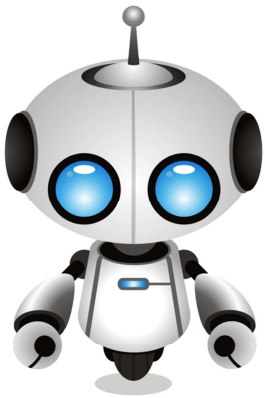


Industrialisation des tests automatisés

Documentation utilisateur

Contributions

Rédacteur	Date	Description
Fabrice Mougin	31/07/2020	Création du document
Fabrice Mougin	01/09/2020	Mise à jour avec les fonctionnalités d'août 2020
Fabrice Mougin	09/09/2020	Mise à jour options
Fabrice Mougin	14/10/2020	Mise à jour des tags
Marc Laverroux	15/10/2020	Relecture pour alimenter l'évolution
Fabrice Mougin	16/10/2020	Amélioration Getting started TSTFAC-174

Table des matières

Présentation du projet

La première étape de l'industrialisation consiste en une livraison d'une librairie robot framework (MaifTstFacLibrary) concentrant toutes les fonctionnalités du framework déjà capitalisées dans le template (http://gitlab-fabfonc.maif.local/tnr_produits/projet_reference_robotframework_new).

Il y a eu une ré-écriture de la partie socle du template en 100% python.

La livraison comprend :

- Une librairie MaifTstFacLibrary disponible sur le repository Nexus
- Une procédure de création d'un projet depuis le début 'Getting started'

L'ajout de fonctionnalités se fait dans le cadre d'un atelier organisé le lundi après-midi à 15h. Les étapes se déroulent de cette manière :

- Phase de cadrage
    - Atelier de présentation des besoins d'un ou plusieurs projets
    - Proposition par un projet ou par la stream outils
    - Tour de table, validation par consensus
- Phase de livraison
    - Implémentation d'une maquette
    - Atelier dédié à la présentation

Un autre moyen d'ajout d'une fonctionnalité à la librairie se fait par participation au développement encadré par la stream outils. Il y a déjà eu un contributeur pour le POC NeoLoad (Emmanuel Souris).

Présentation générale des fonctionnalités

- Choix de la cible d'exécution (locale, grid Selenoid, Darwin)
- Choix du navigateur, type et version
- Choix des comportement du profile du navigateur par défaut
    - Proxy
    - Cookies
    - Ouverture des nouvelles fenêtres dans les tab
    - Maximisation de la fenêtre de lancement
    - Navigation privée
- Re runs en cas d'erreur
- Règles RFLint pour guider les testeurs dans la rédaction (à venir)
- Archivage des logs dans un répertoire YYYY/MM/DD/ possible
- Activation des screenshots automatiques possible
- Choix du reporter (logs locaux par défaut, report portal en prod)
- Ajout de variables présentées dans le reporter
- Choix du parallélisme (robot ou pabot)
- Choix de variable pour l'intégration CI/CD ultérieure
- Choix de l'environnement, RECX, RECN, PPCOR
- Timeout de session Selenoid
- Choix de l'activation sur la GRID de :
    - VNC
    - Enregistrement vidéo
    - Enregistrement des logs
- Extensions
    - Connecteur NEOLoad automatique sans modification du script robot framework
    - Récupération des logs (pour vérification de messages Javascript de la console de firefox)
    - Possibilité de passer des variables pour faire muter le comportement des tests
        - Applicationversion pour la version de l'application testée, beta ou pas
        - 3 Variables Custom
    - Pages de statistiques sur l'état des tags des tests (à venir)
    - La Toolbox, un ensemble de keywords fréquemment utilisés par les projets

Toutes les fonctionnalités sont sélectionnables par priorité décroissante :

- Options passées au runner (surcharge tout)
- Variables d'environnement préfixées par TF_ pour passer à robot ou pybot dans le mode rétrocompatible
- Variables de launch pouvant être classées dans un fichier de configuration
- Variables de projet, le moins prioritaire, donnant le comportement de la fonctionnalité si elle n'a pas été déclarée de manière explicite par une des méthodes ci-dessus

Dans la suite du document, il sera fait référence aux fonctionnalités. Pour savoir comment les lancer, veuillez consulter l'aide en ligne de commande du runner, par l'option --help :

```
./Helper/run-test.py --help
```

et l'aide en ligne de commande des variables de launch :

```
./Helper/run-test.py --launchvariables help
```

et l'aide en ligne de commande des variables de fichiers de configuration complets :

```
./Helper/run-test.py --launchvariables example
```

Pré-requis à l'utilisation de la librairie

Un projet GitLab

Vous avez besoin d'un repository pour votre projet créé dans GitLab.

Voici les étapes pour le créer.

