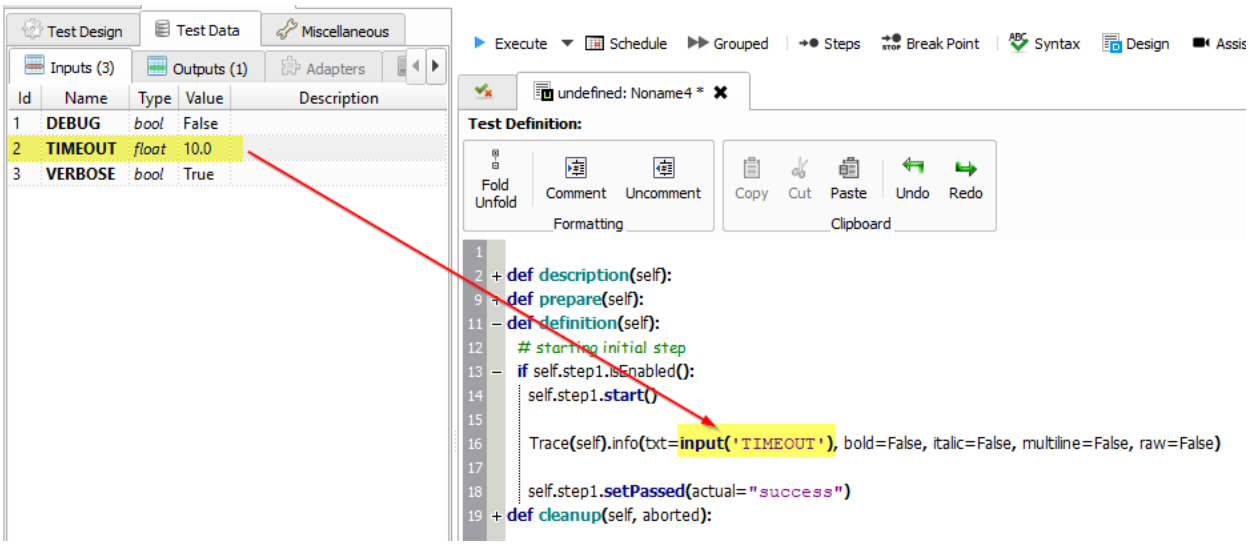
Il est donc possible d'ajouter autant d'arguments que nécessaire au niveau de la fonction `execute()` et de les ajouter à l'identique au niveau des 4 sections.

Note : Il est possible d'ajouter un préfixe au niveau du cas du test en utilisant l'argument `prefix`.

8.3 Variables de test

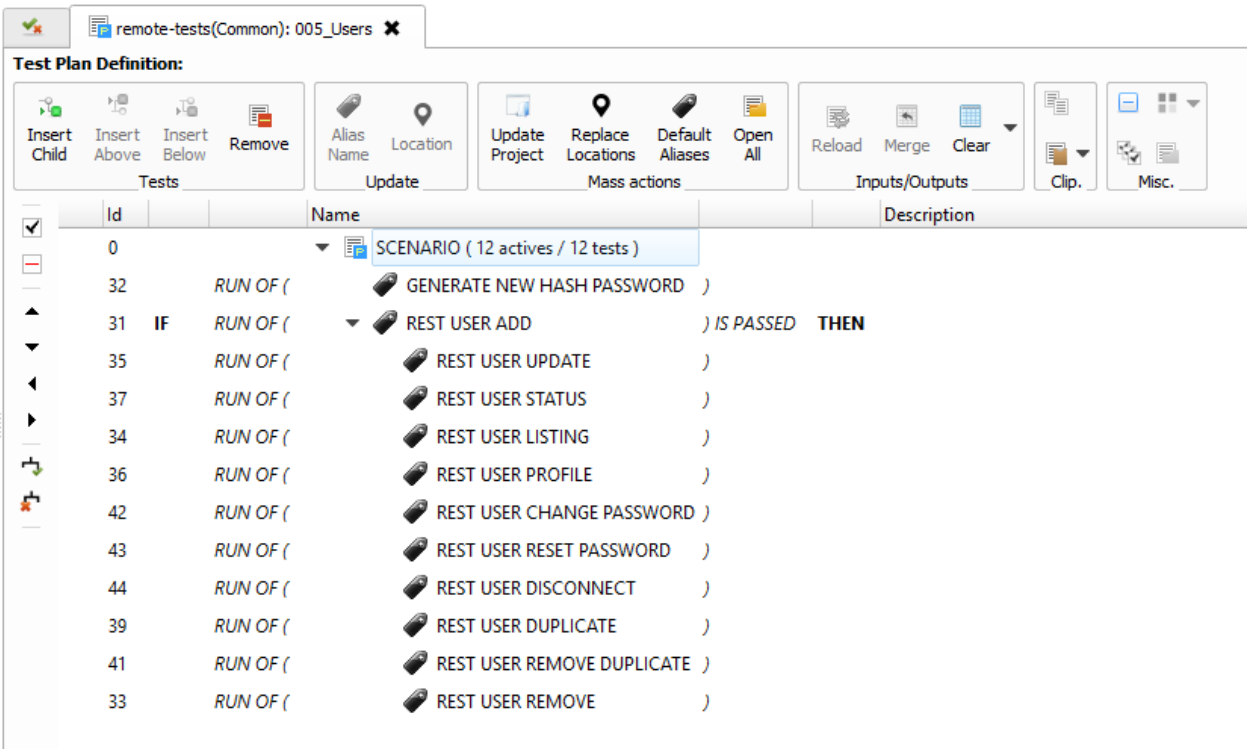
Les variables sont utilisables depuis un test. Il en existe plusieurs types. L'exemple ci-dessous montre comment récupèrer un paramètre depuis son test.

Un paramètre de test peut être récupéré au niveau du test en utilisant la fonction *input*. Le nom du paramètre à récupérer est à préciser.



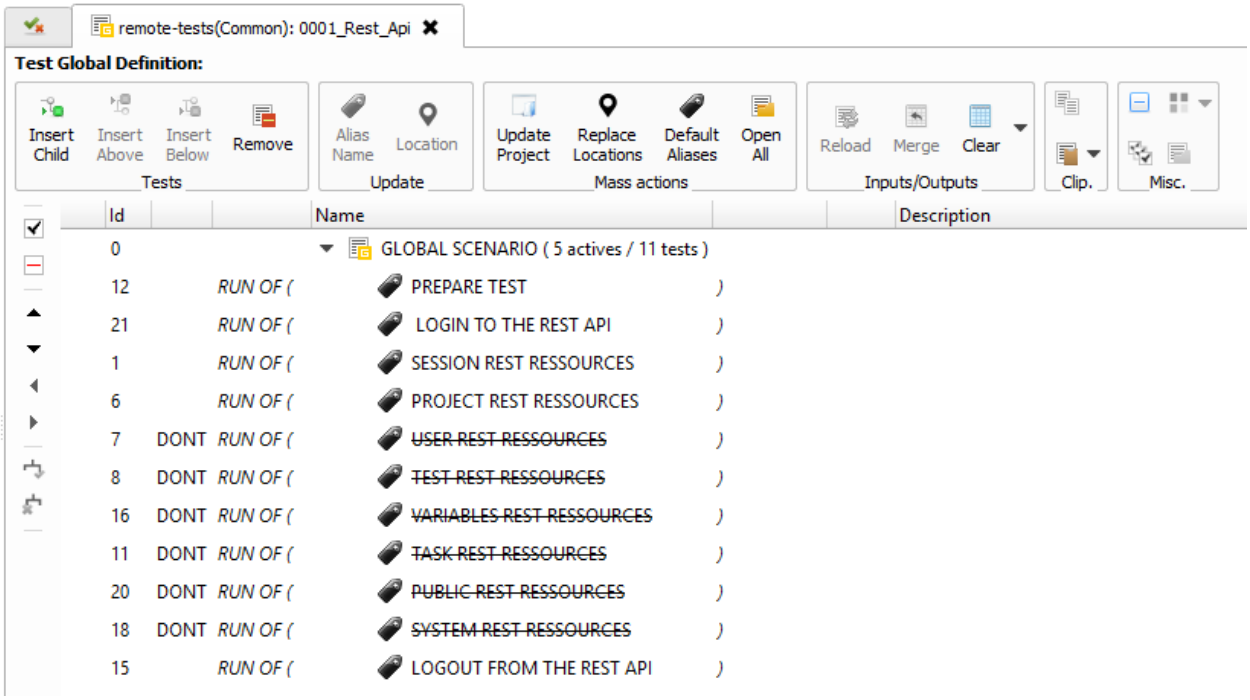
8.4 Scénario

Un scénario permet d'exécuter plusieurs cas de tests à la suite avec des conditions de résultats entre eux. Il est ainsi possible de lancer ou non un test selon que le test précédent soit OK ou non. Il est possible de surcharger les paramètres de tests au niveau du scénario.



8.5 Campagne de tests

Une campagne permet d'exécuter plusieurs scénarios. Il est possible de surcharger les paramètres de tests au niveau des paramètres de la campagne.



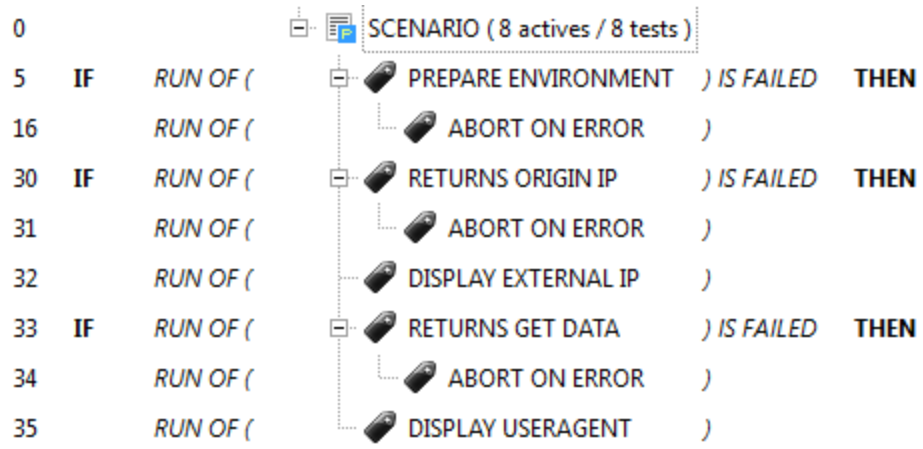
8.6 Rest API

Pour écrire un test d'api REST, il est conseillé :

— d'utiliser le test réutilisable `/Snippets/Protocols/04_Send_JSON`
— de décrire le serveur cible en JSON (ip/port destination, support du http)

Exemple :

Le test appelle le service `httpbin.org` en https et appelle le service `ip` qui permet d'obtenir l'ip réelle du client en json.



Le scénario se décompose en plusieurs étapes :

1. Préparation de l'environnement : description de l'environnement testé (adresse, port réseaux, etc...) L'environnement est configuré dans le paramètre *ENVIRONMENT* du test *PREPARE ENVIRONMENT* (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. Si la préparation de l'environnement ne fonctionne pas, alors le scénario est arrêté en appelant le test réutilisable `Snippets/Do/02_Terminate` (Id=16)

3. On envoie une requête REST et on décrit la réponse attendue en utilisant le test réutilisable `/Snippets/Protocols/04_Send_JSON` (Id=30). Si cette étape ne fonctionne pas alors on annule le test (Id=31)

La réponse reçue est vérifiée par le framework et ce qui a été décrit par le testeur dans le paramètre `HTTP_RSP_BODY`

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

La configuration indique qu'il faut vérifier dans la réponse que la clé *origin* est présente et d'enregistrer la valeur dans le cache avec la clé `EXTERNAL_IP`

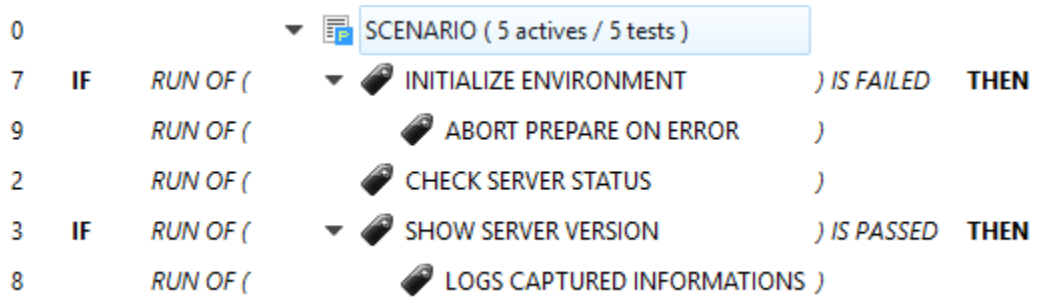
4. On affiche la valeur reçue dans la réponse avec le test réutilisable `Snippets/Cache/02_Log_Cache` (Id=32)

Note : L'exemple présenté ci-dessous est disponible en totalité dans les échantillons de test : `/Samples/Web_API/001_httpbin_rest.tpx`.

8.7 Contrôles SSH

Pour écrire un test SSH, il est conseillé :

— d'utiliser le test réutilisable `/Snippets/Protocols/01_Send_SSH`
— de décrire le serveur cible en JSON (ip, compte, mot de passe à minima)



Le test se décompose en plusieurs étapes :

1. Chargement de la description (ip, compte, mot de passe) de la machine cible dans le cache
2. Appel au test générique `/Snippets/Protocols/01_Send_SSH` pour récupérer la version du serveur La version (si trouvée à l'écran) est sauvegardée dans le cache avec la clé *SERVER_VERSION* Si la version n'est pas trouvée, le test part en erreur.

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

3. Affichage de la version depuis le cache.

Note : L'exemple complet est disponible dans les échantillons de tests `/Self Testing/SYSTEM/000_System.tpx`.

8.8 Navigateurs Internet

Pour écrire le test d'une application web, il faut :

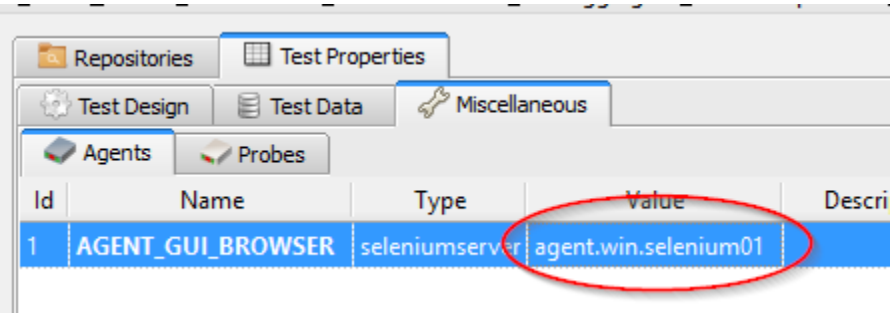
— obligatoirement déployer un agent `selenium` sur un poste disposant d'un navigateur firefox, chrome, internet explorer ou edge

— avoir accès au code source de la page web depuis son navigateur
— avoir des connaissances en xpath
— connaître les bases du code HTML

L'approche préconisée pour écrire les tests web est la suivante :

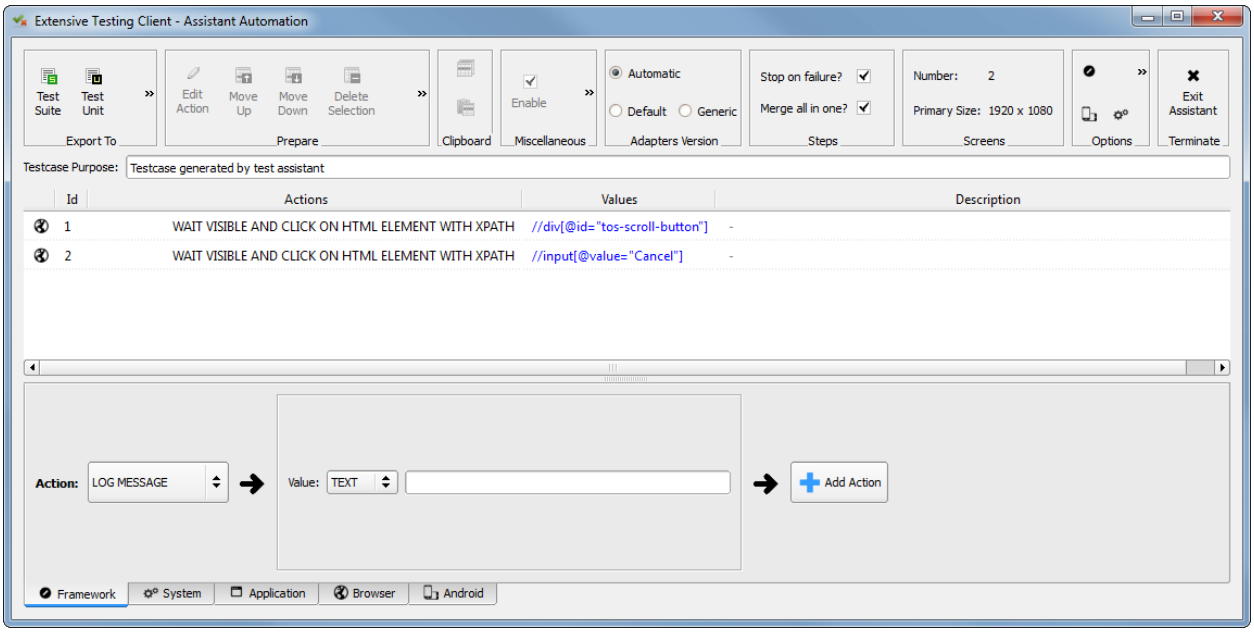
— identifier le nombre de pages affichées à scripter (et la réutilisation possible de ces pages)
— identifier les différents enchaînements de pages pour créer les scénarios
— identifier les parcours utilisateur

Pour exécuter ce type de tests, il faut absolument déclarer l'agent qui sera utilisé

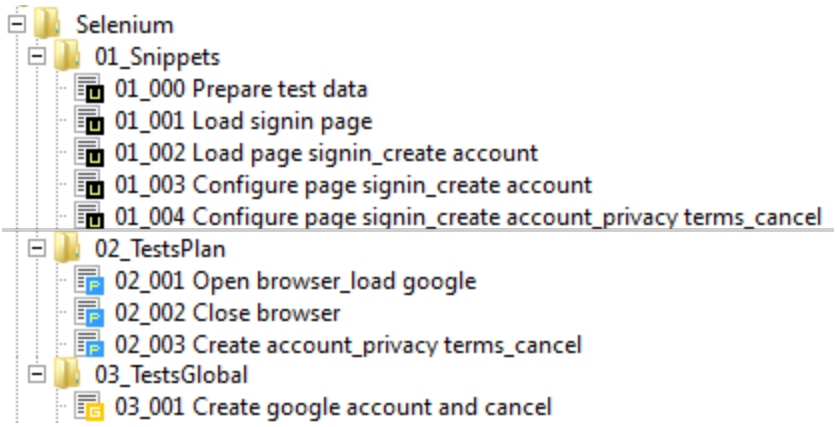


L'écriture des tests se réalise à travers l'assistant. Il permet de décrire les différentes étapes et de générer le test unit équivalent. Les enchaînements de pages sont à décrire dans les tests plans. Le parcours utilisateur est à définir dans un test global.

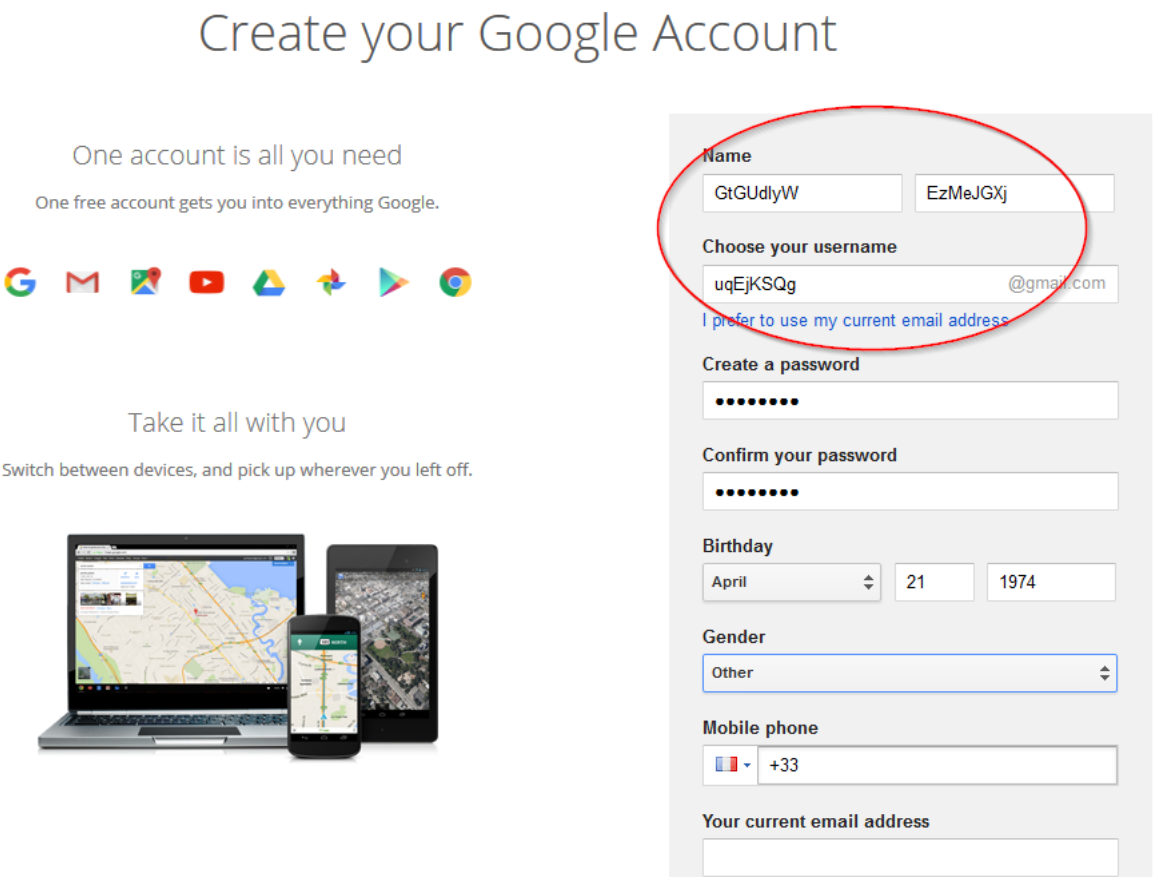
La solution préconise aussi de n'utiliser que le xpath pour identifier des élements HTML.



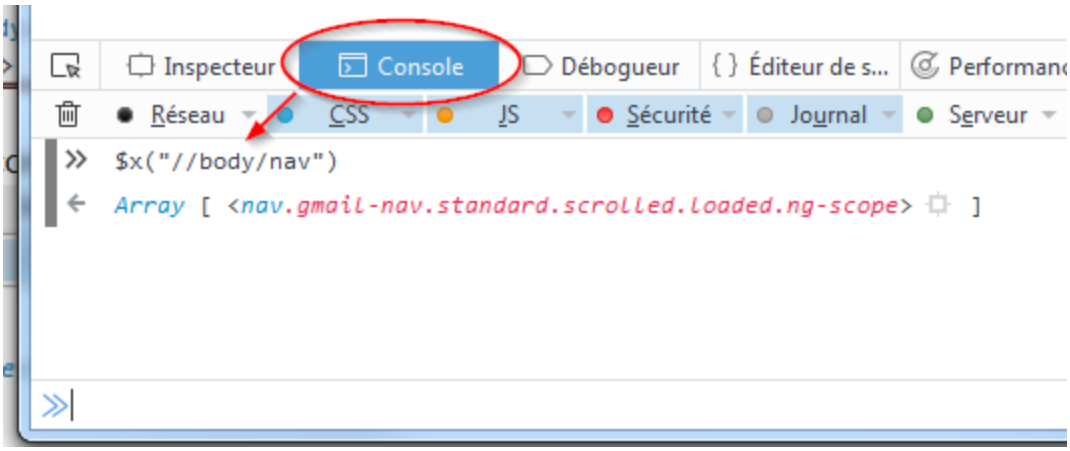
L'exemple ci-dessous montre comment créer un compte Google en utilisant un nom et prénom aléatoires.



Exemple de résultat :



Astuce : Il est possible d'utiliser les outils de développement des navigateurs pour valider les xpaths.



Note : L'exemple présenté ci-dessous est disponible en totalité dans les échantillons de test `/Samples/Tests_Gui/Selenium/`.

Note : Selenium3 nécessite au minimum Java 8 sur le poste client.

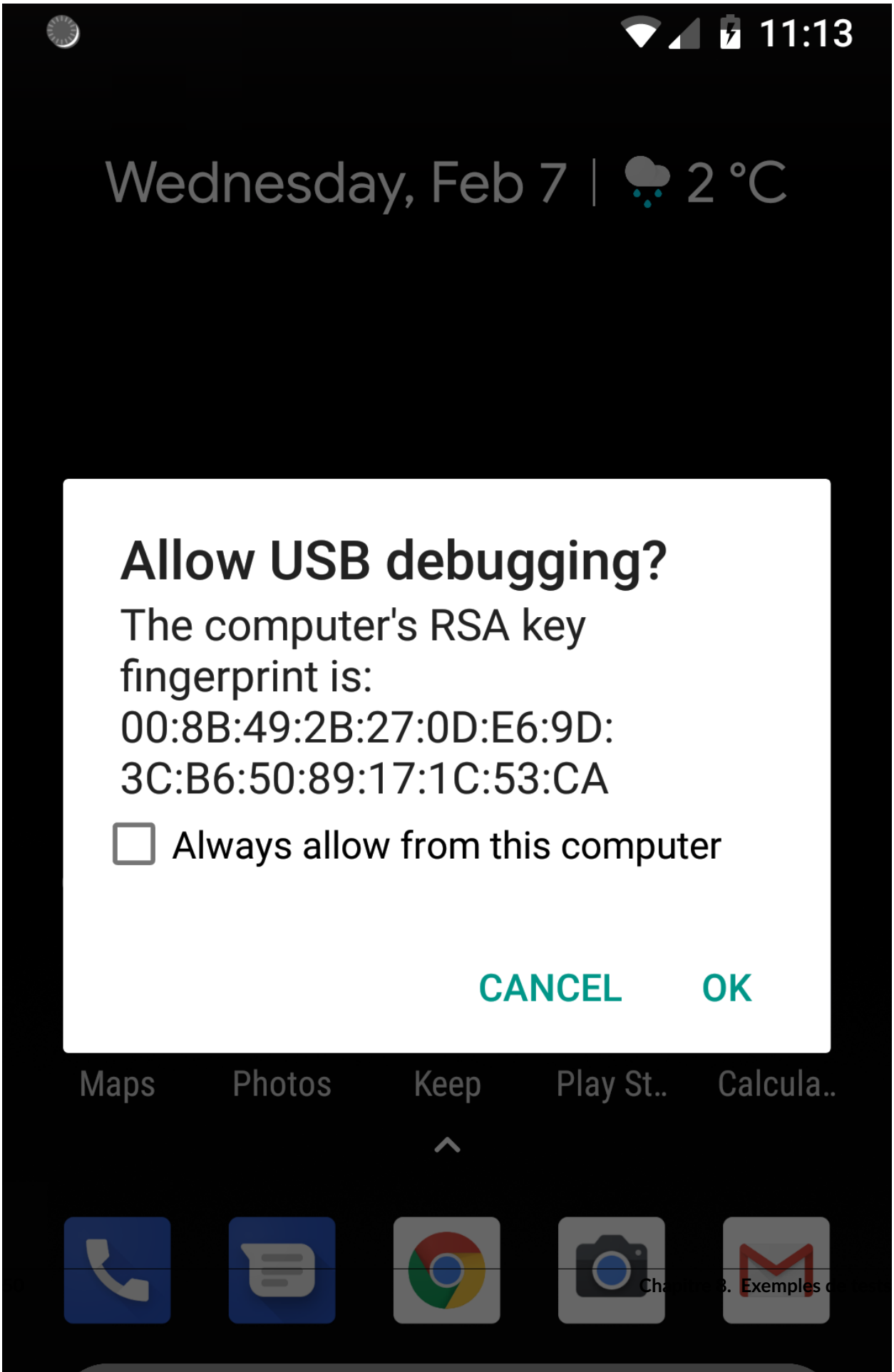
Navigateurs	Version Selenium	Gecko
Firefox <47	Selenium 2	Non
Firefox > 47	Selenium 3	Oui
IE	Selenium 3	N/A
Chrome	Selenium 3	N/A

8.9 Mobile Android

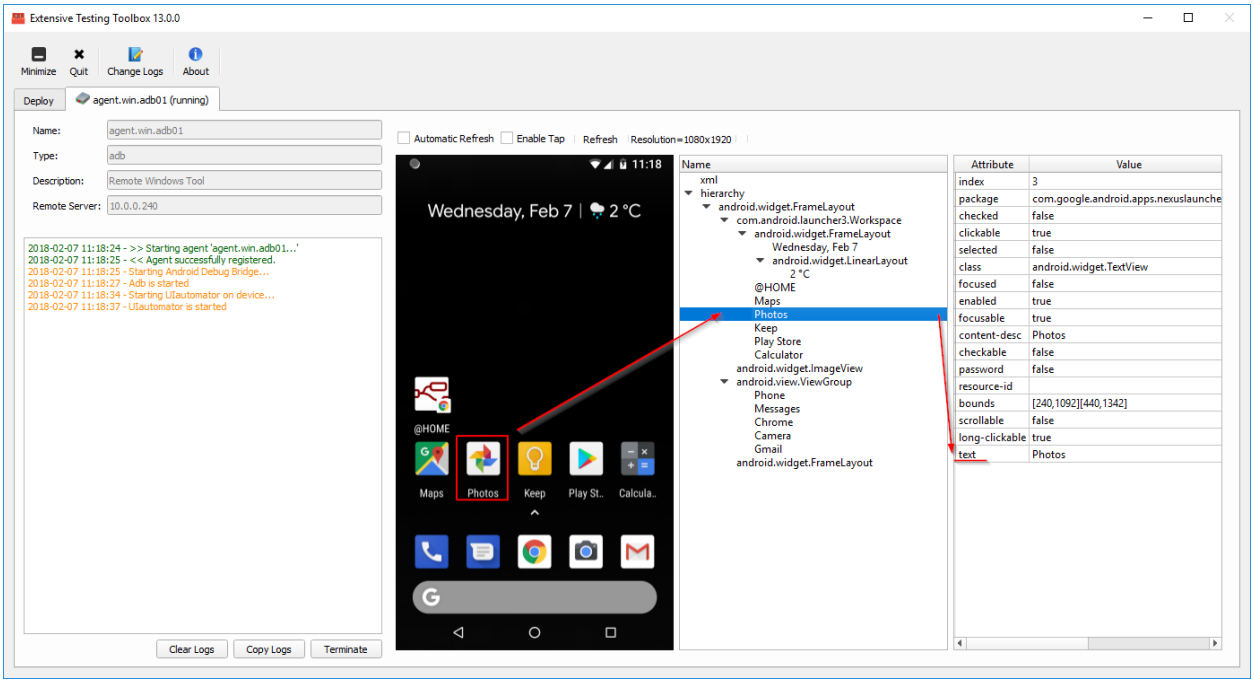
Pour écrire le test d'une application mobile, il faut :

- Avoir un téléphone mobile Android connecté en USB sur un PC
- Déployer un agent adb sur un poste avec un mobile android connecté dessus.
- Avoir accès à la description xml des applications depuis l'agent

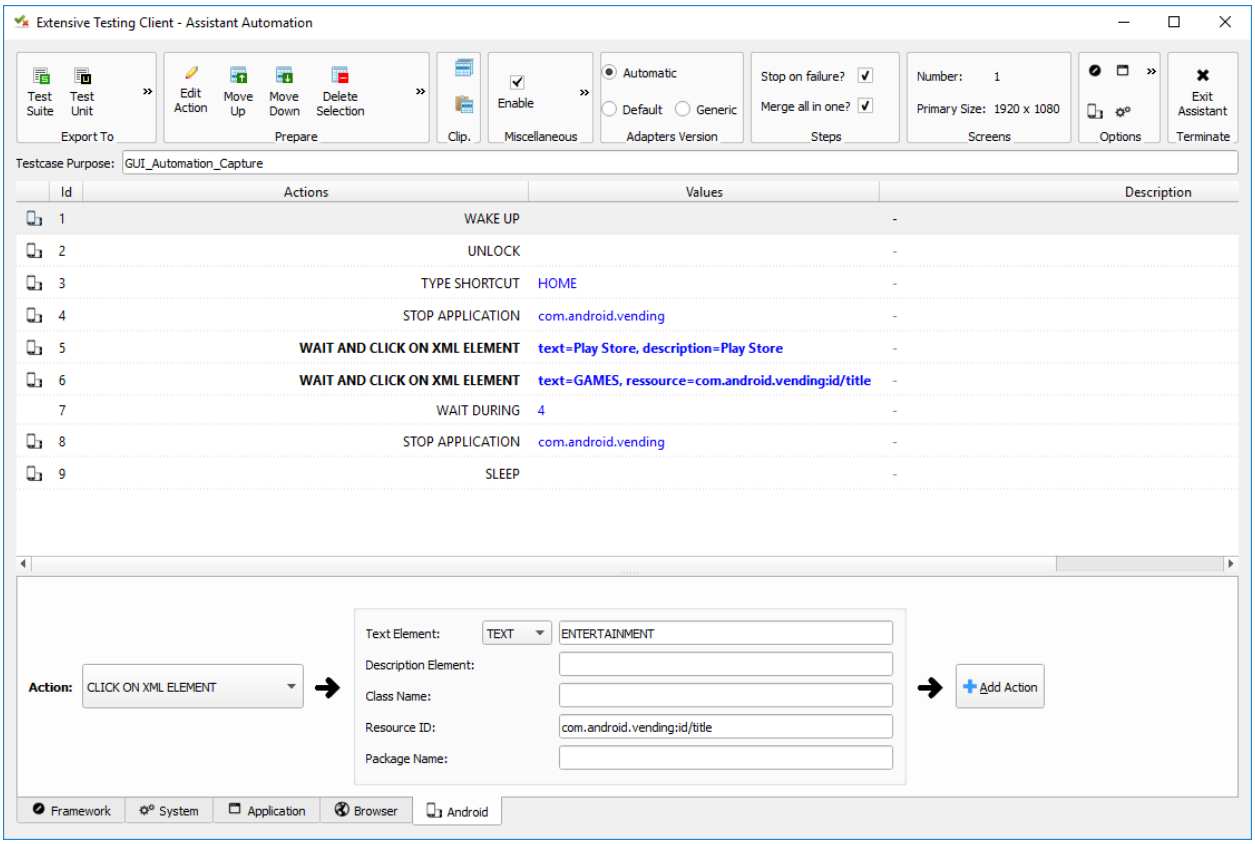
La connexion de l'agent adb sur le mobile android nécessite d'accepter la clé RSA.


11:13
Wednesday, Feb 7 | 2 °C
Allow USB debugging?
The computer's RSA key fingerprint is:
00:8B:49:2B:27:0D:E6:9D:
3C:B6:50:89:17:1C:53:CA
Always allow from this computer
CANCEL
OK
Maps
Photos
Keep
Play St..
Calcula..

Après connexion, l'agent affiche un aperçu de l'écran sur le pc. Il est possible de parcourir l'interface depuis l'agent et d'avoir les élements XML disponibles dans la page.



L'écriture des tests se réalise avec l'assistant, il permet de décrire les différentes étapes et de générer le test unit équivalent. Il est indispensable de se baser sur l'agent adb pour avoir la liste des élements et attributs XML disponibles.



Note : L'exemple complet est disponible dans les échantillons de tests `/Samples/Tests_Mobiles/03_PlayStore.tux`.

Important : L'activation du mode debogage `USB` est obligatoire sur le téléphone.

CHAPITRE 9

Tests réutilisables

L'intérêt des tests réutilisables
- factoriser la base de tests
- réutiliser les tests
- limiter l'écriture de scripts pour concevoir les scénarios

Ces types de tests sont à utiliser en mode `test plan`.

9.1 Données partagées

9.1.1 Mise en cache d'une valeur

Important : Chemin d'accès du test réutilisable `/Snippets/Cache/01_Set_Cache.tux`

Ce test réutilisable consiste à sauvegarder une valeur dans le cache de données disponible durant l'exécution d'un test.

Paramètre(s) à configurer :

Paramètres	Description
DATAS	Contient la liste des valeurs à sauvegarder

Le paramètre `DATAS` contient la liste des valeurs à sauvegarder avec le format :

```
# mon commentaire
[!TO:CACHE:<MA_CLE>:];ma valeur
```

Exemple

```
# Save misc data
[!TO:CACHE:EXAMPLE:];hello world

# Save server information in the cache
[!TO:CACHE:SERVER_DESCRIPTION:];[!FROM:INPUT:TEST_PURPOSE:]
```

Note : Il est possible de sauvegarder plusieurs valeurs avec ce test.

9.1.2 Affichage d'une valeur

Important : Chemin d'accès du test réutilisable `/Snippets/Cache/02_Log_Cache.tux`

Ce test réutilisable permet d'afficher la valeur d'une clé présente dans le cache durant l'exécution du test.

Paramètre(s) à configurer :

Paramètres	Description
MESSAGES	Contient la liste des paramètres à logguer dans le test

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

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

Note : Il est possible d'afficher plusieurs valeurs en une seule fois

9.1.3 Reset du cache

Important : Chemin d'accès du test réutilisable `/Snippets/Cache/02_Reset_Cache.tux`

Ce test réutilisable permet de vider totalement le cache. Aucun paramètre à configurer.

Note : Ce test peut être utilise lorsque plusieurs scénarios sont enchainés dans un test global.

9.1.4 Vérification d'une valeur dans le cache

Important : Chemin d'accès du test réutilisable `/Snippets/Cache/02_Checking_Cache.tux`

Ce test réutilisable permet de vérifier la valeur dans une clé présente dans le cache.

Paramètre(s) à configurer :

Paramètres	Description
CHECKING	Liste des valeurs à vérifier dans le cache

Les opérateurs disponibles :

Paramètres	Description
contains	Permet de vérifier si la valeur contient une chaîne de caractères
matches	Permet de vérifier si la valeur correspond à l'expression régulière
==	Permet de vérifier si la valeur est égal à
!=	Permet de vérifier si la valeur est différent de
>	Permet de vérifier si la valeur est supérieur à
<	Permet de vérifier si la valeur est inférieur à
>=	Permet de vérifier si la valeur est supérieur égal à
<=	Permet de vérifier si la valeur est inférieur égal à

```
# Vérifie si la valeur contient la chaine de caractère test
[!FROM:CACHE:EXAMPLE:] contains test
```

Note : Il est possible de vérifier plusieurs valeurs en une seule fois

9.1.5 Suppression d'une entrée dans le cache

Important : Chemin d'accès du test réutilisable `/Snippets/Cache/02_Delete_Cache.tux`

Ce test réutilisable permet de supprimer une clé et sa valeur associée dans le cache.

Paramètre(s) à configurer :

Paramètres	Description
MESSAGES	Liste des clés à supprimer

```
# supprime la clé EXEMPLE du cache
[!FROM:CACHE:EXEMPLE:]
```

Note : Il est possible de supprimer plusieurs clés en une seule fois

9.2 Actions basiques

9.2.1 Mise en attente

Important : Chemin d'accès du test réutilisable `/Snippets/Do/01_Wait.tux`

Ce test réutilisable permet d'attendre xx secondes durant l'exécution du test.

Paramètre(s) à configurer :

Paramètres	Description
DURATION	durée en secondes

9.2.2 Arrêt d'un test

Important : Chemin d'accès du test réutilisable `/Snippets/Do/02_Terminate.tux`

Ce test réutilisable permet de forcer l'arrêt d'un scénario en cas d'erreur. Un message expliquant l'arrêt peut être spécifié avec le paramètre `STOP_TEST_MSG`.

Note : Il est possible de personnaliser le message d'arrêt en configurant la variable `STOP_TEST_MSG`.

9.2.3 Chargement d'un environnement de test

Important : Chemin d'accès du test réutilisable `/Snippets/Do/03_Initialize.tux`

Ce test réutilisable permet de charger dans le cache les données de son environnement de tests (les adresses ip, compte d'accès des serveurs, etc.).

Un environnement se décrit avec 4 niveaux :

- `environnement`
- `cluster`
- `noeud`
- `instance`

Un `environnement` peut être constitué de un ou plusieurs clusters.

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

Un `cluster` est constitué d'une liste de noeuds.

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

Un `noeud` est constitué d'une ou plusieurs instances.

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

Une `instance` se constitue de plusieurs clés/valeurs.

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

Paramètre(s) à configurer :

Paramètres	Description
ENVIRONMENT	Lien vers une variable partagée ou bien contient directement du `JSON`.

Exemple d'un environnement de test contenant un serveur http avec une instance de type rest. Après chargement dans le cache, l'instance REST est accessible en utilisant la clé `NODE_HTTP_REST`. L'ensemble des clés présentes dans `COMMON` sont automatiquement copiées dans chaque 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": [    ]
}
```

La clé `DATASET` peut contenir des jeux de données.

9.3 Générateurs

9.3.1 Hash SHA

Important : Chemin d'accès du test réutilisable `/Snippets/Generators/01_Gen_Sha.tux`

Ce test réutilisable permet de générer un hash d'une valeur et de la stocker dans le cache.

Paramètre(s) à configurer :

9.3.2 Hash MD5

Important : Chemin d'accès du test réutilisable `/Snippets/Generators/01_Gen_Md5.tux`

Ce test réutilisable permet de générer un hash md5 d'une valeur et de la stocker dans le cache.

Paramètre(s) à configurer :

Paramètres	Description
DATA_IN	Chaine de caractère à hasher
CACHE_KEY	Nom de la clé ou le résultat sera sauvegarder dans le cache

9.3.3 UUID

Important : Chemin d'accès du test réutilisable `/Snippets/Generators/01_Gen_Uuid.tux`

Ce test réutilisable permet de générer un id uuid et de la stocker dans le cache.

Paramètre(s) à configurer :

Paramètres	Description
CACHE_KEY	Nom de la clé pour sauvegarder le résultat dans le cache

9.3.4 BASE64

Important : Chemin d'accès du test réutilisable `/Snippets/Generators/01_Gen_Base64.tux`

Ce test réutilisable permet d'encoder ou décoder une chaine de caractères et de stocker le résultat dans le cache.

Paramètre(s) à configurer :

9.3.5 GZIP

Important : Chemin d'accès du test réutilisable `/Snippets/Generators/01_Gen_Gzip.tux`

Ce test réutilisable permet de compresser ou décompresser une chaine de caractères et de stocker le résultat dans le cache.

Paramètre(s) à configurer :

9.4 Protocoles réseaux

9.4.1 SSH

Important : Chemin d'accès du test réutilisable `/Snippets/Protocols/01_Send_SSH.tsx`

Ce test réutilisable permet d'envoyer un enchainement de commandes ssh. Il est à utiliser conjointement avec le test réutilisable `/Snippets/Do/03_Initialize.tux` permet de charger un environnement dans le cache.

Paramètre(s) à configurer :

Paramètres	Description
SERVERS	Liste des serveurs à contacter
COMMANDS	Listes des commandes à exécuter sur la machine distante
TIMEOUT_CONNECT	Durée max pour se connecter sur la machine distante

Le paramètre *COMMANDS* attend un ou plusieurs blocs de 4 lignes. Chaque bloc doit respecter le formalisme suivant :

1. Un commentaire expliquant l'action, cette information est utilisée pour initialiser l'étape de test
2. La commande à exécuter
3. La chaine de caractères attendue à l'écran, si la valeur attendue n'est pas trouvée alors l'étape sera en erreur. (ligne optionnelle)
4. vide

Avertissement : Chaque bloc sera exécuté même si le précédent est en erreur.

L'exemple suivant effectue les actions suivantes :

1. Envoi de 3 pings sur la machine distante dont l'ip est stockée dans le cache `DEST_HOST`
2. Vérification de l'obtention d'un message à l'écran indiquant que les 3 paquets ont été envoyés. Ensuite la valeur mdev est stockée dans le cache avec la clé "STATS'
3. Le deuxième bloc efface l'écran en envoyant la commande clear.
4. Enfin le test attend de trouver le prompt à l'écran

```
# 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 : Il est possible d'exécuter le test plusieurs fois en fournissant une liste de serveurs.

Note : Par défaut, le test attend 20 secondes au maximum pour trouver la chaine de caractères attendue. Il est possible de configurer cette valeur avec le paramètre `TIMEOUT`.

Note : Par défaut, le test attend 10 secondes pour effectuer la connexion au serveur distant. Il est possible de configurer cette valeur avec le paramètre `TIMEOUT_CONNECT`.

9.4.2 HTTP

Important : Chemin d'accès du test réutilisable `/Snippets/Protocols/01_Send_HTTP.tsx`

Ce test réutilisable permet d'envoyer une requête HTTP en vérifiant la réponse reçue. Il est à utiliser conjointement avec le test réutilisable `/Snippets/Do/03_Initialize.tux` qui permet de charger un environnement dans le cache.

Paramètre(s) à configurer pour définir la destination :

Paramètres	Description
SERVERS	Liste des serveurs à contacter
TIMEOUT_CONNECT	Durée max pour se connecter sur la machine distante

Paramètre(s) pour configurer la requête HTTP à envoyer :

Paramètres	Description
HTTP_REQ_BODY	Corps de la requête
HTTP_REQ_HEADERS	Liste des headers à ajouter
HTTP_REQ_METHOD	Methode HTTP (GET, POST, etc..)
HTTP_REQ_URI	URI appelée

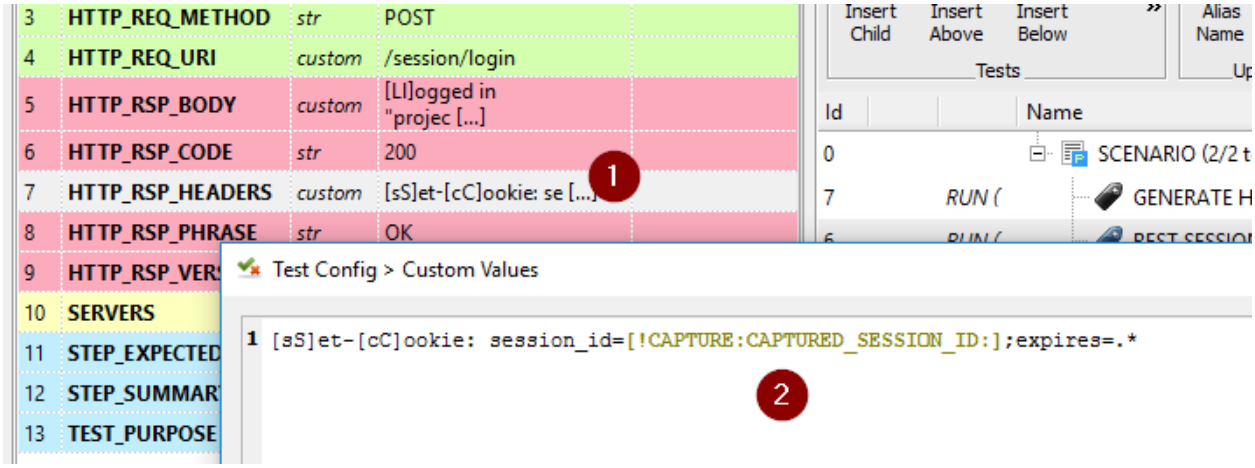
1	**HTTP_REQ_BODY**	*custom*	{ "login": "[!CA [...]
2	**HTTP_REQ_HEADERS**	*custom*	Content-Type: applic [...]
3	**HTTP_REQ_METHOD**	*str*	POST
4	**HTTP_REQ_URI**	*custom*	/session/login

Paramètre(s) pour configurer la réponse HTTP attendue (et qui permettra de considérer le test comme valide) :

Paramètres	Description
HTTP_RSP_BODY	Corps de la réponse attendue.
HTTP_RSP_CODE	Le code HTTP attendue. 200 par défaut
HTTP_RSP_HEADERS	Liste des entêtes attendues
HTTP_RSP_PHRASE	La phrase HTTP attendue. OK par défaut
HTTP_RSP_VERSION	La version HTTP attendue. HTTP/1.[0|1] par défaut

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 : L'utilisation des expressions régulières est possible pour vérifier ou enregistrer une valeur dans le corps de la réponse ou bien dans les entêtes.



9.4.3 XML

Important : Chemin d'accès du test réutilisable `/Snippets/Protocols/01_Send_XML.tsx`

Ce test réutilisable permet d'envoyer une requête HTTP avec du `XML` dans le corps de la requête. Il permet aussi de vérifier la réponse reçue avec plusieurs critères. Il est à utiliser conjointement avec le test réutilisable `/Snippets/Do/03_Initialize.tux` qui permet de charger un environnement dans le cache.

Paramètre(s) à configurer pour définir la destination :

Paramètres	Description
SERVERS	Liste des serveurs à contacter
TIMEOUT_CONNECT	Durée max pour se connecter sur la machine distante

Paramètre(s) pour configurer la requête HTTP à envoyer :

Paramètres	Description
HTTP_REQ_BODY	Corps de la requête
HTTP_REQ_HEADERS	Liste des headers à ajouter
HTTP_REQ_METHOD	Methode HTTP (GET, POST, etc..)
HTTP_REQ_URI	URI appelée

Paramètre(s) pour configurer la réponse HTTP attendue (et qui permettra de considérer le test comme valide) :

Paramètres	Description
HTTP_RSP_BODY	Liste des xpaths à vérifier.
HTTP_RSP_CODE	Le code HTTP attendue. 200 par défaut
HTTP_RSP_HEADERS	Liste des entêtes attendues
HTTP_RSP_NAMESPACES	Liste des namespaces à prendre en compte
HTTP_RSP_PHRASE	La phrase HTTP attendue. OK par défaut
HTTP_RSP_VERSION	La version HTTP attendue. HTTP/1.[0|1] par défaut

Avertissement : Le test sera en erreur si la réponse ne contient pas de XML.

9.4.4 JSON

Important : Chemin d'accès du test réutilisable `/Snippets/Protocols/01_Send_JSON.tsx`

Ce test réutilisable permet d'envoyer une requête HTTP avec du JSON en vérifiant la réponse reçue. Il est à utiliser conjointement avec le test réutilisable `/Snippets/Do/03_Initialize.tux` qui permet de charger un environnement dans le cache.

Paramètre(s) à configurer pour définir la destination :

Paramètres	Description
SERVERS	Liste des serveurs à contacter
TIMEOUT_CONNECT	Durée max pour se connecter sur la machine distante

Paramètre(s) pour configurer la requête HTTP à envoyer :

Paramètres	Description
HTTP_REQ_BODY	Corps de la requête
HTTP_REQ_HEADERS	Liste des headers à ajouter
HTTP_REQ_METHOD	Methode HTTP (GET, POST, etc..)
HTTP_REQ_URI	URI appelée

Paramètre(s) pour configurer la réponse HTTP attendue (et qui permettra de considérer le test comme valide) :

Paramètres	Description
HTTP_RSP_BODY	Liste des json paths à vérifier.
HTTP_RSP_CODE	Le code HTTP attendue. 200 par défaut
HTTP_RSP_HEADERS	Liste des entêtes attendues
HTTP_RSP_PHRASE	La phrase HTTP attendue. OK par défaut
HTTP_RSP_VERSION	La version HTTP attendue. HTTP/1.[0|1] par défaut

Avertissement : Le test sera en erreur si la réponse ne contient pas de JSON.

9.5 Interface utilisateur

9.5.1 Ouverture application Windows

Important : Chemin d'accès du test réutilisable `/Snippets/UI/01_Win_OpenApp.tux`

Tests réutilisables permettant d'ouvrir ou de fermer une application sur un poste Windows ou Linux. Ce test nécessite de définir quel agent sera utilisé avec la clé `AGENT_GUI`.

Paramètre(s) à configurer :

Paramètres	Description
APP_PATH	Chemin d'accès de l'application à ouvrir

9.5.2 Fermeture application Windows

Important : Chemin d'accès du test réutilisable `/Snippets/UI/02_Win_CloseApp.tux`

Test réutilisable permettant d'ouvrir ou de fermer une application sur un poste Windows ou Linux. Ce test nécessite de définir quel agent sera utilisé avec la clé `AGENT_GUI`.

Paramètre(s) à configurer :

Paramètres	Description
APP_NAME	Nom de l'application à fermer

9.5.3 Ouverture navigateur

Important : Chemin d'accès du test réutilisable `/Snippets/UI/03_OpenBrowser.tux`

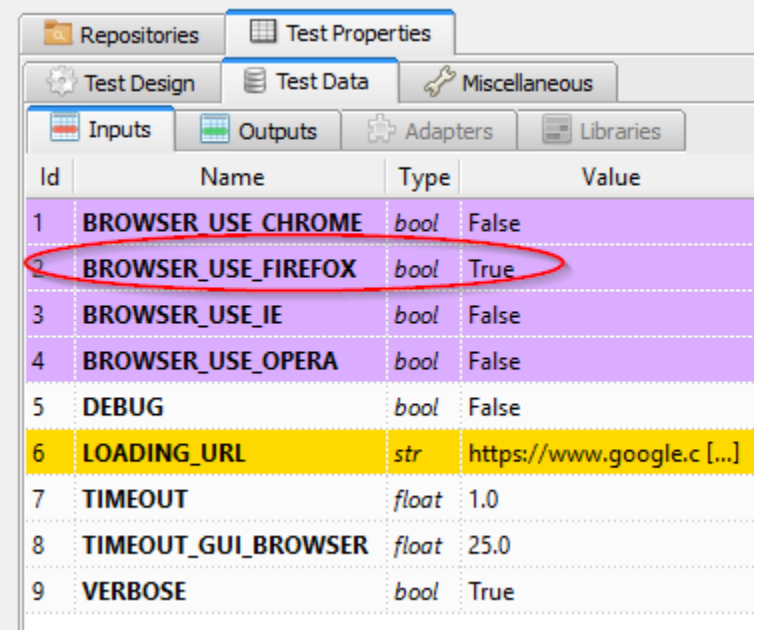
Test réutilisable permettant d'ouvrir ou de fermer une navigateur sur un poste Windows ou Linux. Ce test nécessite de définir quel agent sera utilisé avec la clé `AGENT_GUI_BROWSER`.

Paramètre(s) à configurer :

Paramètres	Description
LOADING_URL	Url du site à charger

Il est possible de sélectionner le navigateur à ouvrir, les navigateurs suivants sont supportés :

- Firefox
- Chrome
- Internet Explorer
- Opera
- Edge



Note : L'url doit obligatoirement commencer par `http://` ou `https://`

9.5.4 Fermeture navigateur

Important : Chemin d'accès du test réutilisable `/Snippets/UI/03_CloseBrowser.tux`

Tests réutilisables permettant d'ouvrir ou de fermer une navigateur sur un poste Windows ou Linux. Ce test nécessite de définir quel agent sera utilisé avec la clé `AGENT_GUI_BROWSER`.

9.6 Vérifications

9.6.1 Contenu de type XML

Important : Chemin d'accès du test réutilisable `/Snippets/Verify/01_Check_XML.tux`

Ce test réutilisable permet de vérifier du contenu de type XML avec l'outil xpath.

Paramètre(s) à configurer :

Paramètres	Description
XML_STR	XML brut à inspecter
XML_XPATH	xpath qui sera vérifier le test
XML_NAMESPACES	Définition des namespaces

Exemple de valeur pour le paramètre `XML_STR` :

```
<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>
```

Exemple de valeur pour le paramètre `XML_XPATH` permettant d'enregistrer dans le cache le nom de la ville à la 2ième position dans la liste.

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

La valeur sera accessible dans le cache avec la clé `CITY`.

9.6.2 Contenu de type JSON

Important : Chemin d'accès du test réutilisable `/Snippets/Verify/01_Check_JSON.tux`

Ce test réutilisable permet de vérifier du contenu de type JSON avec l'outil jsonpath

Paramètre(s) à configurer :

Paramètres	Description
JSON_STR	Json à inspecter
JSON_XPATH	jsonpath qui sera vérifié par le test

Exemple de valeur pour le paramètre `JSON_STR` :

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


Exemple de valeur pour le paramètre `JSON_XPATH` permettant d'enregistrer dans le cache la valeur de la clé *Connection* dans le dictionnaire *headers*.

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

La valeur sera accessible dans le cache avec la clé `CX`.

CHAPITRE 10

Variables globales

Les variables globales sont principalement utilisées pour décrire un environnement de tests pour un projet donné. Elles sont accessibles depuis un test en utilisant le paramètre de test de type `global` ou `list-global`.

10.1 Ajout/suppression d'une variable

L'ajout ou la suppression d'une variable peut se faire à travers l'interface web ou bien depuis l'api. Le format attendu est de type `JSON`. Ces variables sont accessibles automatiquement au niveau de chaque test depuis les propriétés.

10.2 Description environnement de test

La description d'un environnement de test doit respecter le formalisme décrit ci-dessous. Ce type de déclaration est à utiliser avec le test réutilisable qui initialise l'environnement et le met à disposition dans le cache.

Déclaration d'un noeud `SAMPLE_NODE` :

```
{
        "COMMON": {
                "HOSTNAME": "extensivetesting"
        },
        "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,
```

(suite sur la page suivante)

(suite de la page précédente)

```
                                "SSH_AGENT_SUPPORT": false,
                                "SSH_AGENT": {
                                        "type": "ssh",
                                        "name": "agent.ssh01"
                                }
                        }
                }
        }
}
```

Déclaration d'une donnée de test SAMPLE_DATASET_AUTH :

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

Déclaration de l'environnement SAMPLE_ENVIRONMENT :

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

10.3 Import/export des variables

Il est possible d'exporter ou d'importer en masse l'ensemble des variables depuis l'api REST au format CSV.

Avertissement : Les différentes variables sont encodées en base64.

CHAPITRE 11

Conception assistée

Le client lourd comporte un assistant qui permet de créer des tests sans avoir de connaissances en développement. On peut s'en servir pour :

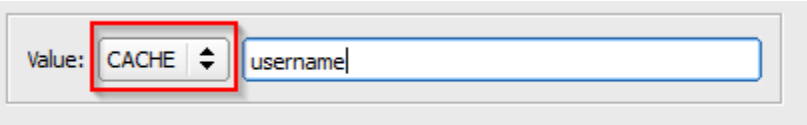
- Utiliser les fonctions basiques du framework
- Exécuter des commandes systèmes (ssh)
- Tester des applications avec un client lourd
- Tester des applications web
- Exécuter des actions sur un mobile Android

Le test se compose d'un enchaînement d'actions à réaliser. L'assistant génère automatiquement un `test unit` ou un `test suite`. Un test (script) existant peut être mis à jour depuis l'assistant aussi.

Pour ajouter une action dans l'assitant, il faut

- sélectionner l'action à réaliser
- la configurer
- enregistrer l'action

L'assistant supporte nativement l'utilisation du cache. Il est donc possible de sauvegarder ou récupérer des valeurs depuis le cache.



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

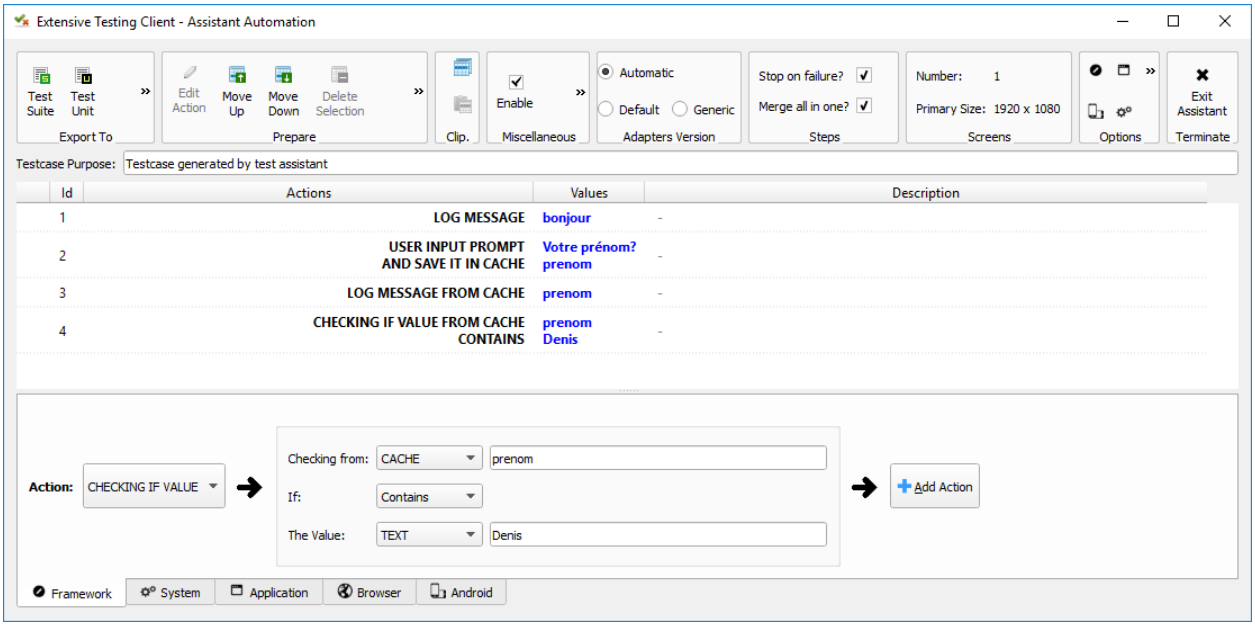
Important : L'assistant permet de générer des tests en mode automatique mais il est aussi possible d'ajouter son propre code à l'intérieur avec l'action `USERCODE`.

11.1 Onglet Framework

L'onglet `framework` permet d'utiliser les fonctions de base du framework de test.

Exemple de test réalisé avec l'assistant :

1. Affiche le message « bonjour » dans le test
2. Demande à l'utilisateur durant l'exécution son prénom et l'enregistre dans le cache avec la clé `prenom`
3. Affiche le prénom dans le log du test
4. Vérifie depuis le cache si le prénom contient une valeur spécifique.



Liste des actions disponibles :

Note : En rouge, les actions indispensables.

`LOG MESSAGE`	Affiche un message d'information durant l'exécution du test
LOG WARNING	Affiche un message d'attention durant l'exécution du test
`SET VALUE`	Sauvegarde une donnée dans le cache
RESET CACHE	Vide complètement le cache
USERCODE	Permet d'ajouter du code personnalisé dans le test
WAIT DURING	Attend pendant xx secondes
`CHECK IF VALUE`	Vérifie si la value contient un texte spécifique
ASK SOMETHING	Demande une valeur à l'utilisateur (mode interaction)

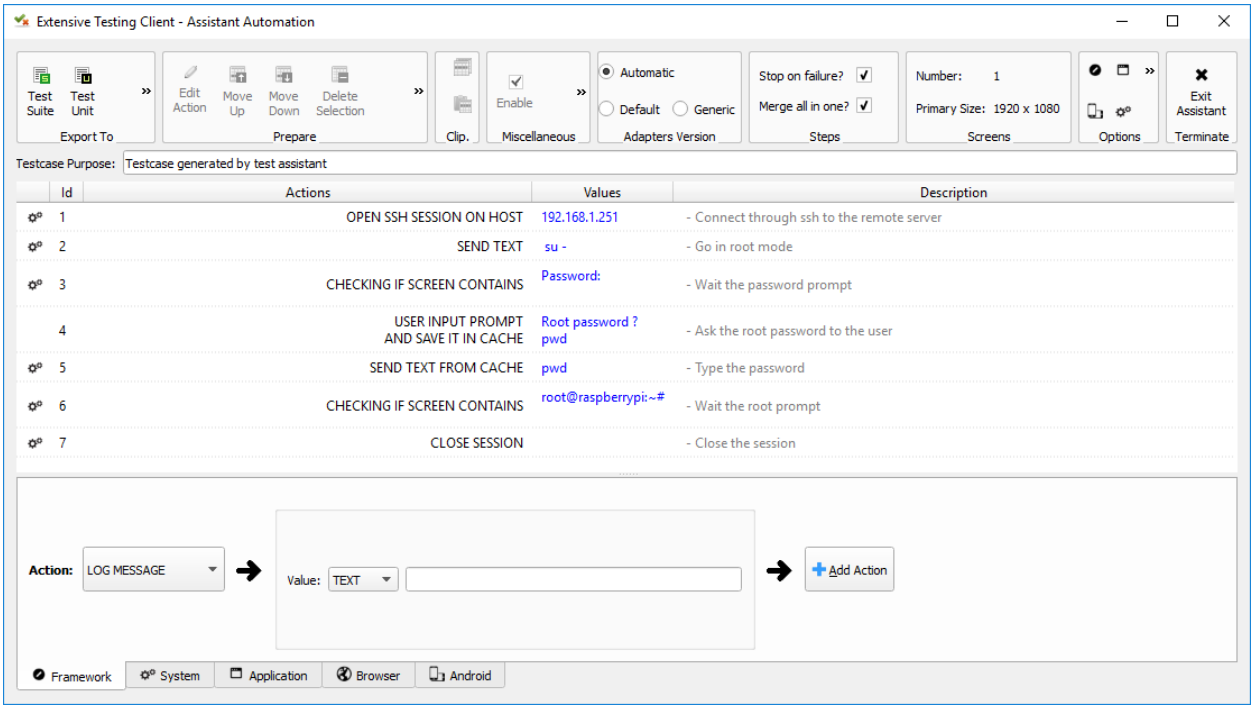
11.2 Onglet Système

L'onglet `système` permet d'exécuter des commandes sur un serveur distant disponible via SSH.

Exemple de test réalisé avec l'assistant :

1. Ouverture de la session ssh sur la machine distante 192.186.1.251

2. Envoi du texte *su -*
3. Attend de détecter le texte *Password :* à l'écran
4. Demande à l'utilisateur le mot de passe root et le stocke dans le cache avec la clé *pwd*
5. Envoi le mot de passe root en utilisant la valeur stockée dans le cache
6. Attend de détecter à l'écran le prompt de connexion
7. Ferme la connexion SSH.



Liste des actions disponibles :

Note : En rouge, les actions indispensables.

`OPEN SSH SESSION`	Ouvre une session SSH
CLOSE SESSION	Ferme la session
CLEAR SCREEN	Vide l'écran
`SEND TEXT`	Envoi une chaîne de caractères
SEND SHORTCUT	Envoi un raccourci clavier (pour interrompre une action)
`CHECKING IF SCREEN`	Vérifie si l'écran contient un texte spécifique

Note : L'utilisation de l'action `OPEN SSH SESSION` est obligatoire avant de pouvoir utiliser les autres disponibles.

11.3 Onglet Application

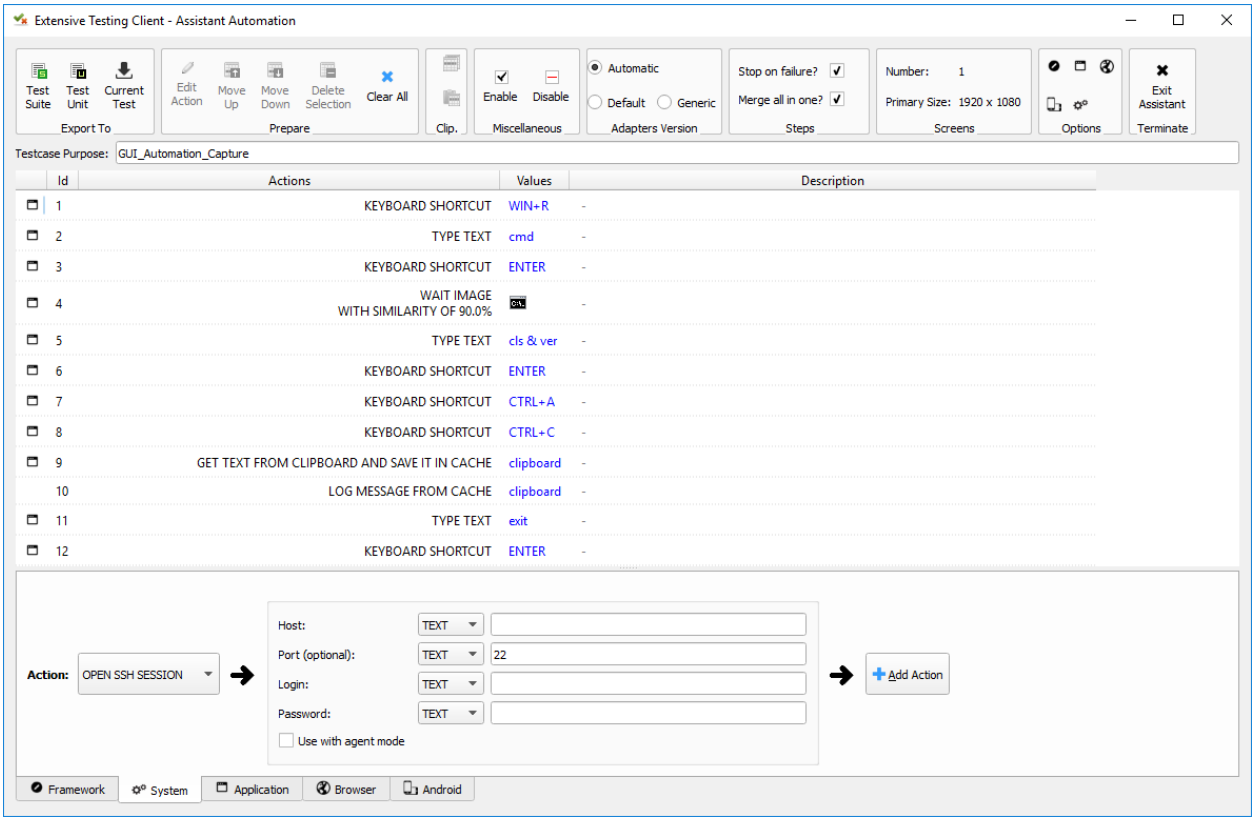
L'onglet `application` permet d'automatiser des applications riches en permettant :
— de simuler le clavier

— de simuler la souris
— de rechercher des élements graphiques à l'écran
— de rechercher du texte

Avertissement : un agent `sikulix-server` est nécessaire pour utiliser les actions.

Exemple de test réalisé avec l'assistant :

1. Envoie le raccourci clavier *Win+R* pour ouvrir la fenêtre exécuter
2. Écrit le texte *cmd*
3. Envoie le raccourci clavier *Enter* pour ouvrir une fenêtre cmd.
4. Attend de détecter l'icône de la fenêtre cmd
5. Écrit le texte *cls & ver* pour afficher la version de Windows
6. Envoie le raccourci clavier *Enter* pour valider
7. Envoie le raccourci clavier *Ctrl+A* pour sélectionner le texte dans la fenêtre
8. Envoie le raccourci clavier *Ctrl+C* pour copier le texte sélectionné dans le presse-papier
9. Récupère le texte du presse papier et l'enregistre dans le cache
10. Affiche le texte copié depuis le cache
11. Écrit le texte *exit* dans la fenêtre cmd
12. Envoie le raccourci clavier *Enter* pour fermer la fenêtre.



Liste des actions disponibles :

Note : En rouge, les actions indispensables.

Contrôle de la souris

CLICK ON POSITION	Clic sur la position (x,y)
DOUBLE CLICK ON POSITION	Double clic sur la position (x,y)
RIGHT CLICK ON POSITION	Clic droit sur la position (x,y)
MOUSE WHEEL DOWN	Tourne la molette de la souris vers le bas
MOUSE WHEEL UP	Tourne la molette de la souris vers le haut
MOVE TO POSITION	Déplace le curseur sur la position (x,y)

Contrôle du clavier

TYPE TEXT	Écrit du texte
TYPE PATH	Écrit du texte (à utiliser pour les chemins d'accès)
TYPE PASSWORD	Écrit du texte (à utiliser pour taper un mot de passe)
GET TEXT FROM CLIPBOARD	Récupère le texte présent dans le presse-papier
KEYBOARD SHORTCUT	Permet de taper un raccourci clavier

Contrôle chaîne de caractères

CLICK ON WORD	Recherche un mot à l'écran et clic dessus
DOUBLE CLICK ON WORD	Recherche un mot à l'écran et double-clic dessus
RIGHT CLICK ON WORD	Recherche un mot à l'écran et effectue un clic-droit dessus
WAIT WORD	Recherche un mot jusqu'à ce qu'il apparaisse
WAIT AND CLICK ON WORD	Recherche un mot jusqu'à ce qu'il apparaisse et clic dessus

Contrôle d'images

CLICK ON IMAGE	Recherche une image et clic dessus
DOUBLE CLICK ON IMAGE	Recherche une image et double-clic dessus
RIGHT CLICK ON IMAGE	Recherche une image et effectue un clic-droit dessus
WAIT IMAGE	Recherche une image jusqu'à la voir apparaître à l'écran
WAIT AND CLICK ON IMAGE	Recherche une image jusqu'à la voir apparaître à l'écran et clic dessus
HOVER MOUSE ON	Recherche une image et déplace le curseur de la souris dessus
DRAG IMAGE AND DROP TO	Recherche une image et effectue un drag and drop vers la position (x,y)

11.4 Onglet Navigateur

L'onglet `navigateur` permet d'automatiser des applications web en permettant :

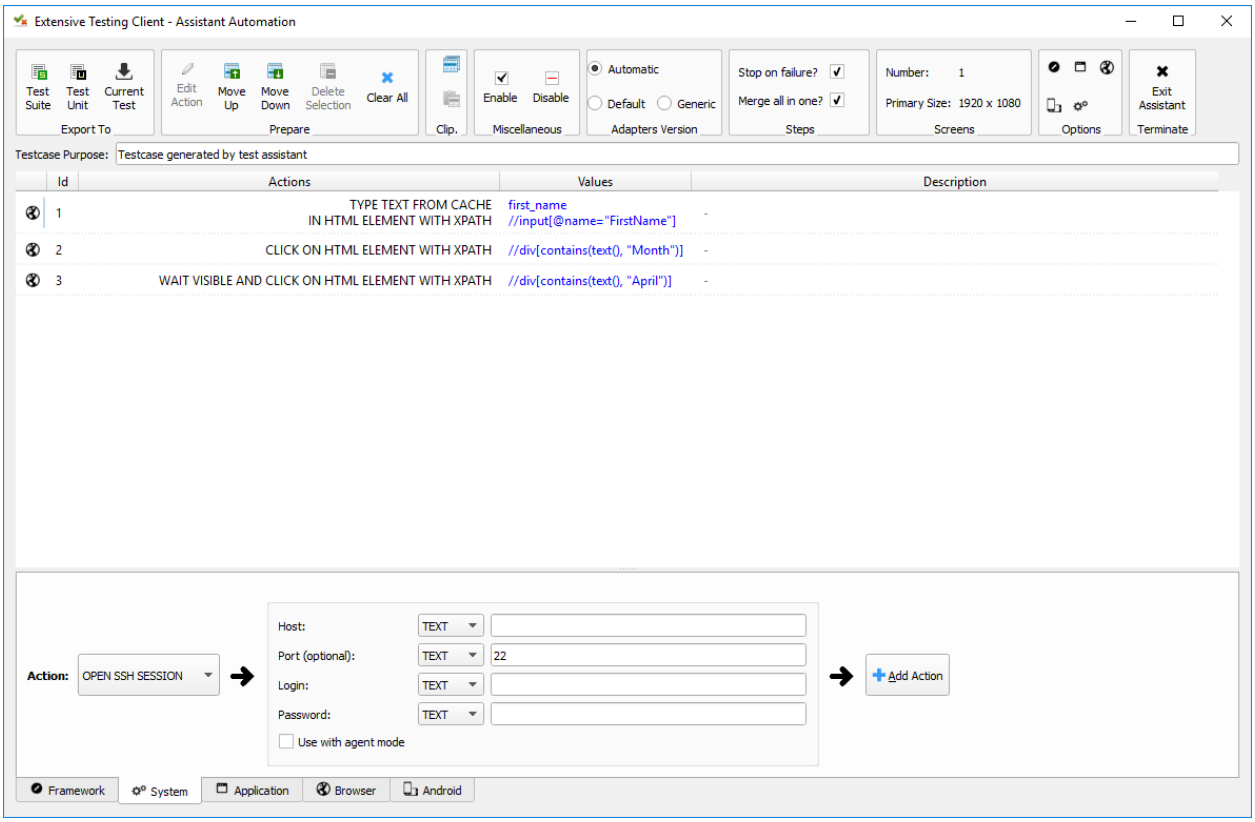
— de piloter les navigateurs (firefox, internet explorer, chrome, edge)
— de simuler le clavier

Avertissement : un agent `selenium3-server` ou `selenium2-server` est nécessaire pour utiliser les actions.

Astuce : Pour cliquer sur un élement HTML, il est conseillé d'utiliser systématiquement la fonction `WAIT VISIBLE AND CLICK ON HTML ELEMENT`.

Exemple de test réalisé avec l'assistant :

1. Récupère depuis le cache le prénom et l'envoie dans l'élément HTML trouvé par le xpath
2. Clic sur l'élément HTML trouvé par le xpath
3. Recherche l'élément HTML trouvé par le xpath et clic dessus dès qu'il est visible à l'écran.



Note : Il est possible d'ouvrir plusieurs navigateur en parallèle sur le même poste à définissant une nouvelle session. La nom se la session se définit sur l'action `OPEN BROWSER`. Il faut ensuite utiliser l'action `SWITCH TO SESSION` pour changer de session.

Liste des actions disponibles :

Note : En rouge, les actions indispensables.

Contrôle navigateur

Actions de navigation

REFRESH PAGE	Rafraîchissement de la page
GO BACK	Retour arrière
GO FORWARD	Go forward
ACCEPT ALERT	Valide l'alerte javascript
DISMISS ALERT	Dismiss the javascript alert
CLOSE CURRENT WINDOW	Ferme la fenêtre courante
SWITCH TO NEXT WINDOW	Bascule sur la fenêtre suivante
SWITCH TO FRAME	Bascule sur la frame suivante
SWITCH TO SESSION	Bascule sur une autre session selenium
SWITCH TO WINDOW	Bascule sur la frame suivante

Actions javascript

EXECUTE JAVASCRIPT ON HTML ELEMENT	Permet d'injecter du javascript script sur un élement html

Actions sur les éléments html

WAIT HTML ELEMENT	Attend l'apparition d'un élément HTML précis
WAIT AND CLICK ON HTML ELEMENT	Attend l'apparition d'un élément HTML précis et clic dessus
WAIT VISIBLE HTML ELEMENT	Attend qu'un élément HTML soit visible à l'utilisateur
WAIT NOT VISIBLE HTML ELEMENT	Attend qu'un élément HTML ne soit pas visible à l'utilisateur
WAIT VISIBLE AND CLICK ON HTML ELEMENT	Attend qu'un élément HTML soit visible à l'utilisateur et clic dessus
HOVER ON HTML ELEMENT	Déplace le curseur de la souris sur un élement HTML précis
CLICK ON HTML ELEMENT	Clic sur un élément HTML précis
DOUBLE CLICK ON HTML ELEMENT	Double clic sur un élément HTML précis
CLEAR TEXT ON HTML ELEMENT	Vide le texte sur un élément HTML précis
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)

Récupération de texte

GET TEXT ALERT	Récupère le texte d'un message alerte javascript
GET TEXT FROM HTML ELEMENT	Récupère le texte un élément html précis
GET PAGE TITLE	Récupère le titre de la page
GET PAGE URL	Récupère l'url de la page
GET PAGE CODE SOURCE	Récupère le code source la page

Simulation clavier

TYPE KEYBOARD SHORTCUT	Envoie un raccourci clavier sur un élément HTML précis
TYPE TEXT ON HTML ELEMENT	Envoie du texte sur un élément HTML précis

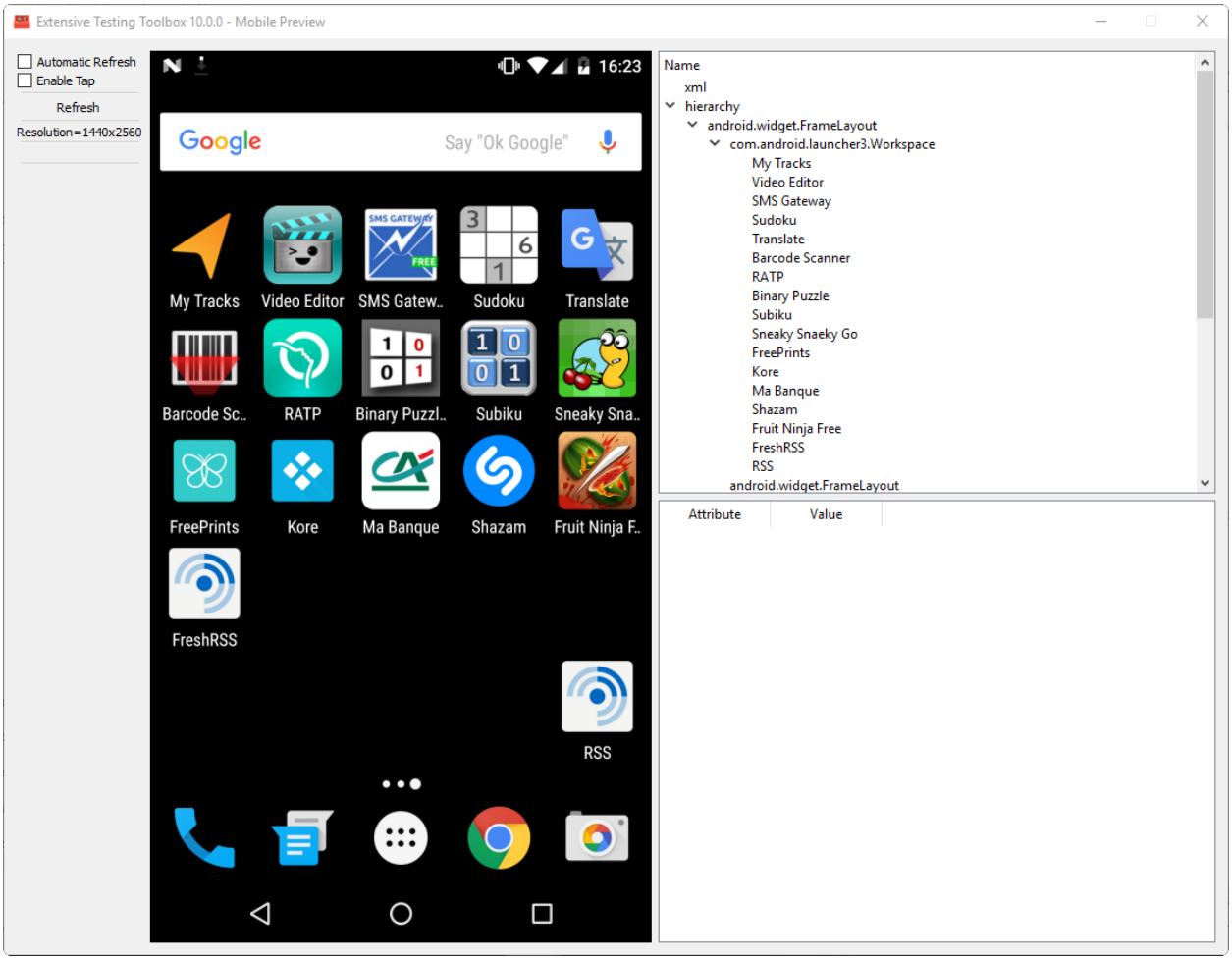
11.5 Onglet Android

L'onglet `android` permet d'automatiser des applications mobiles en permettant :
— de simuler le clavier

— de simuler l'utilisation du doigts sur l'écran
— de piloter le système et les applications

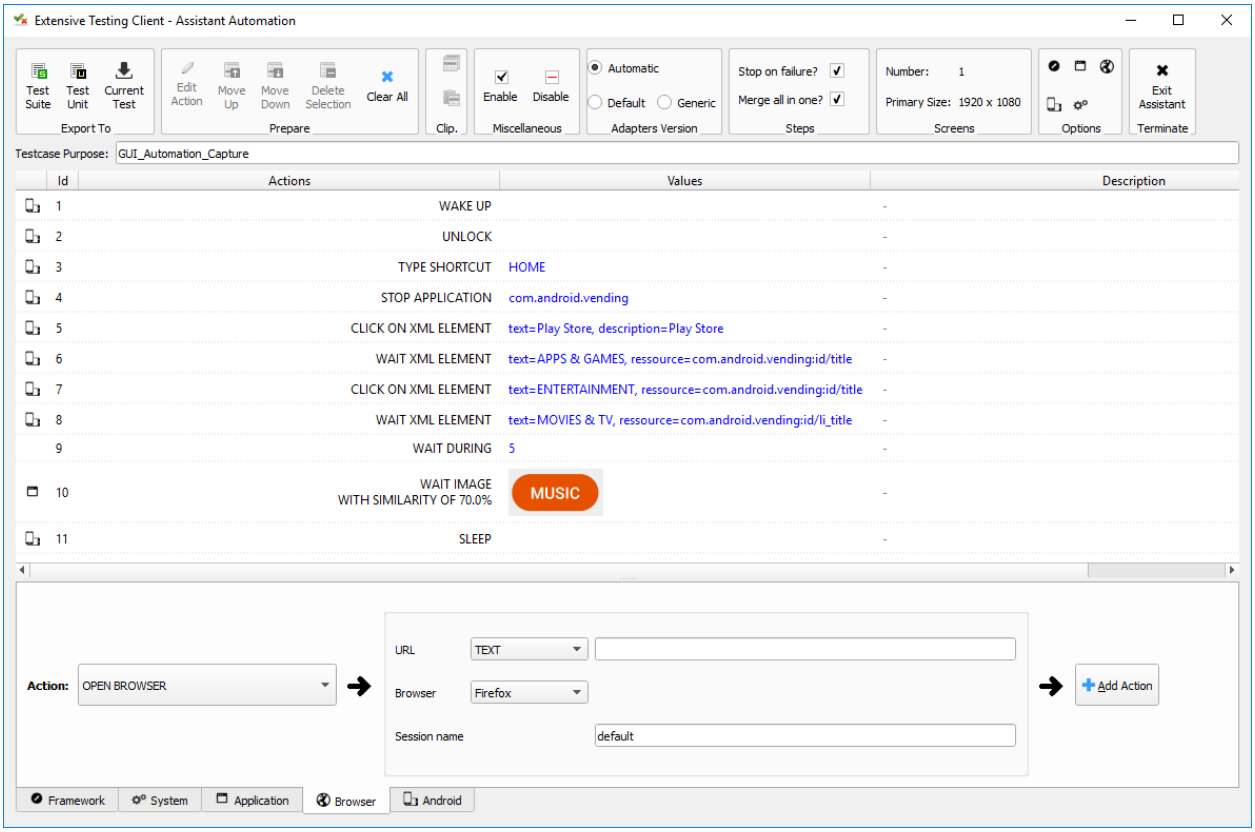
Avertissement : un agent adb est nécessaire pour utiliser les actions.

Aperçu de l'agent



Exemple de test réalisé avec l'assistant :

1. Réveille l'appareil
2. Débloque l'appareil
3. Clic sur le bouton *HOME*
4. Arrête l'application
5. Clic sur l'application *Play Store* pour l'ouvrir
6. Attend que l'application s'ouvre et recherche le menu *APPS & GAMES*
7. Clic sur le texte *ENTERTAINMENT*
8. Clic sur le menu *MOVIES & TV*
9. Attend pendant 5 secondes
10. Recherche l'image
11. Mise en veille de l'appareil.



Liste des actions disponibles :

Note : En rouge, les actions indispensables.

Contrôle du mobile

WAKE UP AND UNLOCK	Réveille et débloque l'appareil
REBOOT	Redémarrage de l'appareil
SLEEP	Mise en veille

Textes

TYPE SHORTCUT	Simule un raccourci
TYPE TEXT ON XML ELEMENT	Envoie du texte sur un élément précis de l'interface
GET TEXT FROM XML ELEMENT	Récupère le texte d'un élément précis de l'interface

Contrôles des éléments XML

CLEAR XML ELEMENT	Supprime le texte d'un élément précis de l'interface
CLICK ON XML ELEMENT	Clic sur un élément précis de l'interface
LONG CLICK ON XML ELEMENT	Clic longue-durée sur un élément précis de l'interface
WAIT AND CLICK ON XML ELEMENT	Attend l'apparition d'un élément précis de l'interface et clic dessus

Tap sur l'écran

CLICK TO POSITION	Clic sur la position x,y
DRAG FROM POSITION	Drag depuis la position x1,y1 vers x2,y2
SWIPE FROM POSITION	Swipe depuis la position x1,y1 vers x2,y2

CHAPITRE 12

Dépannage

12.1 Code erreurs

Erreurs liées au moteur d'exécution

Code erreur	Description
ERR_TE_000	Erreur générique au niveau de l'exécution du test
ERR_TE_001	Erreur générique au niveau de l'exécution du cas de test
ERR_TE_500	Erreur générique au niveau de l'exécution du script

Erreurs liées aux étapes de tests

Erreurs liées à l'accès aux paramètres de test

Code erreur	Description
ERR_PRO_004	Le paramètre demandé dans la fonction *input* n'existe pas dans le test

12.2 FAQ

12.2.1 Impossible de générer la description HTML d'un test

L'impossibilité de générer la description d'un test peut venir de plusiers choses :

- la version des adaptateurs ou librairies utilisés dans le test n'est pas la bonne
- une erreur de syntaxe existe dans un test
- un appel au cache est utilisé dans la définition des étapes de tests

CHAPITRE 13

Installation

13.1 Serveur

L'installation est détaillée sur github

13.2 Plugins pour le serveur

L'installation des plugins pour le serveurest détaillée sur github

13.3 Client Web

L'installation est détaillée sur github

13.4 Client lourd

L'installation est détaillée sur github

13.5 Boîte à outils

L'installation est détaillée sur github

CHAPITRE 14

Administration

14.1 Arrêt/relance du serveur

Le serveur peut être contrôlé en utilisant la commande `./extensiveautomation`. Cette commande permet
- de démarrer ou arrêter le serveur
- de vérifier le status du serveur
- d'installer un adaptateur
- de générer la clé API
- d'afficher la version du serveur.

Pour démarrer le serveur il faut utiliser la commande `./extensiveautomation start`.

```
# ./extensiveautomation start
```

Pour arrêter le serveur il faut utiliser la commande `./extensiveautomation stop`.

```
# ./extensiveautomation stop
```

Astuce :

Il est possible de vérifier dans les logs si le serveur est correctement démarré ou arrêté.

```
# tailf Var/Log/output.log
2014-12-06 11:00:54,092 - INFO - Extensive Automation successfully started (in 1 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 Status du serveur

La commande permet de vérifier le status du serveur, il y a 3 status possible

— `starting` : le serveur est en cours de démarrage
— `running` : le serveur est en cours d'exécution
— `stopped` : le serveur est arrêté.

14.3 Configuration du serveur

Le fichier `settings.ini` contient l'ensemble des paramètres de configuration du serveur. Les paramètres de configuration sont découpés en plusieurs sections :

— Boot
— Notifications
— Client_Channel
— Agent_Channel
— WebServices
— TaskManager
— Network
— Paths
— Bin
— Server
— Bind
— Misc
— Trace
— Supervision
— Users_Session

14.4 Scripts crontab

Les scripts sont disponibles dans le répertoire `Build` depuis les sources du serveur.

`cron.cleanup-testsresult` : ce script permet de supprimer les résultats plus vieux que 30 jours. Le nombre de jours est configurable.

CHAPITRE 15

Projets

La solution peut être utilisée en mode multi-projet. Il est donc possible d'organiser les tests par projets et d'accorder des droits d'accès pour les utilisateurs.

Note : Le projet `Common` existe par défaut, il est accessible par l'ensemble des utilisateurs et ne peux pas être supprimé.

15.1 Ajout d'un projet

L'ajout d'un projet peut se faire avec un compte administrateur. La création d'un projet nécessite de préciser son nom et peut se faire via l'interface web ou bien l'API

15.2 Suppression d'un projet

La suppression d'un projet peut se faire avec un compte administrateur. Cette action peut se faire à travers l'interface web ou bien l'API.

Note : Si le projet est associé à un utilisateur, la suppression n'est pas autorisée.

15.3 Associer un projet à un utilisateur

Un utilisateur peut accéder à plusieurs projets, en fonction des autorisations accordées. L'autorisation s'effectue depuis le profil d'un utilisateur avec un compte administrateur. Un administrateur peut sélectionner les projets autorisés, il est aussi possible de configurer le projet par défaut, ie. celui qui sera affiché par défaut à la connexion.

CHAPITRE 16

Utilisateurs

La solution peut être utilisée en mode multi-utilisateur. Des utilisateurs suivants existent par défaut :

- `Admin`
- `Tester`
- `Monitor`

Note : Ne pas oublier de désactiver les comptes par défaut dans un environnement de production.

16.1 Ajout d'un utilisateur

L'ajout d'un utilisateur peut se faire avec un compte administrateur. La création d'un utilisateur nécessite à minima les paramètres suivants et peut se faire via l'interface web ou bien l'API

- nom d'utilisateur
- mot de passe

Note : L'email est utilisée par la solution pour envoyer les rapports de tests et résultats.

Note : Il est possible de configurer plusieurs adresses email pour un utilisateur en les séparants avec ;

16.2 Suppression d'un utilisateur

La suppression d'un utilisateur peut se faire avec un compte administrateur. Cette action peut se faire à travers l'interface web ou bien l'API.

CHAPITRE 17

Dépannage

17.1 Récupération des logs

17.1.1 Serveur

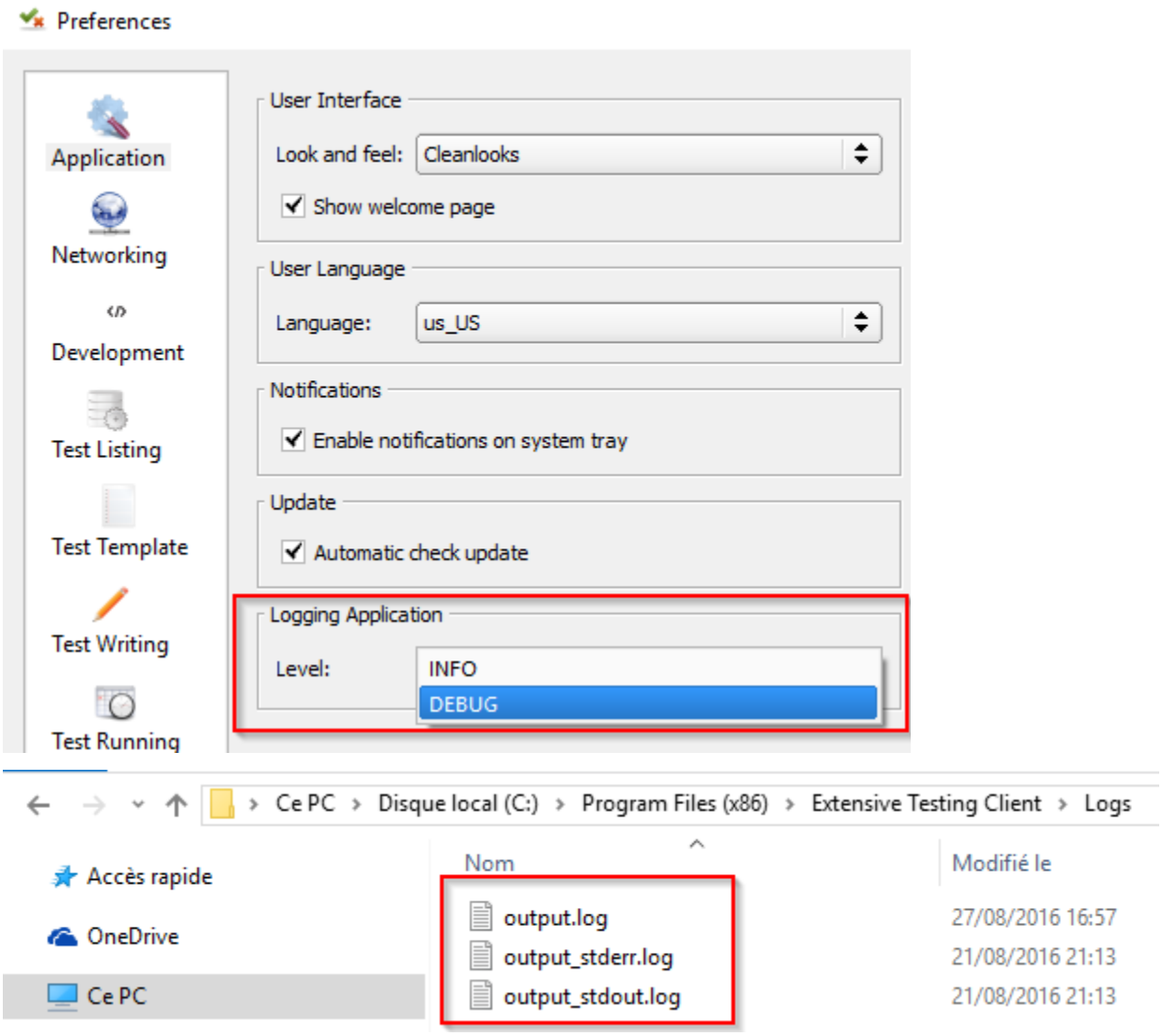
Les logs serveurs sont stockés sur `[...]Var/Logs/`. Les logs sont configurés en mode `INFO` par défaut. Le niveau DEBUG peut être activé depuis le fichier `settings.ini`.

Note : Il est possible de changer le niveau de logs à chaud en faisant un `./extensiveautomation reload`

17.1.2 Client

Les logs du client sont disponibles dans `<Program Files>/Extensive Automation Client/Logs/` Les logs sont configurés en mode `INFO` par défaut.

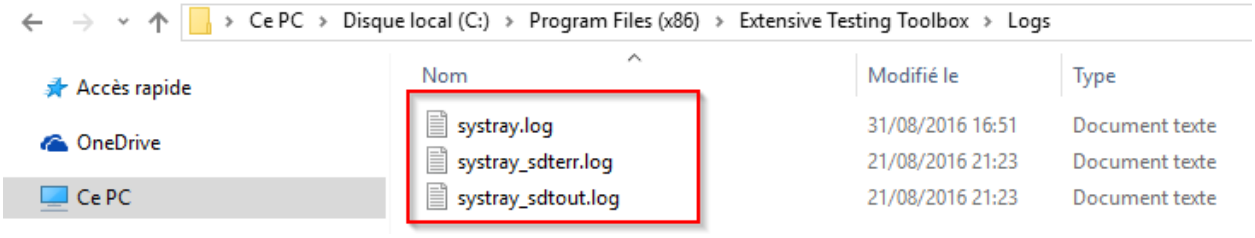
Le niveau DEBUG peut être activé depuis les préférences du client.



17.1.3 Boîtes à outils

Les logs de la boîte à outils sont disponibles dans `<Program Files>/Extensive Automation Toolbox/Logs/`
Les logs sont configurés en mode `INFO` par défaut.

Le niveau DEBUG peut être activé depuis le fichier `settings.ini`.

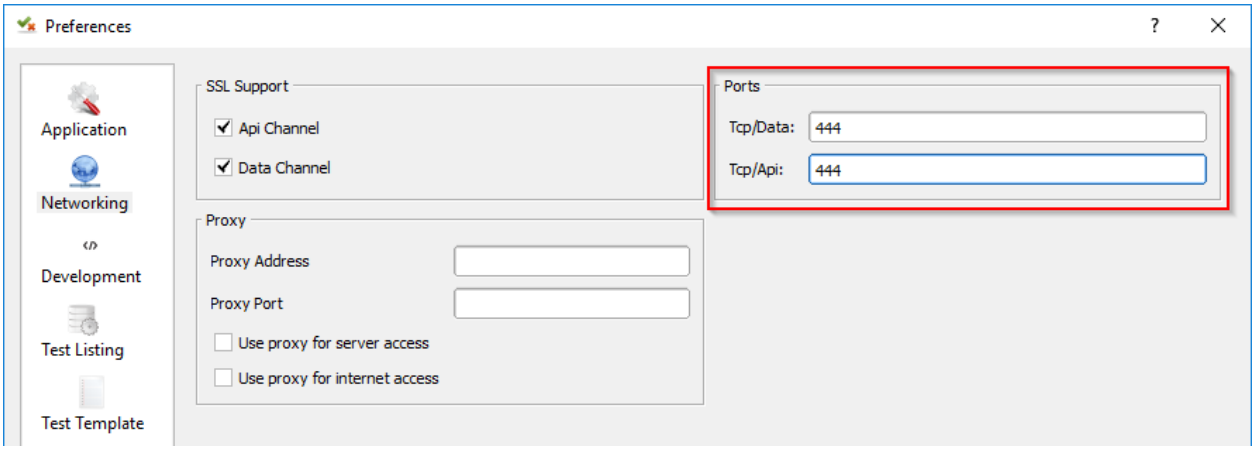


Note : Un redémarrage de la boîte à outils est nécessaire pour prendre en compte le changement

17.2 Foire aux questions

17.2.1 Comment changer le port de connexion du client ?

Le port de destination peut être modifié depuis les préférences du client. Ou bien directement depuis le fichier `settings.ini`.



17.2.2 Afficher la version du serveur ?

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

17.2.3 Quoi faire si ma connection au serveur ne fonctionne pas ?

Si la connection depuis le client au serveur ne fonctionne pas, une analyse est nécessaire.

Le 1er reflex à avoir est de se connecter sur le serveur en SSH et d'exécuter la commande `./extensiveautomation --status` pour vérifier si le serveur tourne.

1. Si le serveur est en cours d'exécution alors il faut vérifier :

— la connectivité réseau en le client et le serveur
— un parefeu bloquant le flux https (443)

2. Si la connectivité réseau est bonne et que le serveur fonctionne (ou pas), il faut vérifier les logs. Le fichier est disponible dans le répertoire `(...]Var/Logs/output.log`. Il faut rechercher les messages de type `ERROR`

17.2.4 Comment corriger l'erreur « hping3 n'est pas installé » ?

Cette erreur apparait durant l'exécution d'un test quand l'adaptateur `Pinger` est utilisé. En effet nécessite d'avoir la librairie système hping3 d'installée sur le serveur.

Il faut récupérer les sources depuis https://github.com/antirez/hping et les compiler :

```
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
```

(suite sur la page suivante)

(suite de la page précédente)

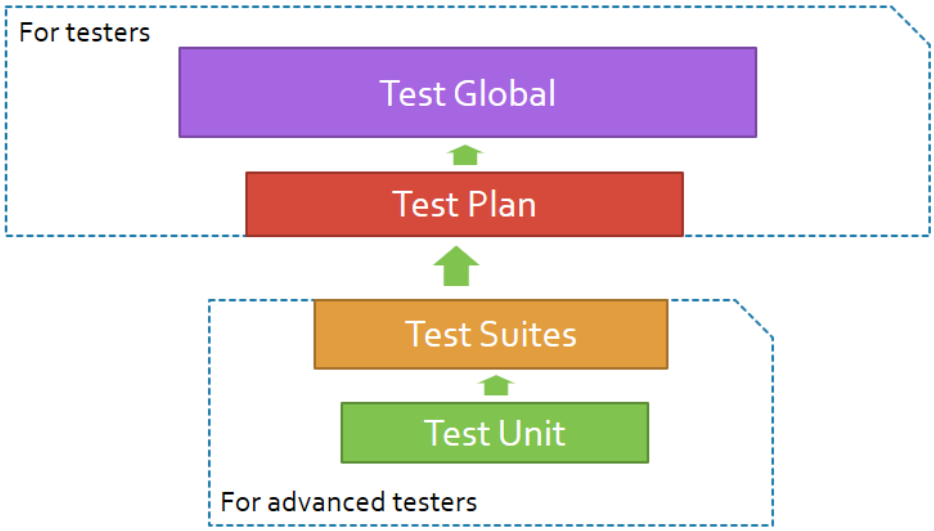
```
make
make install
```

CHAPITRE 18

Les types de tests

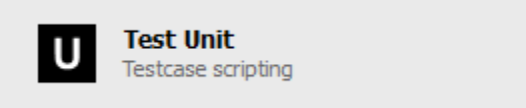
La solution se base sur différents types de tests pour :

— permettre la construction de tests avancés
— diminuer l'utilisation de script



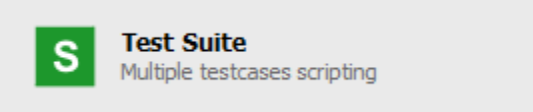
18.1 Test Unit

Le `test unit` (tux) permet d'écire un cas de test avec plusieurs étapes. Ce format est orienté développement.



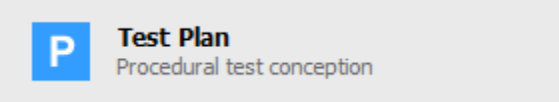
18.2 Test Suite

Le `test suite` (tsx) permet d'écire plusieurs cas de test avec plusieurs étapes. Ce format est orienté développement.



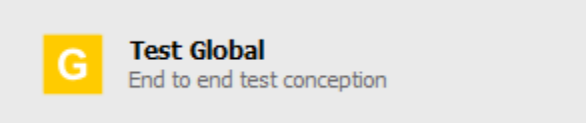
18.3 Test Plan

Le `test plan` (tpx) permet d'écrire des scénarios de test. La conception se réalise en imbriquant les tests *abstract*, *unit* et *suite* Ce format de test necéssite aucune connaissance en développement.



18.4 Test Global

Le `test global` (tgx) permet d'écrire des campagnes de test. La préparation des campagnes se réalise en important les tests *plans*.



Note : Il est aussi possible d'importer les autres types de tests.

CHAPITRE 19

Les fondamentaux

Le framework de test fournit un cadre permettant de normaliser la création des cas de tests.

Les principales fonctionnalités sont :

— le support des cas de tests avec étapes
— le support des extensions permettant de communiquer avec le système à tester ou piloter
— la génération automatique des rapports de tests.

19.1 Cas de test

La création d'un cas de test dans la solution est normalisée.

Un cas de test se découpe en 4 sections :

— `description` : description des différentes étapes du test
— `prepare` : préparation des adaptateurs et librairies permettant de communiquer avec le système à tester ou piloter
— `definition` : déroulement du test
— `cleanup` : phase de nettoyage

Le résultat d'un cas de test est automatiquement calculé par le framework lorsque le test est terminé en fonction des différentes étapes définies.

Il existe 3 résultats possibles :

— `PASS` : toutes les étapes du tests ont été exécutées avec succès
— `FAILED` : au moins une étape est en erreur après exécution
— `UNDEFINED` : au moins une étape du test n'a pas été exécutée

Note : La section `cleanup` est systématiquement appelée, même en cas d'erreur.

19.2 Etapes de test

Un cas de test se découpe en sous-étapes.

Une étape se définit par :

— un résumé de l'action à réaliser
— la description détaillée de l'action à réaliser
— la description du résultat attendu pour valider l'étape.

La définition des étapes du test doit être faite dans la section `description` :

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

Le résultat d'une étape est à préciser dans la section `definition`

Exemple pour mettre le résultat à `PASS` ou `FAILED`

```
self.step1.setPassed(actual="step executed as expected")

self.step1.setFailed(actual="error to run the step")
```

Avertissement : Il ne faut pas oublier de démarrer une étape avec la fonction `start` sinon il n'est pas possible de mettre le résultat.

Note : Il ne faut pas oublier de préciser le résultat d'une étape, sinon il sera considéré comme `UNDEFINED`.

Important : Une étape positionnée à `FAILED` ne pourra pas devenir `PASS` par la suite dans un test.

19.3 Annulation d'un test

Il est possible de forcer l'exécution d'un cas de test en utilisant la fonction `interrupt` dans la section `description` de votre test.

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

Utiliser la fonction `interrupt` permet d'arrêter le test et d'appeler automatiquement la section `cleanup` du cas de test. Dans ce cas précis, l'argument `aborted` est mis à True par le framework pour indiquer l'annulation du test.

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

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

19.4 Ajout de trace

Le framework met à disposition certaines fonctions pour ajouter des messages durant l'exécution d'un test.

Les niveaux suivants sont disponibles :

— Exemple pour afficher un message de type `info`

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

— Exemple pour afficher un message de type `warning`

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

— Exemple pour afficher un message de type `error`

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

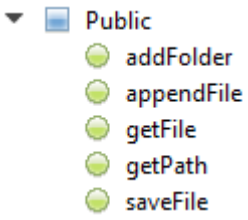
Note : Si un message de niveau `error` est affiché alors le résultat sera automatiquement mis à `FAILED`.

Note : Les messages apparaissent automatiquement dans le rapport basique.

19.5 Données

19.5.1 Public

Un espace public est disponible sur le serveur de test. Cet espace permet de mettre à disposition des fichiers qui sont nécessaire durant l'exécution d'un test.

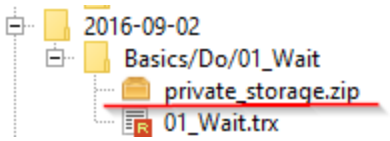


Les fichiers sont stockés dans le répertoire `/opt/xtc/current/Var/Public/` sur le serveur.

Avertissement : Cet espace est commun à l'ensemble des projets configurés sur le serveur.

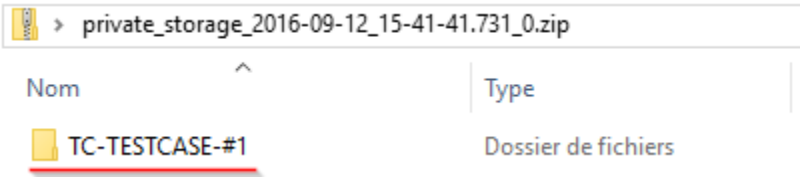
19.5.2 Privé

L'espace de stockage privé n'existe que durant l'exécution d'un test. Il permet de sauvegarder des logs générés ou récupérés lors de l'exécution du test. Ces logs sont automatiquement mis à la disposition de l'utilisateur dans un fichier zip lorsque le test est terminé. Ils sont récupables depuis le client ou bien depuis l'API du serveur.



Les logs sont organisés par répertoire :

- Répertoire TC-TESTCASE-#<id_tc> : contient les logs générés par le cas de test
- Répertoire ADP-#<id_adp> : contient les logs générés par les différents adaptateurs utilisés durant le test



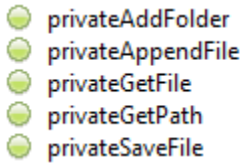
Exemple pour sauvegarder le texte *hello world* dans un fichier *my_logs* depuis le cas de test

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

Exemple pour ajouter du texte dans un fichier de log déjà existant

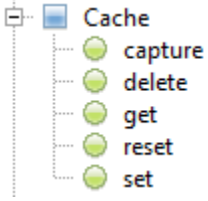
```
Private(self).appendFile(destname="my_logs", data="hello world2")
```

Note : Il est aussi possible de sauvegarder des fichiers depuis un adaptateur. Ils seront automatiquement stockés dans un répertoire portant le nom de l'adaptateur.



19.5.3 En cache

Le framework de test permet de stocker en cache des données sous la forme clé/valeur. Cette fonction peut être nécessaire pour partager des données entre tests lors de l'écriture d'un scénario par exemple.



Exemple pour sauvegarder une valeur dans le cache

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

Lire une valeur depuis le cache

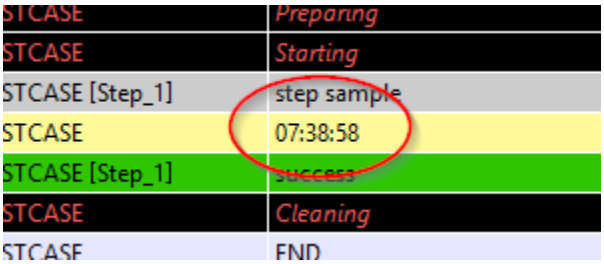
```
my_data= Cache().get(name="my_data")
Trace(self).warning(my_data)
```

Exemple pour capturer une donnée avec une expression régulière et avec enregistrement dans le 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") )
```



Il est aussi possible de s'appuyer sur un paramètre de type `custom` pour fournir l'expression régulière.

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

ou en mode `greedy`

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

Important : Le cache n'existe que durant l'exécution d'un test.

19.6 Mettre en attente

Cette fonction permet de faire une pause durant l'exécution d'un test.

Exemple de mise en attente pendant 10 secondes :

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

Exemple de mise en attente tant que la date et heure courante ne correspondent pas à la date indiquée :

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

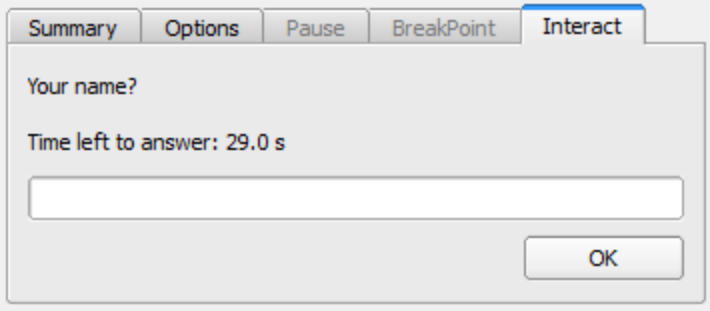
19.7 Interaction avec le testeur

Le framework permet d'écrire des tests semi-automatiques, c'est à dire en mode interaction. Cette fonction peut être intéressante pour faire un test en mode question/réponse (ex : configuration d'un équipement)

Exemple demandant le nom de la personne :

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

Depuis le client, l'onglet `Interact` apparait automatiquement pour répondre à la question posée durant l'exécution du test. Cette fenêtre est disponible depuis le fenêtre d'analyse.



Note : Si aucune réponse n'est fournie dans le temps imparti, il est possible de fournir une valeur par défaut avec l'argument `default`.

19.8 Paramètres d'un test

19.8.1 Paramètres

Les paramètres entrants (inputs) sont à utiliser pour ajouter des variables sur un test. Ils sont configurables depuis le client.

Les principaux types à utiliser sont :

Type	Description usage
text	variable de type chaîne de caractères en mode avancé
json	variable de type JSON en mode avancé
global	les variables globales par projets

Il existe plusieurs autres types de paramètres :

Type	Description usage
str/pwd	chaîne de caractères
list	liste de chaînes de caractères
bool	valeur booléen
hex	valeur hexadécimale
none	valeur nulle
alias	raccourci paramètre
list-shared	liste de valeurs de variables projets
cache	clé d'une valeur présente dans le cache
int	entier
float	décimal
dataset	intègre un fichier de type dataset
remote-image	intègre une image présente dans le dépôts de tests
local-image	intègre une image présente en local sur un le poste
snapshot-image	intègre une capture d'écran
local-file	intègre un fichier présent en local sur le poste
date	date
time	heure
date-time	date et heure
self-ip	liste des adresses IP du serveur
self-mac	liste des adresses MAC du serveur
sef-eth	liste des interfaces réseau du serveur

Les variables sont accessibles depuis un test avec la fonction `input(...)`

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

Le paramètre custom

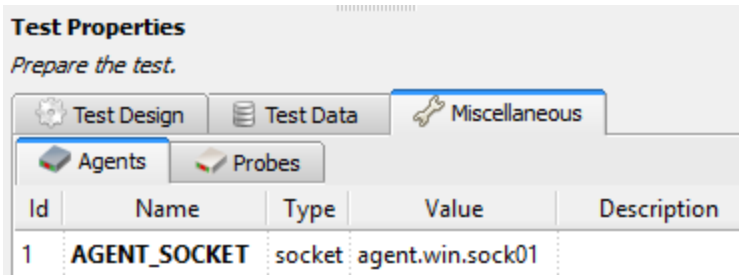
Le type `custom` permet de construire des paramètres utilisants d'autre paramètre ou le cache. Ils est donc possible d'utiliser des mots clés qui seront interprétés par le framework de test au moment de l'exécution.

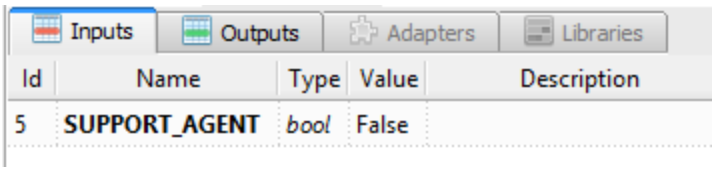
Liste des mots clés disponibles :

Mots clés	Description
`[!INPUT: :]`	Permet de récupérer la valeur d'un paramètre présent dans le test
`[!CACHE: :]`	Permet de récupérer une valeur présente dans le cache

Note : Le nom d'un paramètre est unique et obligatoirement en majuscule.

19.8.2 Les agents



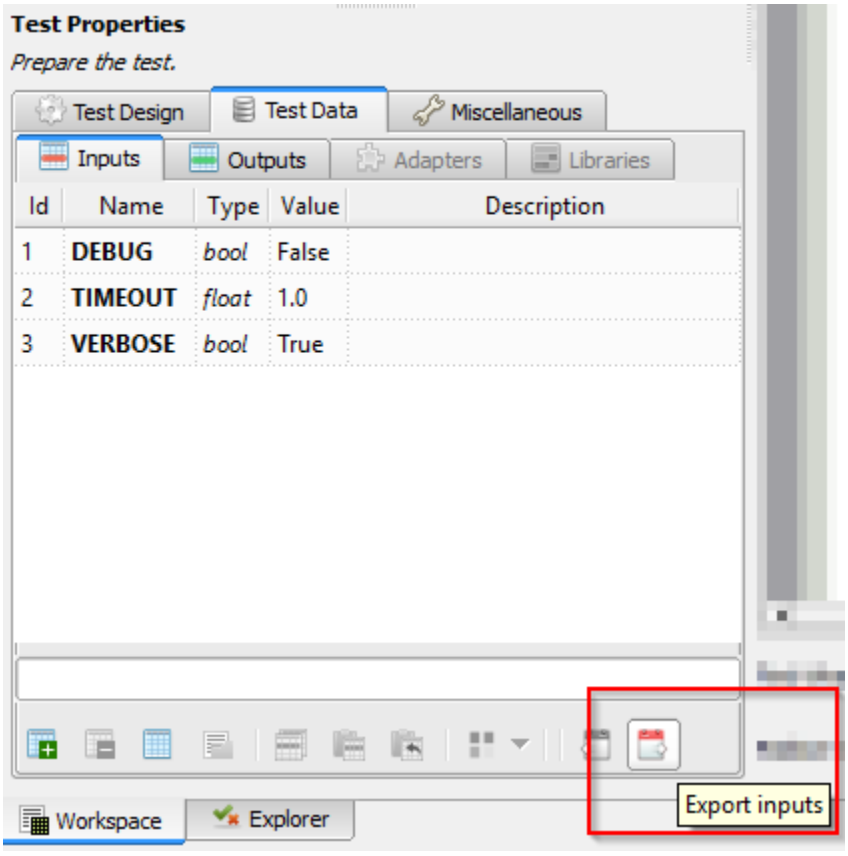


Il est possible d'accéder à la liste des agents depuis un test en utilisant le mode clé `input()`.

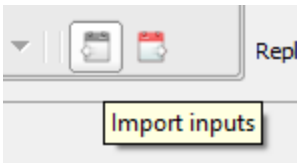
```
self.ADP_REST= SutAdapters.REST.Client(
                                        parent=self,
                                        destinationIp=input('HOST'),
                                        destinationPort=input('PORT'),
                                        debug=input('DEBUG'),
 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

19.8.3 Import/export des paramètres

Les paramètres de tests peuvent être exportés dans un type de fichier dédié `testconfig` (tcx). Il est donc possible de travailler/préparer les paramètres sans avoir le test.



A l'inverse il est possible d'importer un fichier de configuration dans un test. L'import écrasera l'ensemble des paramètres si le nom est identique.


Repl
Import inputs

CHAPITRE 20

La traçabilité

20.1 Les évènements

L'exécution d'un test se découpe en évènements, l'ensemble de ces évènements sont stockés et peuvent être visualisés après coup. Un évènement peut représenter :

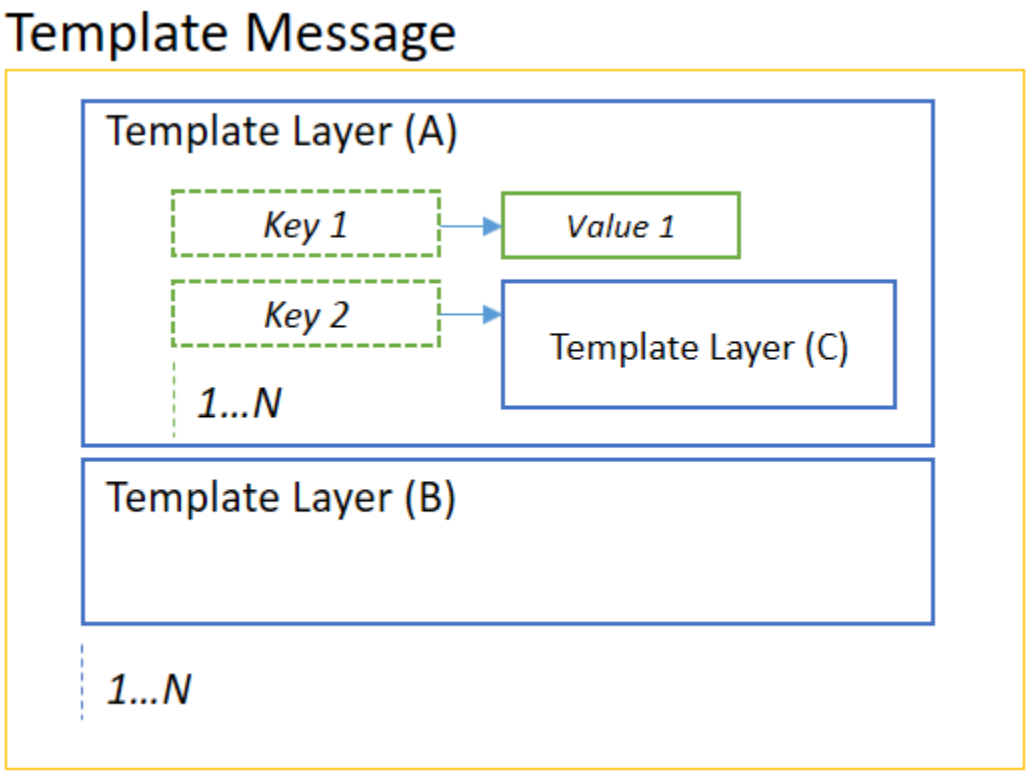
- une action effectuée par le framework de test
- une action effectuée par le test
- une donnée reçue par le système à tester ou contrôler.
- une donnée à envoyer au système à tester ou contrôler.

L'exécution évènementielle permet d'avoir des tests robustes grâce à la définition des intervalles d'observations. L'approche consiste à écrire les tests avec le formalisme suivant :

- J'exécute une action dans mon test.
- Pendant un interval donné, je regarde et compare tous les évènements reçues avec un attendu.
- **Je décide de l'action suivante**
  - après avoir reçu l'évènement que j'attendais
  - ou bien quand l'intervalle d'observation est terminé.

Durant l'exécution d'un test, le framework capture tous les évènements générés par le système testé ou piloté. Les évènements sont ensuite convertis et stockés dans un message appelé `modèle`.

Un `modèle` se découpe en une ou plusieurs `couches`. Une `couche` se définit par un ensemble de clé/valeur. La valeur d'une couche peut être une autre couche aussi.



20.1.1 Création d'un modèle

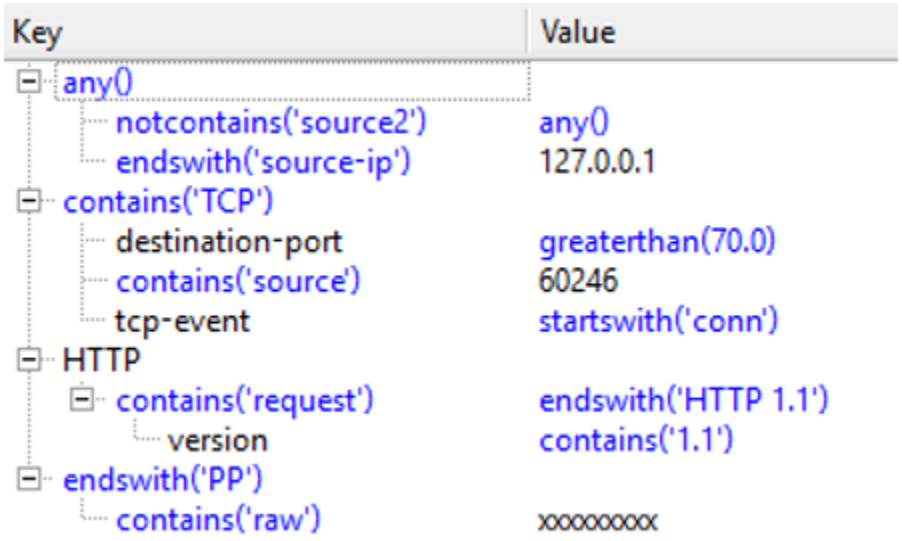
La création d'un modèle peut se faire en utilisant le framework de test `TestTemplates`.

```
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)
```

Ce modèle indique que l'évènement devra contenir :

- une couche s'appelant *response* et contenant la clé *code* et *msg*
- la clé code devra être strictement égale à la valeur 500
- la clé msg devra être strictement égale au texte hello world.

Exemple d'un message attendu visible depuis le client graphique :



20.1.2 Les opérateurs

Des opérateurs sont disponibles pour faciliter la comparaison des modèles reçus.

Nom	Description
Any	Accepte toute les valeurs
Contains	Vérifie si la chaîne de caractères contients des caractères
Endswith	Vérifie si la chaîne de caractères se termine par
Startswith	Vérifie si la chaîne de caractères commence par
GreaterThan	Vérifie si un entier est plus grand que
LowerThan	Vérifie si un entier est plus petit que
RegEx	Vérifie si la chaîne de caractères répond à l'expression régulière

Exemple de modèle utilisant les opérateurs de comparaison :

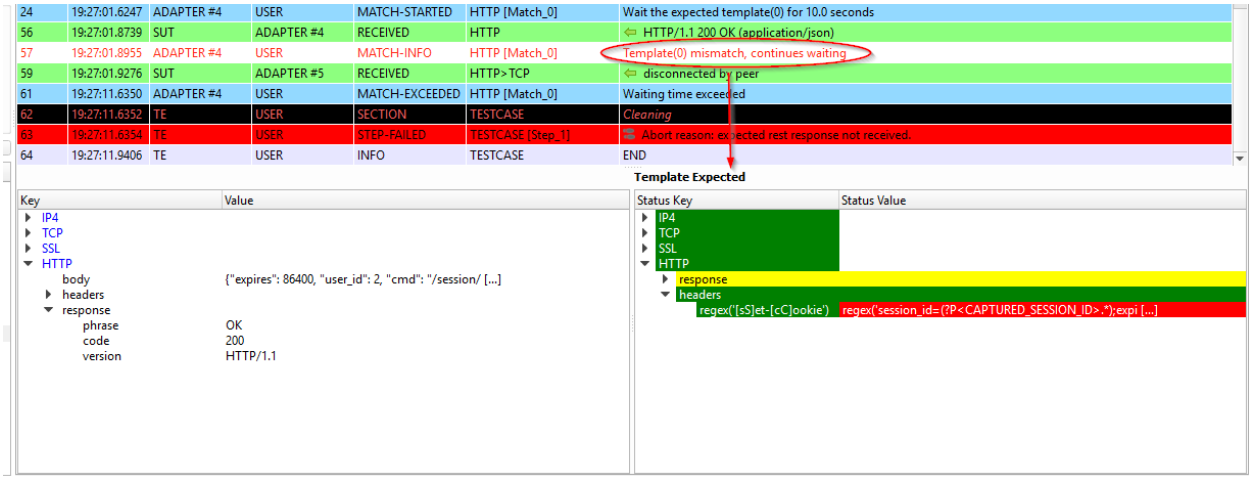
```
tpl = TestTemplates.TemplateMessage()
layer = TestTemplates.TemplateLayer(name='response')
layer.addKey(name='code', data=TestOperators.LowerThan(x=500)))
layer.addKey(name='msg', data=TestOperators.Contains(x="hello"))
tpl.addLayer(layer=layer)
```

Ce modèle indique que l'évènement devra contenir :

— une couche s'appelant *response* et contenant la clé *code* et *msg*
— la clé code devra être inférieur à la valeur 500
— la clé msg devra contenir le texte hello.

20.1.3 La visualisation

Le client permet de visualiser graphiquement la comparaison effectuée par le framework.



Définition du code couleur :

Vert	Correspondance parfaite entre la valeur reçue et attendue
Rouge	La valeur reçue ne correspond pas à la valeur attendue
Jaune	La valeur attendue n'a pas été vérifiée

20.2 Les rapports de tests

Après chaque exécution d'un test, le framework génère automatiquement les rapports de tests associés.

Il existe 2 type rapports :

- Un rapport avancé
- Un rapport basique (accessible par défaut depuis le client graphique)

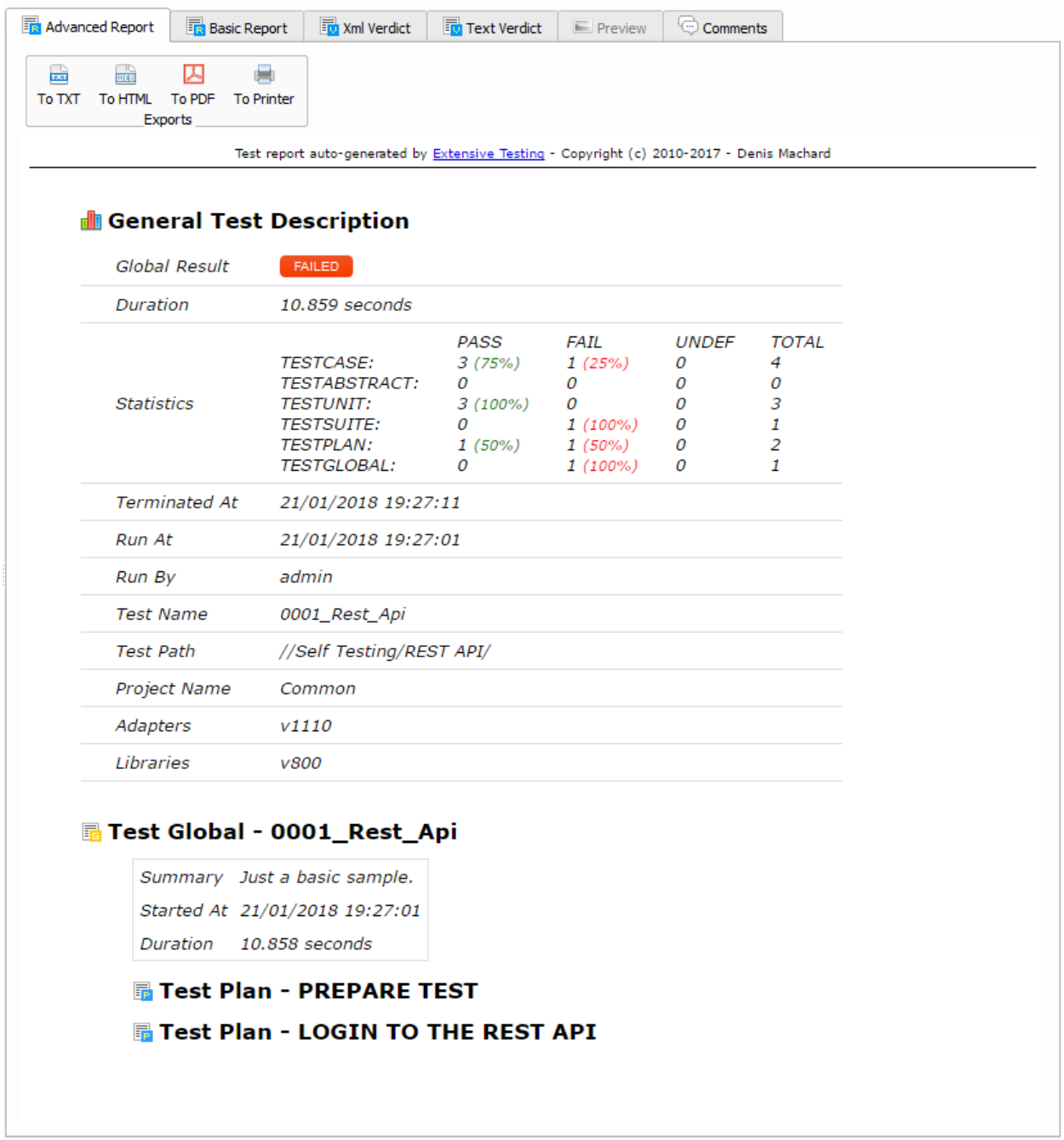
Les rapports sont accessibles depuis le client, l'interface web ou bien depuis l'API.

Note : Les rapports peuvent être exportés au format html, csv, xml et pdf.

20.2.1 Rapport avancé

Le rapport avancé affiche les informations comme :

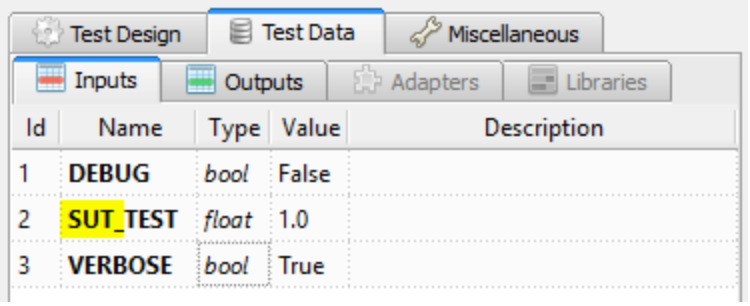
- la durée d'exécution de chaque cas de test
- la description complète des étapes de test.
- des statistiques sur l'exécution.
- les paramètres de tests.

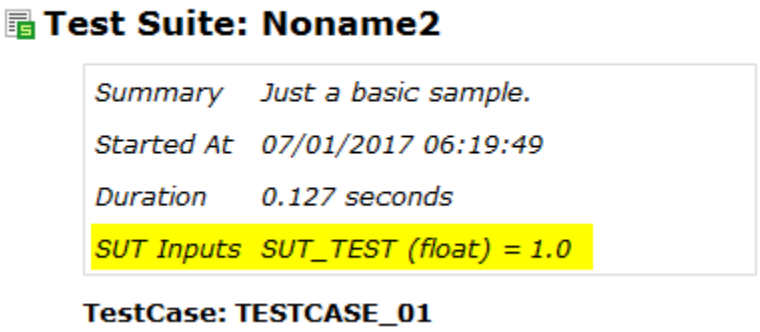


Il est possible d'afficher des variables dans le rapport de test en préfixant les variables :

- `SUT_` Variables décrivant la version du système à tester ou piloter
- `DATA_` Variables décrivant des données spécifiques
- `USER_` Variables utilisateurs

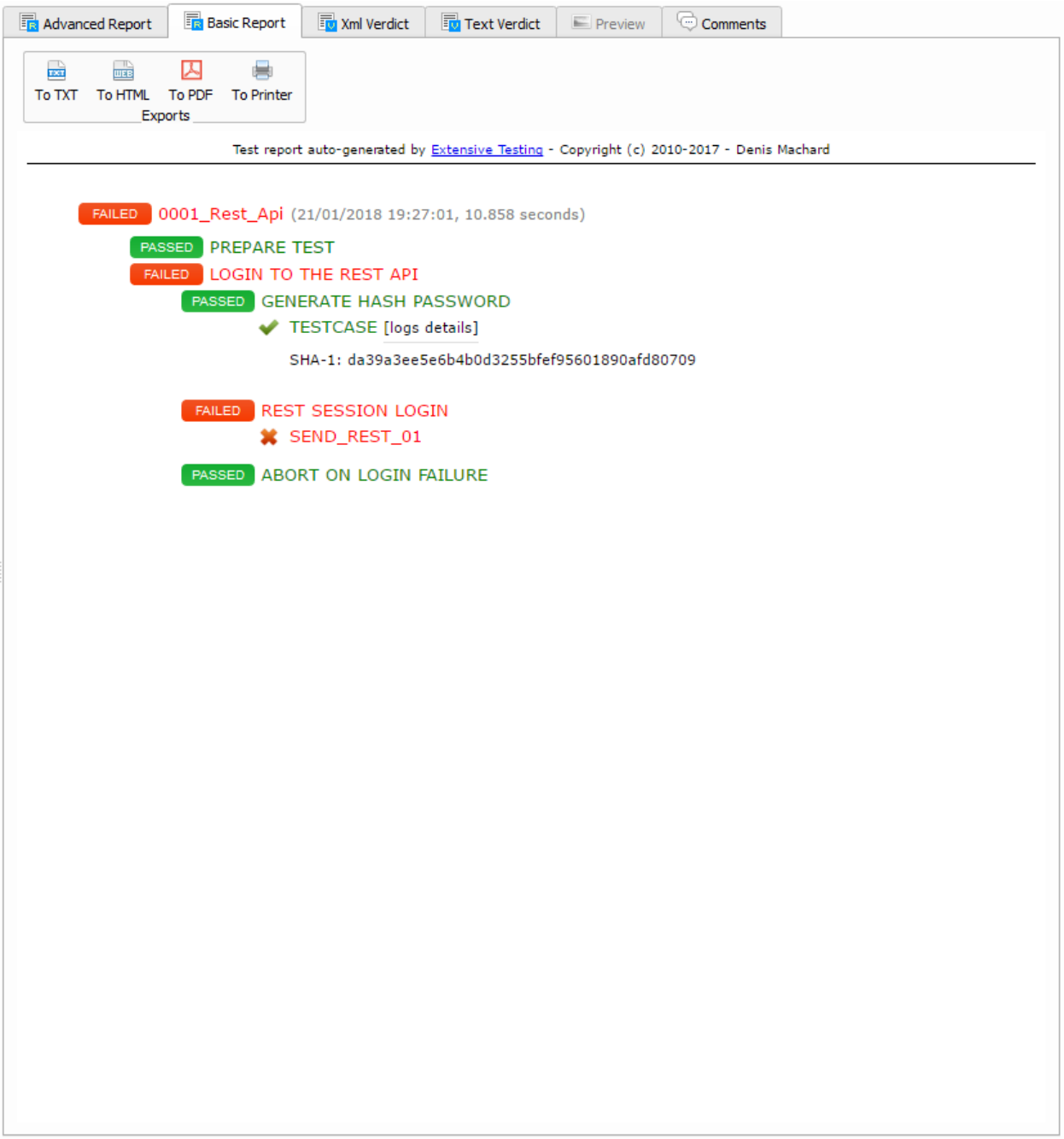
Cette fonctionnalité peut être utile pour augmenter le niveau de traçabilité dans les rapports.





20.2.2 Rapport basique

Le rapport basique résume le résultat de l'ensemble des cas de tests et des états.



Code couleur :

Vert	Le cas de test est valide
Rouge	Le cas de test est en erreur
Orange	Le résultat du cas de test n'est pas déterminé
Gris	Le cas de test n'a pas été exécuté

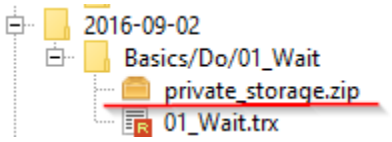
Astuce : Il faut cliquer sur les cas de tests pour afficher les étapes.

Note : Les messages affichés par le test avec la fonction `Trace(self).info()` sont disponibles dans le rapport en cliquant sur le lien `[logs details]`.

Les erreurs sont aussi affichées en cliquant sur le lien `[errors details]`.

20.3 Les logs

Le framework permet d'enregistrer des logs durants l'exécution d'un test et de les mettre à disposition rapidement auprès de l'utilisations. L'ensemble des logs supplémentaires sont zippés et accessibles depuis le client lourd ou bien l'API.



Note : Pour plus de détaills, il faut lire le chapitre *Les fondamentaux >> Données*.

CHAPITRE 21

L'interopérabilité

21.1 Adaptateurs

Les adaptateurs permettent de communiquer avec le système à tester ou piloter. La solution embarque plusieurs a

- support de protocoles réseaux
- support de protocoles niveau application
- communication avec les bases de données
- interaction systèmes
- interaction avec les interfaces graphiques
- support de protocoles télécom

Les adaptateurs ont deux modes d'utilisation :

- un mode direct : la communication se fait directement depuis le serveur de test vers le système à contrôler.
- un mode agent : la communication avec le système à contrôler se fait par l'intermédiaire d'un agent communiquant avec le serveur de test.

Note : Le mode `Verbose` est activé par défaut sur tous les adapateurs. Ce mode peut être désactivé pour réduire le nombre d'évènements durant un test.

Note : Le mode `Debug` n'est pas activé par défaut. Il peut être activé en cas de problème.

Note : Des exemples sont disponibles dans les échantillons de tests `Samples/Tests_Adapters`

Liste des adaptateurs disponibles par défaut :

21.1.1 Protocoles réseaux

Adaptateurs	Agents	Descriptions
ARP	socket	Sniffer permettant d'envoyer et recevoir des packets ARP
ICMP	socket	Sniffer permettant d'envoyer et recevoir des packets ICMP
Ethernet	socket	Sniffer permettant d'envoyer et recevoir des frames Ethernet
IP	socket	Sniffer permettant d'envoyer et recevoir des packets IPv4
Pinger	non supporté	Tests de vie de machines via ICMP, TCP ou une URL
UDP/TCP	socket	Sniffer et client UDP et TCP
NTP	socket	Client permetttant de requêter un serveur NTP
DNS	non supporté	Client résolveur
SNMP	socket	Réception d'alarmes SNMPv2

21.1.2 Protocoles réseaux applications

Adaptateurs	Agents	Descriptions
HTTP	socket	Serveur et client avec le support TLS et proxy
SOAP	socket	Client avec le support TLS et proxy
REST	socket	Client avec le support TLS et proxy
WebSocket	socket	Client websocket
SoapUI	soapui	Client permettant d'exécuter des campagnes SoapUI

21.1.3 Commandes systèmes

Adaptateurs	Agents	Descriptions
Dig		Client dig
Curl		Client curl
Nmap		Client nmap
Ncat		Client ncat
Openssl		Client openssl

21.1.4 Interfaces utilisateurs

Adaptateurs	Agents	Descriptions
Adb	adb	Intégration avec la passerelle Android
Selenium	selenium2-server ou selenium3-server	Intégration avec le projet Selenium
Sikuli	sikulix-server	Intégration avec le projet SikuliX

21.1.5 Bases de données

Adaptateurs	Agents	Descriptions
Microsoft SQL	database	Communication avec une base de type Microsoft SQL
MySQL	database	Communication avec une base de type MySQL/MariaDB
PostgreSQL	database	Communication avec une base de type PostgreSQL

21.1.6 Contrôles systèmes

Adaptateurs	Agents	Descriptions
SSH/SFTP	ssh	Console SSH
TELNET	socket	Client permettant d'envoyer et recevoir du texte
FTP	ftp	Client avec support TLS
System File	file	Permet l'interaction avec les fichiers systèmes Linux ou Windows
System Win/Unix	command	Permet de contrôler les systèmes Linux et Windows (wmic)
Cisco Catalyst	ssh	Client de configuration, basé sur l'adaptateur Telnet

21.1.7 Protocoles Télécoms

Adaptateurs	Agents	Descriptions
SMS Gateway	gateway-sms	Permet de recevoir ou d'envoyer des SMS en utilisant un smartphone Android
SIP	socket	Téléphone SIP
RTP	socket	Module permettant d'envoyer et recevoir des flux audios et vidéos

21.2 Librairies

Une librairie permet de mettre à disposition rapidement des fonctions pour

- supporter les méthodes de chiffrement de données
- supporter les formats de compression existants
- supporter les fonctions d'authentification
- manipuler les différents format de date, heure et unités
- supporter les codecs (XML, JSON, etc. . .)
- supporter les fonctions de hash de données

Une librairie ne communique pas en direct avec le système à tester ou piloter. Elle est utilisée :

- directement depuis les tests
- depuis les adaptateurs.

Astuce : Si plusieurs adaptateurs ont besoin des mêmes fonctions, il est conseillé de les factoriser dans une librairie.

Liste des librairies disponibles par défauts :

21.2.1 Chiffrement

AES	Support chiffrement ou déchiffrement
Blowfish	Support chiffrement ou déchiffrement
OpenSSL	Permet d'exécuter la commande SSL
RC4	Support chiffrement ou déchiffrement
XOR	Support chiffrement ou déchiffrement
RSA	Générateur clé RSA

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/02_Ciphers`

21.2.2 Codecs

Base64	Encode ou décode au format base64
Excel	Lecture de fichier excel
G711A	Encode ou décode le codec audio
G711U	Encode ou décode le codec audio
JSON	Encode ou décode du texte au format JSON
XML	Encode ou décode du texte au format XML

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/03_Codecs`

21.2.3 Compression

GZIP	Compression ou décompression au format GZIP

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/09_Compression`

21.2.4 Hashing

Checksum	Générateur de checksum
HMAC	Création d'un hash md5, sha1 et sha256
MD5	Création d'un hash md5
SHA	Création d'un hash sha1, sha256 et sha512
CRC32	Générateur de checksum

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/05_Hashing`

21.2.5 Identifiant

SessionID	Générateur de session ID
UUIDS	Générateur de UUID (Universally Unique IDentifier)

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/07_Identifiers`

21.2.6 Média

ChartsJS	Générateur de graphique visible dans les rapports de test
DialTones	Générateur de tonalité
Image	Manipulation des images
Noise	Générateur de bruit
SDP	Décode ou encode des messages SDP
WavContainer	Création de fichier audio de type WAV
Waves	Générateur d'ondes simples

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/04_Media`

21.2.7 Date

Today	Permet de récupérer la date du jour

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/11_Date`

21.2.8 Sécurité

Basic	Décode ou encode l'autorisation
Digest	Décode ou encode l'autorisation
Hmac	Décode ou encode l'autorisation
Oauth	Décode ou encode l'autorisation
Wsse	Décode ou encode l'autorisation
Certificate	Décode les certificats dans un format lisible
JWT	Décode ou encode des tokens

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/01_Security`

21.2.9 Temps

Timestamp	Permet de générer un timestamp ou de convertir en valeur lisible

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/06_Time`

21.2.10 Unités

Bytes	Permet de convertir des bytes en valeur lisibles

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Libraries/08_Units`

21.3 Outils tiers

Le produit vient à la base avec un certain nombre de plugins pour s'interfacer avec d'autre d'outils existants (suivi de défauts, managements de tests, etc..).

Ces plugins peuvent être utilisés directement depuis un test.

Liste des outils supportés :

Git	Clone/commit de fichier sur un dépôt distant
Jira	Création de ticket
HP ALM QC	Exécution de test, création de ticket. Version 12 minimum
ExtensiveAutomation	Exécution de test, création de variable
Jenkins	Exécution de tests avant ou après un build
VSphere	Création ou supression de machine virtuelle sur VMware

Note :

La solution dispose d'une API REST, elle peut être pilotée aussi par ces outils.

— Plugin `Jenkins` : https://wiki.jenkins.io/display/JENKINS/ExtensiveTesting+Plugin

21.3.1 HP ALM

Ce plugin permet d'exporter des résultats de tests dans l'outil HP ALM. Il peut etre utilisé depuis un etst pour exporter des résultats sans intervention utilisateur.

Exemple d'utilisation :

```
HP ALM ------> Appel REST API -----> ET
^                                    |
|                                    v
|                              Exécution du test demandé
|                                    v
+<-------- Push du résultat ----------+
```

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/02_HP_QC`

21.3.2 Jenkins

Ce plugin permet de lancer un build depuis la solution Extensive.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/06_Jenkins`

21.3.3 VSphere

Ce plugin permet de piloter un environnement virtuel VMware. Il peut être utilisé pour :

— créer des machines virtuelles en mode automatiquement
— supprimer des machines

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/05_VSphere`

21.3.4 ExtensiveTesting

Ce plugin permet de faire un lien entre plusieurs environnement (dev, intégration, qualification) en permettant d'exécuter des tests d'un environnement à l'autre.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/03_ExtensiveTesting`

21.3.5 Jira

Ce plugin permet de créer des tickets suite à l'exécution d'un test dans l'outil Jira.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/01_Jira`

21.3.6 Git

Ce plugin permet de récupérer ou pousser des fichiers depuis un dépôt de sources. Il peut être utilisé en prérequis d'un test.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Interop/04_Git`

21.4 Agents

Les agents sont disponibles depuis la boîte à outils. Il sont à utiliser conjointement avec les adaptateurs

— pour communiquer avec le système à tester ou piloter lorsque qu'il n'est pas accessible en direct par le serveur de test (ex : une page web)

— exécuter un test sur plusieurs environnements différents.

Note : L'agent `dummy` est à utiliser comme base pour le développement d'un nouvel agent.

21.4.1 Protocoles réseaux

socket	Permet de démarrer des sockets TCP/UDP
ftp	Permet de se connecter sur un serveur FTP(S)
database	Permet de requêter les bases de données (MySQL, Microsoft SQL et PostgreSQL)
ssh	Permet de se connecter sur des machines via SSH ou SFTP

21.4.2 Systèmes

command	Permet d'exécuter des commandes systèmes sur Windows ou Linux
file	Permet de récupérer des fichiers sur les systèmes Windows ou Linux

21.4.3 Outils tiers

sikulix-server	Intéractions avec les applications lourdes
selenium3-server	Permet de piloter les navigateurs web dernières générations
selenium2-server	Permet de piloter les navigateurs web
soapui	Permet d'exécuter des tests SoapUI
adb	Permet de piloter les smartphones Android
gateway-sms	Permet d'envoyer ou recevoir des SMS

Note : L'utilisation de l'agent `Selenium3-Server` nécessiste au minimum d'avoir `Java 8` sur le poste.

CHAPITRE 22

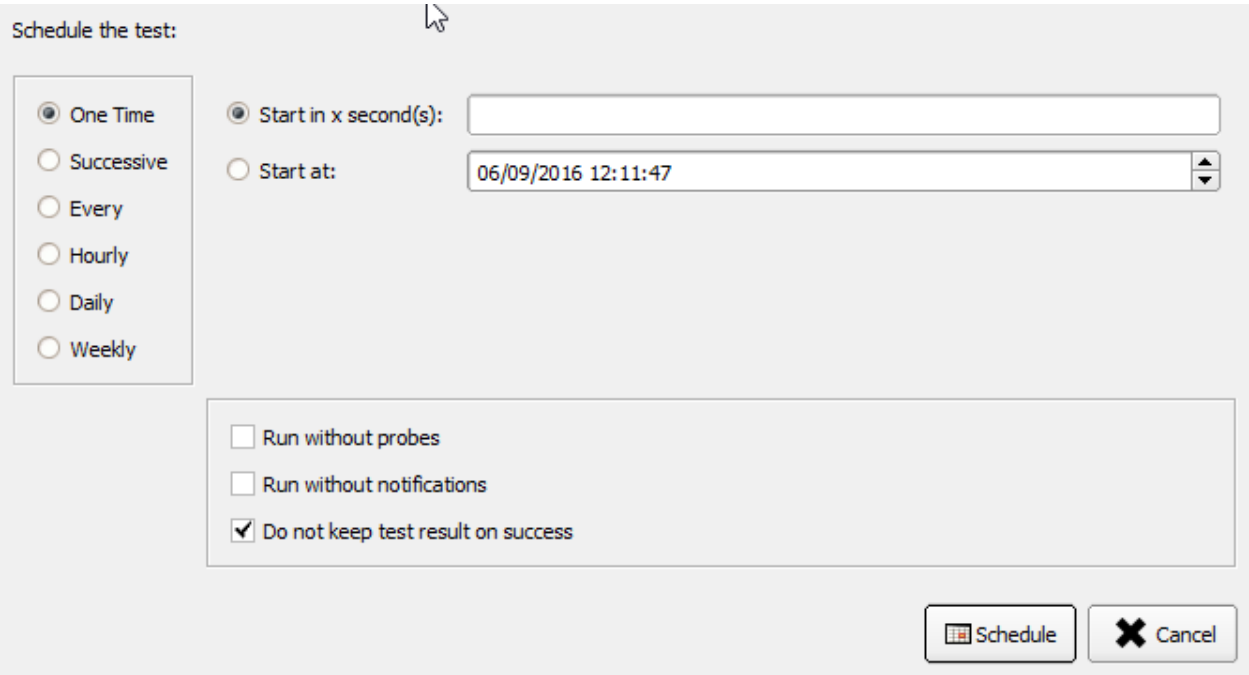
Le moteur d'exécutions

22.1 L'ordonnanceur

22.1.1 Programmation des exécutions

L'ordonnanceur présent dans le serveur permet de programmer l'exécution des tests de plusieurs manières.

- Exécuter le test une seule fois dans `x_secondes` ou à `date_heure`
- Exécuter le test plusieurs fois à `date_heure`.
- Exécuter le test à chaque `interval` de `heure_début` à `heure_fin`
- Exécuter le test toutes les heures à une `heure` précise
- Exécuter le test tous les jours à une `heure` précise
- Exécuter le test une fois par semaine le `jour de la semaine` à une `heure` précise



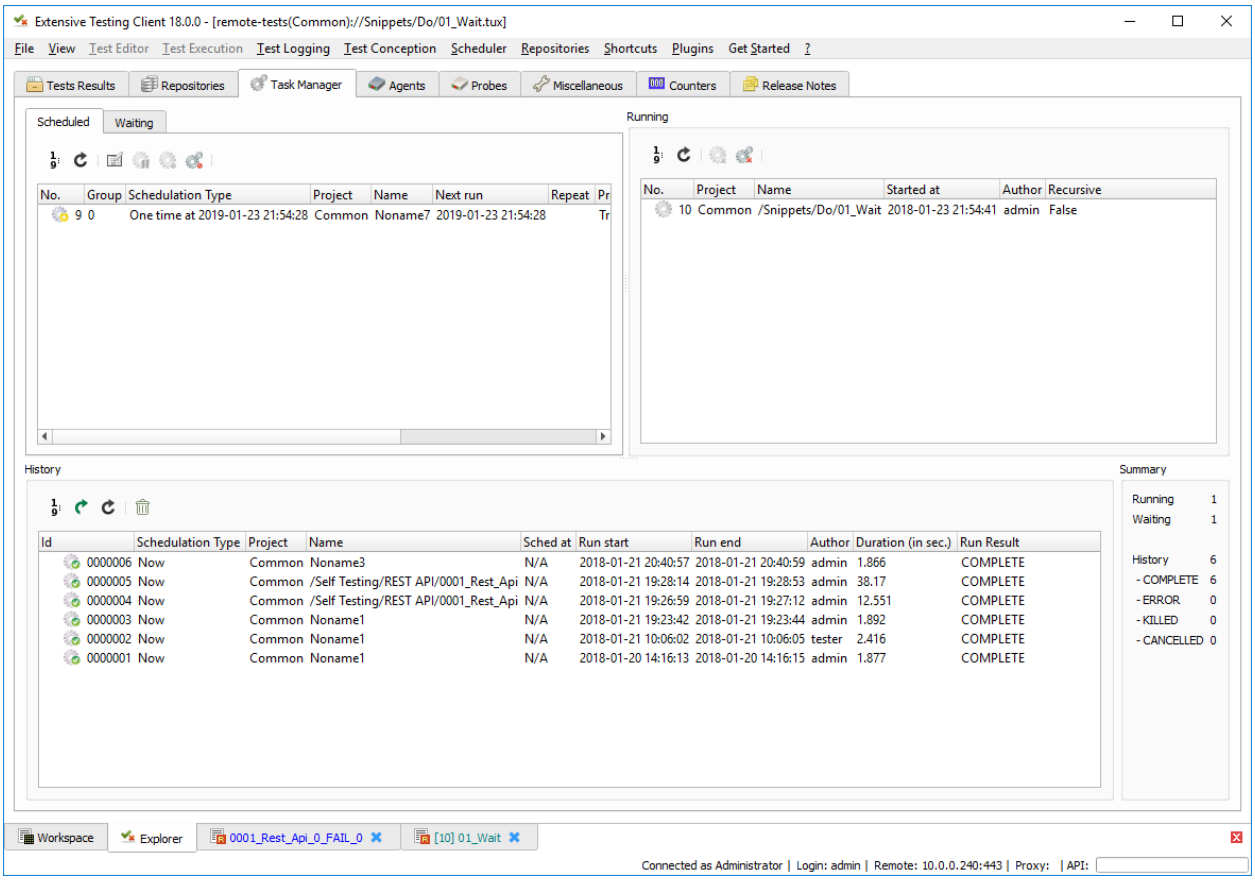
Note : Une tâche récursive sera automatiquement relancée par le serveur après un redémarrage.

22.1.2 La gestion des tâches

Les actions suivantes sont disponibles pour gérer les tâches programmées par les utilisateurs :

- annuler une ou plusieurs tâches
- forcer l'arrêt d'une ou plusieurs tâches
- reprogrammer une ou plusieurs tâches
- visualiser l'historique des exécutions.

L'ensemble de ses actions est réalisable depuis le client lourd ou bien depuis l'API.

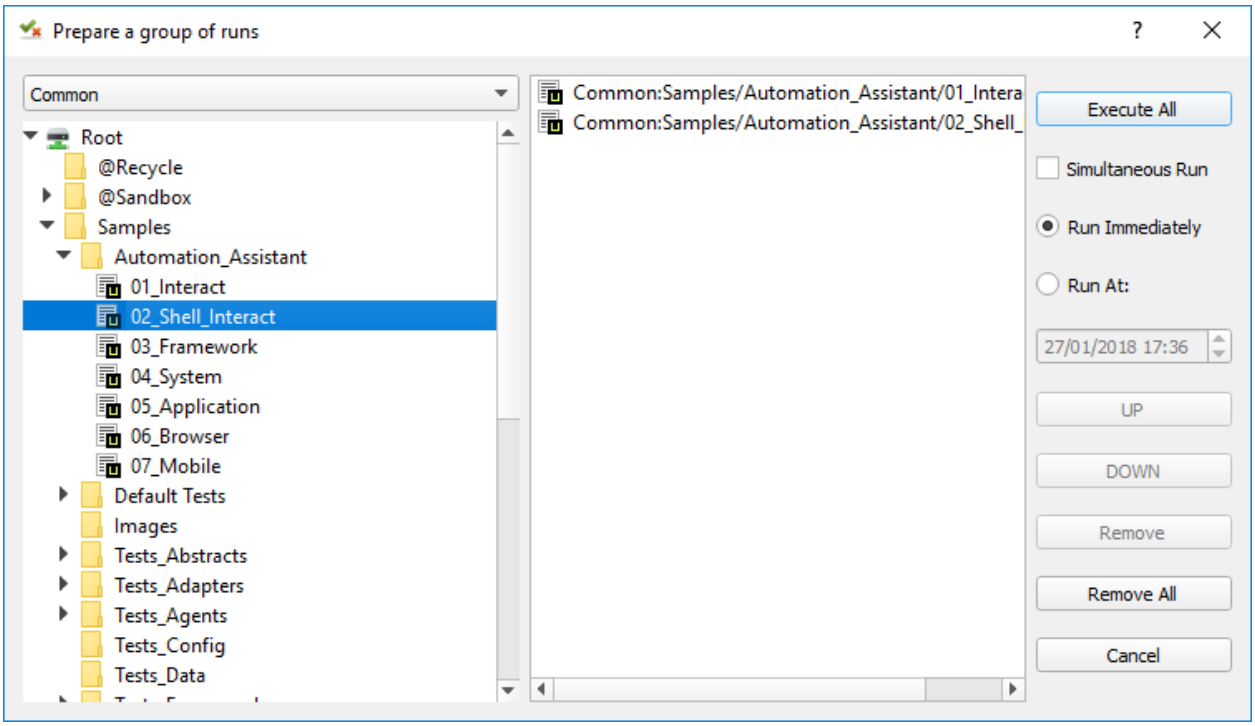


22.2 Exécutions parallélisées

Il est possible d'exécuter plusieurs tests en parallèles en utilisant la fonction `Grouped` Cette fonction est disponible à partir du client lourd ou bien depuis l'API.

Il existe 2 options d'exécutions :

— exécution des tests l'un après l'autre (sans lien)
— ou exécution en parallèle



Note : Depuis l'API, il faut utiliser la fonction `/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 : Il n'y a aucune garantie que les tests vont démarrer en même temps avec ce mode d'exécution.

22.3 Exécutions synchronisées

22.3.1 Partage des adaptateurs

La fonction `mode partagé` permet de réutiliser le même adaptateur dans plusieurs cas de test. Ce mode est à utiliser dans un scénario (test plan) ou un test suite avec plusieurs cas de tests.

Voici un exemple d'utilisation possible :

- le scénario teste une application
- en arrière plan le scénario vérifie aussi les logs générés par l'application
- Il est donc possible d'influer sur le résultat du test en fonction de ce qui est trouvé dans les logs.

Pour activer le mode partagé, il faut mettre à True le paramètre `shared` et donner un nom à l'adaptateur :

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

Note : Il est important de donner un nom à son adaptateur car ça permet de le retrouver plus facilement. Si aucun nom n'est donné, le framework configure l'adaptateur avec un nom aléatoire.

Après initilisation de l'adaptateur il est possible de récupérer un adaptateur depuis un autre cas de test en le recherchant par son nom.

```
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 Partage de donnée

Le cache étant unique lorsqu'un test (peu importe le type) est exécuté, il est possible d'échanger des données entre plusieurs cas de test.

Un premier test peut enregistrer une donnée dans le cache et un 2ième test peut récupérer la valeur stockée par le 1er test.

22.3.3 Synchronisation

Une exécution synchronisée de plusieurs cas de test est possible en utilisant un scénario (testplan). Ce scénario doit contenir :

— un cas de test observateur
— un ou plusieurs cas de tests exécutant des actions en arrière plan

Le test observateur doit être utilisé pour faire le lien entre les différents adaptateurs.

Important : L'utilisation d'adaptateurs en mode partagé est obligatoire.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Non_Sequential`.

22.4 Exécutions distribuées

La solution permet de faire des exécutions distribuées en utilisant des agents répartis à travers le réseaux.

CHAPITRE 23

Exemples avancés

23.1 Adaptateur SSH

L'adaptateur `SSH` permet de se connecter sur des serveurs distants en utilisant le protocole SSH.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le port du serveur distant (par défaut 22)
- le compte utilisateur

L'adaptateur supporte les fonctionnalités suivantes :

- authentification par nom d'utilisateur et mot de passe
- authentification par échange de clé

Exemple de configuration de l'adaptateur dans la section `prepare` du 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')
                                    )
```

Exemple pour se connecter, s'authentifier sur un serveur distant et se déconnecter :

```
connected = self.ADP_SSH.doConnect(
                                    timeout=input('TIMEOUT'),
                                    prompt='~]#'
                                  )
if not connected: self.abort("ssh connect failed")
self.info("SSH connection OK" )
```

(suite sur la page suivante)

(suite de la page précédente)

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

Exemple pour envoyer une commande sur une machine distante :

```
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 )
```

Avertissement : Les réponses SSH peuvent être découpées en plusieurs évènements (celà dépend du réseau). Il faut donc faire attention quand on attend une réponse spécifique, l'utilisation d'un buffer peut être nécessaire dans ce cas là.

Note : Des exemples sont disponibles dans l'échantillon `/Samples/Tests_Adapters/05_SSH.tsx`.

23.2 Adaptateur HTTP

L'adaptateur `HTTP` permet d'envoyer des requêtes et d'inspecter les réponses associés vers un serveur Web.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le port du serveur distant (par défaut 80)

L'adaptateur supporte les fonctionnalités suivantes :

- le chiffrement `tls` de la communication
- l'utilisation de proxy `socks4, 5` et http
- l'authentification `digest` ou `basic`
- le réassemblage des réponses `chunked`

Exemple de configuration de l'adaptateur dans la section `prepare` du 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'),
                                        agentSupport=input('SUPPORT_AGENT')
                                    )
```

Exemple pour envoyer une réquête de type `GET` et d'une réponse avec le code `200`.

```
rsp = self.ADP_HTTP.GET(
                            uri="/",
```

(suite sur la page suivante)

(suite de la page précédente)

```
                        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")
```

Exemple pour envoyer une réquête de type `GET` et attendre une réponse répondant aux critères suivants :

- la version doit se terminer par 1.1
- le code ne doit pas contenir la valeur 200
- la phrase ne doit pas contenir le texte *Testing*
- le corps de la réponse doit contenir le texte *google*
- la réponse doit contenir une entête contenant le texte *server*, peut importe la valeur

```
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 Adaptateur Telnet

L'adaptateur `Telnet` permet de se connecter sur des machines disposant une interface telnet.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le port du serveur distant (par défaut 23)

Exemple de configuration de l'adaptateur dans la section `prepare` du test.

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

Exemple pour se connecter ou se déconnecter du serveur distant

```
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' )
```

Exemple montrant comment attendre la réception d'un texte en particulier.

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

Exemple pour envoyer des données au serveur distant

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

recherche un texte en particulier. Pour se prémunir de ce problème, il faut ajouter un buffer intermédiare, il y a un exemple complet avec l'adaptateur `Catalyst`.

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Adapters/12_Telnet.tsx`.

23.4 Adaptateur MySQL

L'adaptateur `MySQL` permet de se connecter sur une base donnée distante.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le port du serveur distant (par défaut 3306)
- le nom d'utilisateur
- le mot de passe associé

Exemple de configuration de l'adaptateur dans la section `prepare` du test.

```
self.ADP_MYSQL = SutAdapters.Database.MySQL(
                                      parent=self,
                                      host=input('HOST_DST'),
                                      user=input('MYSQL_LOGIN'),
                                      password=input('MYSQL_PWD'),
                                      debug=input('DEBUG'),
                                      verbose=input('VERBOSE'),
                                      agent=input('AGENT_DB'),
                                      agentSupport=input('SUPPORT_AGENT')
                                      )
```

Exemple pour se connecter ou se déconnecter du serveur distant :

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

self.ADP_MYSQL.disconnect()
```

Exemple pour exécuter une requête SQL dans la base de donnée :

```
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 : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Adapters/15_Database.tsx`.

23.5 Adaptateur SNMP

L'adaptateur SNMP permet de recevoir des alarmes SNMP v1 ou v2.

La configuration de l'adaptateur consiste à indiquer à minima :

— l'adresse d'écoute
— le port d'écoute

Exemple de configuration de l'adaptateur dans la section prepare du 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')
                                                )
```

Exemple pour démarrer l'écoute du serveur

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

Exemple pour attendre la réception d'une alarme :

```
trap = self.UDP_ADP.hasReceivedTrap(
                                      timeout=input('TIMEOUT'),
                                      version=SutAdapters.SNMP.TRAP_V1,
                                      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 : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Adapters/18_SNMP.tsx`.

23.6 Adaptateur FTP(s)

L'adaptateur FTP permet de se connecter sur des serveurs distants et supporte les fonctions suivantes :

- Connection en TLS
- Téléchargement ou récupation de fichiers ou répertoires
- Ajout/suppression et renommage de fichiers ou répertoires
- Lister le contenu d'un répertoires
- Détecter l'apparition d'un fichier ou répertoire avec le support des expressions régulières.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le nom d'utilisateur pour se connecter
- le mot de passe

Exemple de configuration de l'adaptateur dans la section prepare du 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')
                                      )
```

Exemple pour se connecter ou déconnecter du serveur FTP :

```
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:
    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" )
```

Exemple pour lister le contenu d'un répertoire :

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

Exemple pour détecter un fichier dans un répertoire avec une expression régulière :

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

(suite sur la page suivante)

(suite de la page précédente)

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

Note : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Adapters/21_Ftp.tsx`.

23.7 Adaptateur SFTP

L'adaptateur SFTP permet de se connecter sur des serveurs disposants d'une interface SSH. Les fonctionnalités suivantes sont supportées :

- Téléchargement ou récupation de fichiers ou répertoires
- Ajout/suppression et renommage de fichiers ou répertoires
- Lister le contenu d'un répertoires
- Détecter l'apparition d'un fichier ou répertoire avec le support des expressions régulières.

La configuration de l'adaptateur consiste à indiquer à minima :

- l'adresse ip du serveur distant
- le nom d'utilisateur pour se connecter
- le mot de passe

Exemple de configuration de l'adaptateur dans la section `prepare` du 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')
                                        )
```

Exemple pour se connecter et déconnecter du serveur :

```
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" )
```

Exemple pour lister le contenu d'un répertoire :

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

(suite sur la page suivante)

(suite de la page précédente)

```
                        )


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") )
```

Exemple pour détecter un fichier dans un répertoire avec une expression régulière :

```
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 : Un exemple est disponible dans les échantillons de tests `/Samples/Tests_Adapters/22_Sftp.tsx`.

23.8 Librairie ChartJS

L'adaptateur `ChartJs`, basé sur la librairie javascript du même nom, permet de générer des graphiques pouvant être intégré dans une page html. L'intérêt principal de cette librairie est de pouvoir intégrer des graphiques dans le rapport de test.

Exemple de configuration de la librairie dans la section `prepare` du test.

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

Exemple pour générer un graphique de type barre et l'intégrer dans le rapport

```
# 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 grahique
myChart = self.LIB_CHART.barChart(
                                    labelsAxes=labelsAxes,
                                    datas=[dataA, dataB],
                                    legendDatas=legendDatas,
                                    width=400,
                                    height=300,
```

(suite sur la page suivante)

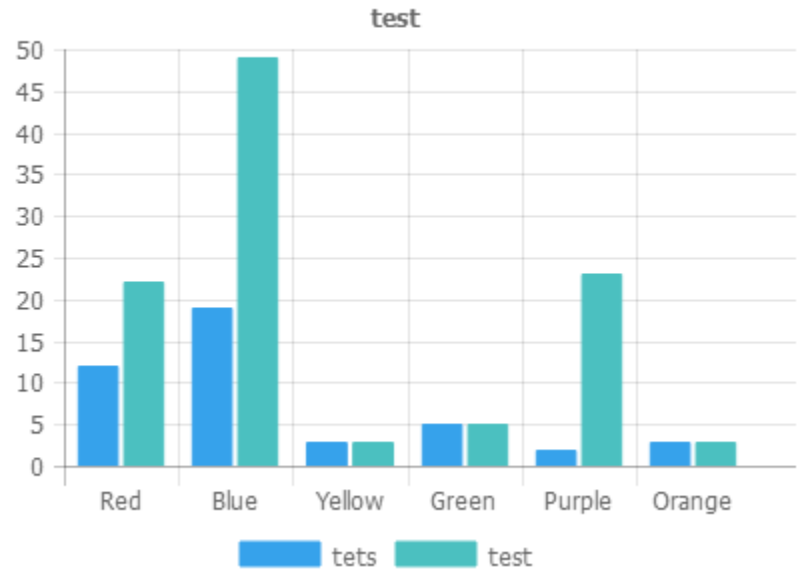
(suite de la page précédente)

```
                                backgroundColors=[borderColor, backgroundColor],
                                borderColors=[borderColor, backgroundColor],
                                chartTitle="test"
                            )

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

Le graphique est inséré automatiquement dans le rapport avancé.

1. PASS - chart



23.9 Paramètre de tests « text »

Le paramètre de type `text` permet de construire des valeurs appelant d'autres variables.

Prenons l'exemple d'un test contenant les 2 variables suivantes :

— DEST_IP avec la valeur 192.168.1.1
— DEST_PORT avec la valeur 8080

2	**DEST_IP**	*str*	192.168.1.1
3	**DEST_PORT**	*int*	8080
4	**DEST_URL**	*custom*	

Le type `text` va nous permettre de construire une 3ième variable

— DEST_URL avec la valeur

Test Config > Custom Values

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

Le mot clé `[!INPUT:<NOM_VARIABLE_ENTRANTE:]` permet d'appeler une autre variable entrante. Le framework remplacera au moment de l'exécution du test les différents mots clés avec la valeur associée. On obtiendra comme valeur https://192.168.1.1:8080/welcome pour la 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

Pour aller plus loin, il est aussi possible d'ajouter une valeur disponible depuis le cache. Partant du principe que la valeur « welcome?user=hello » est dans le cache et accessible via la clé « url_params ». Il est possible de l'intégration dans le paramètre comme ci-dessous

Test Config > Custom Values

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

Exemple de résultat après exécution :

	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 Paramètre de tests « json »

todo

23.11 Paramètre de tests « alias »

Le paramètre de type `alias` peut être utilisé pour définir un nouveau nom pour un paramètre déjà existant. Ce mécanisme peut être utilisé dans les `test plan` pour éviter de surcharger tout les paramètres ayant le même nom.

Exemple d'utilisation

1. Avant exécution

```
Scénario (TIMEOUT_A(int)=2 secondes)
 ---> Test 1 (TIMEOUT_A(int)=10 secondes)
 ---> Test 2 (TIMEOUT_A(int)=30 secondes)
 ---> Test 3 (TIMEOUT_A(int)=20 secondes)
```

2. Après exécution du test

```
Scénario (TIMEOUT_A(int)=2 secondes)
  ---> Test 1 (TIMEOUT_A(int)=2 secondes)
  ---> Test 2 (TIMEOUT_A(int)=2 secondes)
  ---> Test 3 (TIMEOUT_A(int)=2 secondes)
```

Quand on exécute le scénario ci-dessus, le test 1, 2 et 3 ont automatiquement la valeur 2 secondes pour le paramètre TIMEOUT_A. C'est le comportement apporté par le framework de test.

Comment faire si on souhaite que le test 2 garde la valeur 30 secondes par contre le test 1 et 2 hérite de la valeur du scénario ?

Il faut utiliser un paramètre de type `alias`, ils ne sont pas surchargés par le framework.

1. Avant exécution

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

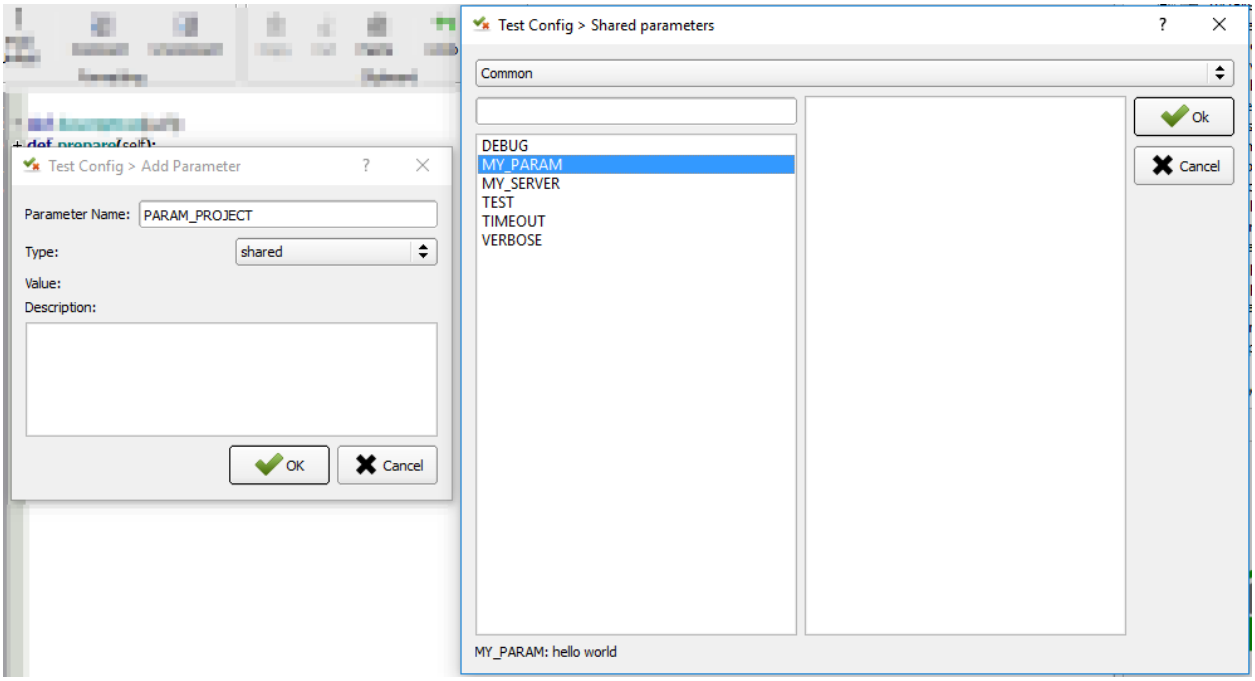
2. Après exécution du test

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

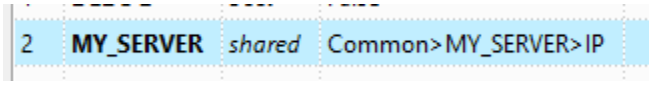
23.12 Paramètre de tests « global »

Les paramètres de type `global` s'ajoutent depuis l'interface web ou depuis l'api REST. Ils sont partagés et accessibles par l'ensemble des tests d'un même projet. La valeur attendue pour ce paramètre est de type `JSON`.

Une fenêtre de sélection dans le client graphique permet de sélectionner le paramètre à utiliser dans le test.



Dans l'exemple ci-dessous, le paramètre de test `MY_SERVER` contient la valeur de la clé `IP` présente dans la variable partagée `MY_SERVER` qui est elle-même présente dans le projet `Common`.



Astuce : Pour avoir un paramètre de test qui contient une liste d'éléments, il faut utiliser le type `list-global`.

23.13 Paramètre de tests « dataset »

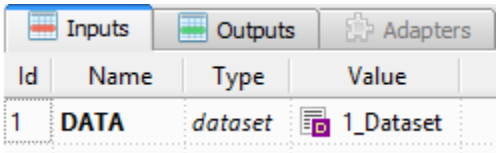
Le paramètre de type `dataset` permet d'importer des fichiers `tdx`. Un fichier `dataset` est juste un fichier texte, il est possible de le créer à partir du client graphique et de le sauvegarder dans le dépôt des tests distants.



Exemple de contenu d'un fichier dataset avec le format csv

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

Ce fichier peut être utilisé dans un test l'important dans les paramètres.



Exemple pour lire la variable :

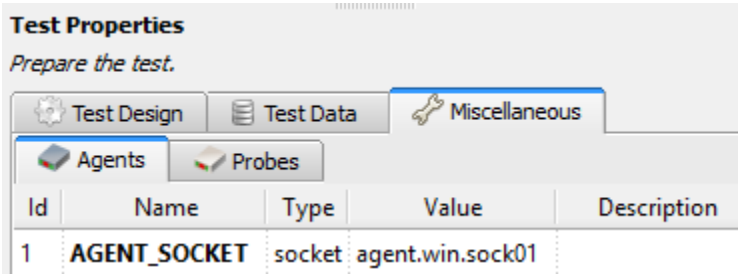
```
for d in input('DATA').splitlines():
    Trace(self).info( d )
```

23.14 Utilisation d'un agent

Pour utiliser un agent, il faut deux choses :

- Déployer la boite à outils et sélectionner l'agent souhaité.
- Déclarer l'agent dans le test
- Configurer l'adaptateur pour utiliser l'agent.

Les agents sont à déclarer depuis le client dans l'onglet `Miscellaneous > Agents`

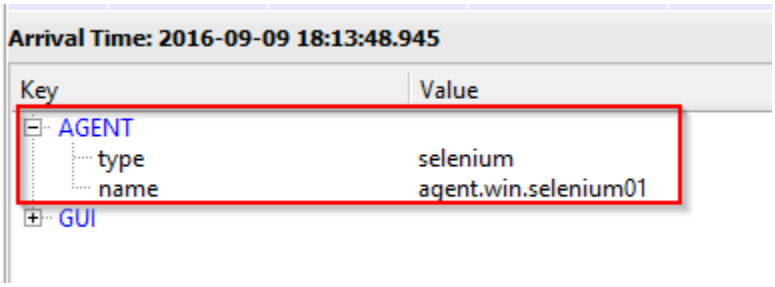


L'activation du mode agent sur les adaptateurs se fait avec les arguments `agentSupport` et `agent`.

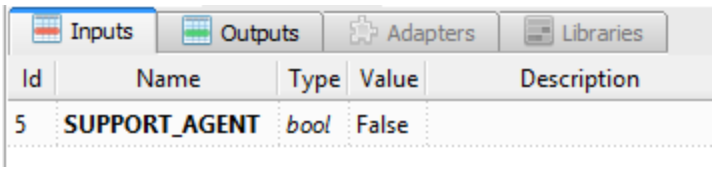
```
agentSupport=input('SUPPORT_AGENT'),
agent=input('AGENT_SOCKET')
```

```
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')
                                       )
```

Dans la fenêtre d'analyse, il est possible de voir l'agent utilisé pour chaque évènement :



Note : Il est conseillé de mettre en paramètre de test l'usage du mode agent.


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

CHAPITRE 24

Prérequis systèmes

24.1 Serveur

Pour l'instant le serveur ne peut être exécuté que sur un environnement Linux. Il est possible de l'exécuter sur un environnement virtuel ou physique.

Caractéristiques	Minimum	Recommandé
OS	Linux	
Python	2.7	
Arch	x86_64	
Mémoire	4Go	8 Go
CPU	2 Coeurs	4 coeurs
Disque	~10Go	~50Go
Réseau	100Mb/s	1 Gbit/s
Nombre Utilisateur	2	5

Note : Le support de Python 3.x est cours de développement.

24.2 Client

Le client peut être exécuté sur un environnement Windows ou Linux.

Caractéristiques	Minimum	Recommandé
OS	Windows 7+ Linux	Windows 10+ Linux
Arch	x86_64	
Mémoire	4Go	8Go
Disque	~1Go	~2Go

Note : L'architecture 32-bit n'est plus supportée depuis la version 17.0.0. Cependant il est toujours possible de compiler les sources sur un environnement 32bits.

Important : Les plugins pour le client ne sont disponibles que pour l'environnement Windows.

24.3 Boite à outils

La boite à outils peut être exécutée sur un environnement Windows ou Linux.

Caractéristiques	Minimum	Recommandé
OS	Windows 7+ Linux	Windows 10+ Linux
Arch	x86_64	
Mémoire	4Go	8Go
Disque	~1Go	

Note : L'architecture 32-bit n'est plus supportée depuis la version 17.0.0. Cependant il est toujours possible de compiler les sources sur un environnement 32bits.

Important : Les plugins pour le client ne sont disponibles que pour l'environnement Windows.

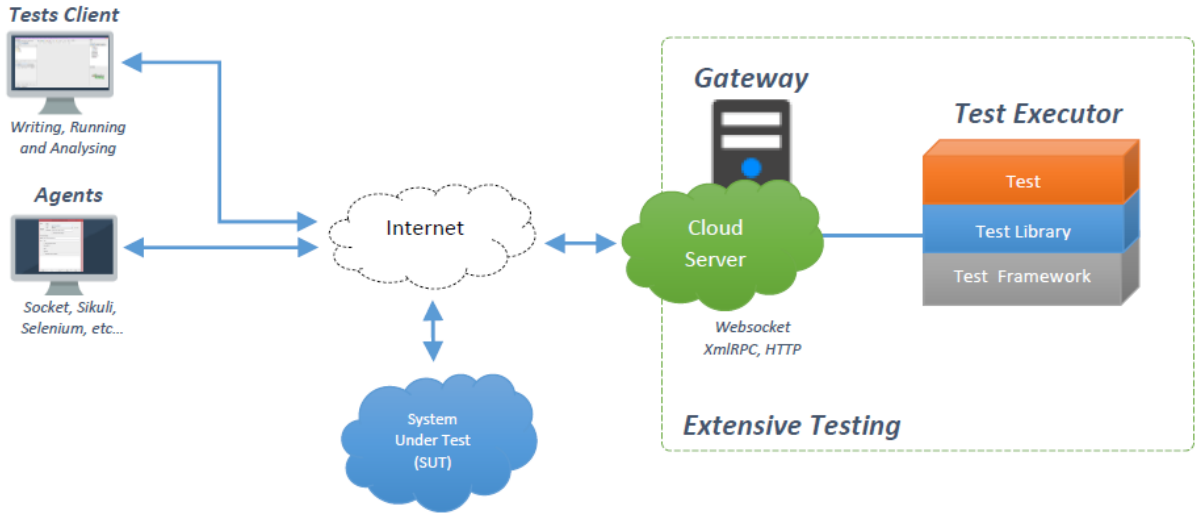
CHAPITRE 25

Architecture

La solution est basée sur un mode client/serveur. Les tests et adaptateurs sont centralisés dans un serveur qui permet de fournir rapidement le même environnement de test à l'ensemble des utilisateurs.

La solution se compose de plusieurs composants :

- Un serveur
- Un client graphique
- Des agents



25.1 Serveur

Le serveur se compose :

- d'un reverse proxy (apache)
- d'un ordonnanceur

— d'une serveur API REST
— du framework de test
— des adaptateurs
— des extensions outils
— d'une interface web

25.2 Client Graphique

Le client graphique utilise un seul flux tcp/8080 (https) entre le client et le serveur. Le flux est bidirectionnel et le client peut :

— effectuer des appels vers l'API REST du serveur
— recevoir des évènements du serveur via des WebSockets.

25.3 Agents

Un agent est obligatoirement contrôlé par un adaptateur via l'intermédiare du serveur de test. Il permet de déporter le point de communication avec le système à tester ou piloter. Les agents utilisent un seul flux tcp/8080 (https) pour communiquer avec le serveur.

CHAPITRE 26

Spécifications

26.1 Cycle des versions

L'ensemble des paquets logiciels de la solution respecte les rêgles suivantes pour le nommage des versions.

La version se découpe en 3 chiffres (A.B.C)

- **A : le 1er chiffre indique la version majeure. L'incrémentation de ce chiffre implique**
  - l'ajout de fonctionnalités majeures (avec pottentiellement une perte de compatibilité avec la version précédente)
  - l'ajout de fonctionnalités mineures
  - la correction de bug
- **B : Le 2ième chiffre indique une version mineure. L'incrémentation de ce chiffre indique**
  - l'ajout de fonctionnalités mineures
  - la correction de bug
- **C : le 3ième chiffre indique une version de maintenance. L'incrémentation de ce chiffre indique**
  - la correction de bug

26.2 Arborescence du serveur

L'ensemble des fichiers manipulés par le serveur sont stockés dans le répertoire `[...]/Var/`.

```
Build/
ServerEngine/
ServerControls/
ServerInterfaces/
ServerRepositories/
Libs/
TestCreatorLib
TestExecutorLib/
TestInterop/
```

(suite sur la page suivante)

(suite de la page précédente)

```
Var/
  SutAdapters/
  Tests/
  TestsResults/
  Logs/
  Backups/
```

Les tests sont stockés dans le répertoire `[...]/Var/Tests/`, ils sont organisés par identifiant de projet.

26.3 Modèle de données

Une base de donnée est utilisée par le serveur pour stocker :

- les utilisateurs de la solution
- la liste des projets
- les données de tests (variables projets)
- des statistiques
- l'historique des exécutions

Tables	Description
xtc-config	Configuration du serveur
xtc-projects	Liste des projets
xtc-relations-projects	Relation entre les projets et les utilisateurs
xtc-users	Liste des utilisateurs
xtc-test-environment	Liste des variables au format JSON
xtc-tasks-history	Historique des tâches exécutées sur le serveur

26.4 Gestion des mots de passes

Aucun mot de passe (en clair) est stocké dans la base de donnée. L'utilisation d'un hash est par contre utilisé. Le hash du mot de passe est stocké dans la table *xtc-users*.

L'algorithme utilisé :

```
MariaDB [xtc112]> select id, login, password from 'xtc-users';
+----+----------+------------------------------------+
| id | login    | password                           |
+----+----------+------------------------------------+
|  1 | system   | 146aeec808258c699abf36fb9cb81697833a946f |
|  2 | admin    | c9188bd8805ea6c0018469b5bf106fb38c3cb9f88 |
|  3 | leader   | c9188bd8805ea6c0018469b5bf106fb38c3cb9f88 |
|  4 | developer | c9188bd8805ea6c0018469b5bf106fb38c3cb9f88 |
|  5 | tester   | c9188bd8805ea6c0018469b5bf106fb38c3cb9f88 |
|  6 | automaton | c9188bd8805ea6c0018469b5bf106fb38c3cb9f88 |
+----+----------+------------------------------------+
```

26.5 Format des fichiers

Les tests sont au format `XML`. Il existe plusieurs formats de tests :

- Test Unit Xml
- Test Suite Xml

— Test Plan Xml
— Test Global Xml

Structure XML partagée

```
<?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>
    <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 Stockage des résultats de tests

Les résultats de tests sont stockés sur le serveur dans le répertoire `/opt/xtc/current/Var/TestsResult`.

Les résultats sont stockés :

- par l'id des projets de test
- par la date du jour d'exécution du test
- et finalement par la date et heure d'exécutions des tests.

Organisation des résultats :

```
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 des fichiers :

- `TESTPATH` contient le chemin d'accès complet pour le résultat de test
- `test.out` contient les logs interne du test, à utiliser pour débugger le framework de test
- `test.ini` contient des paramètres spécifiques au test
- `<testname>_<replayid>.hdr` réprésente l'entête du résultat de test
- `<testname>_<replayid>_<result>_<nbcomments>.trv` contient l'ensemble des évènements générés pendant l'exécution du tests
- `<testname>_<replayid>.tbrp` contient le rapport basique au format html
- `<testname>_<replayid>.trp` contient le rapport complet au format html

— `<testname>_<replayid>.trv` contient le rapport des résultats au format csv

26.7 Contrôle Agents

Le pilotage des agents depuis un test s'effectue à travers :
- les adaptateurs
- et le serveur

La communication s'effectue avec l'échange de quelques messages spécifiques :
- `init` : permet d'initialiser un agent
- `notify` : permet d'envoyer un message à l'agent sans attendre de réponse
- `reset` : permet de faire un reset de l'agent
- `error` : permet à l'agent d'envoyer une erreur à l'adaptateur
- `data` : permet à l'agent d'envoyer des données à l'adaptateur

Sens de communications disponibles :
- Agent -> serveur -> adaptateur -> test
- Test -> adaptateur -> serveur -> agent

	Agent	
	Fonction	Callback
Envoie d'un message « error »	**def sendError** — request — data	
Envoie d'un message « notify »	**def sendNotify** — request — data	
Envoie d'un message « data »	**def sendData** — request — data	
Réception d'un message « init »		**def onAgentInit** — client — tid — request
Réception d'un message « reset »		**def onAgentNotify** — client — tid — request
Réception d'un message « notify »		**def onAgentReset** — client — tid — request

	Adaptateur	
	Fonction	Callback
Réception d'un message « error »		**def receivedErrorFromAgent** — data
Réception d'un message « notify »		**def receivedNotifyFromAgent** — data
Réception d'un message « data »		**def receivedDataFromAgent** — data
Envoie d'un message « init »	**def initAgent** — data	
Envoie d'un message « reset »	def resetAgent	
Envoie d'un message « notify »	**def sendNotifyToAgent** — data	

26.8 Les logs serveurs

Les logs du serveur sont localisés dans le répertoire `[...]/Var/logs/`.

output.log	logs serveurs

CHAPITRE 27

Contributions

27.1 Développement client lourd

27.1.1 Qt client application

Regarder le README

27.1.2 Boite à outils

Regarder le README

27.1.3 Serveur

Regarder le README

27.2 Plugins côté serveur

Regarder le README

27.3 Plugins côté client et agents

Regarder le README

27.4 Documentations

La documentation est stockée sur github dans le dépôt. Il est possible de contribuer en faisant une demande de participation au dépôt.

La documentation est générée par le service readthedocs.

CHAPITRE 28

API

28.1 Authentification

Il y a 2 méthodes pour s'authentifier sur l'API REST :
- En utilisant un cookie de session
- En réalisant une authentification basique

28.1.1 Basique

L'authentification basique nécessite d'utiliser une clé API disponible depuis l'interface web. La re-génération de la clé se fait pour l'instant en ligne de commande sur le serveur.

```
./extensiveautomation --apikey admin
  API Key ID: admin
  API Key Secret: d30278d49e4845e45daa748873e2171b14a0c55a
```

Il faut ensuite ajouter l'header `Authorization` dans l'ensemble des requêtes avec la clé et le secret encodé en base 64

```
Authorization: Basic base64(key_id:key_secret)
```

Note : Avec l'authentification basique, il n'est pas nécessaire d'appeler la fonction login.

28.1.2 Cookie de session

L'authentification par cookie nécessite d'appeler en prérequis la fonction `login` pour générer un cookie de session. Ce cookie doit être ensuite fournit à l'ensemble des requêtes dans l'entête `Cookie`.

```
Cookie: session_id=NjQyOTVmOWNlMDgyNGQ2MjlkNzAzNDdjNTQ3ODU5MmU5M
```

28.2 Exemple d'utilisation

L'api est accessible à travers le `port 443` (https) du serveur de test via l'uri `/rest`.

L'exemple ci-dessous montre comment exécuter un test avec une authentification basique.

```
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} ]
}
```

La réponse reçue :

```
{
  "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 des fonctions les plus importantes :

Authentification

Note : La fonction `login` ne nécessite aucune authentification.

Exécution d'un test

Récupération des résultats