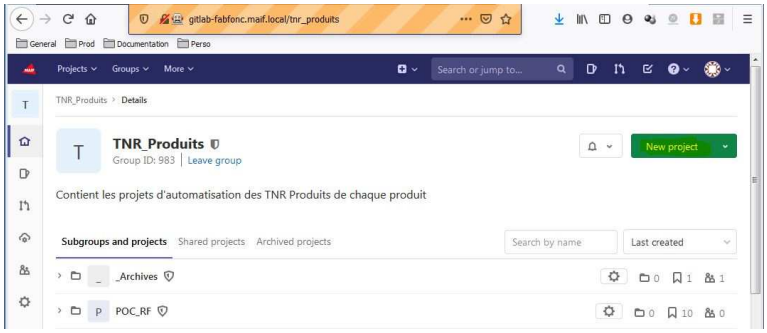
Si vous avez votre URL de projet gitlab, vous pouvez passer cette étape.

Attention, bien changer monprojet par votre nom de projet partout dans la suite de la documentation.

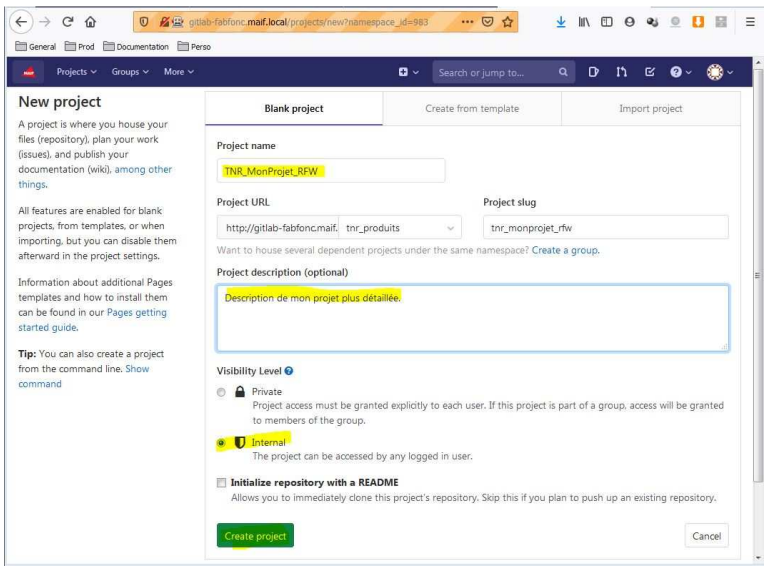
Dans Git dans la section TNR_Produits : http://gitlab-fabfonc.maif.local/tnr_produits

Créer votre projet avec cette nomenclature : TNR_<mon projet>_RFW.
Par exemple, si votre projet s'appelle «Maif test », il faut le modifier un peu pour enlever les espaces (remplacés par des _) et les accents pour avoir quelque chose comme cela : TNR_maif_test_RFW.

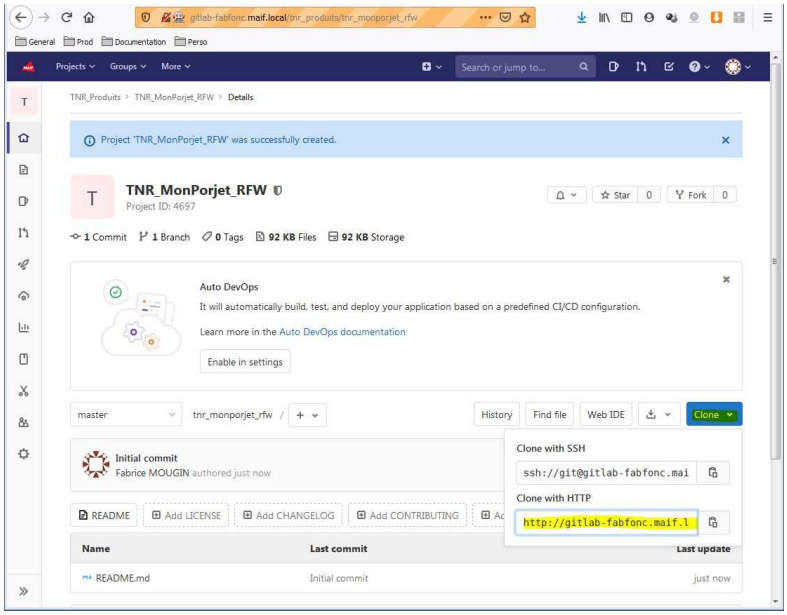
Pour cela, cliquez sur New Project :



Puis remplissez les champs surlignés en jaune, bien sélectionner un projet Interne :



Après avoir créé le projet, Ensuite, il faut aller chercher l'URL de clone de votre projet :



Ici vous avez votre URL GitLab de votre projet crée.

Un accès à report portal

Vous pouvez reprendre le nom de votre projet sans les espaces ni accents qui vous a permis de créer votre repository gitlab et faire une demande de création de projet au canal Teams RobotFramework avec ce template :

Canal Automatisation des tests – TCS > RobotFrameWork.
Le lien est :
https://teams.microsoft.com/l/channel/19%3af7f8176d944249ef94ba785df578b3db%40thread.skype/RobotFrameWork?groupId=8ed03012-7e92-438b-9920-bf8079d6e08d&tenantId=9c9d8823-ab9e-4ac4-8251-32c4a7ae50d5
Pour accélérer la demande, remplissez bien tous les champs et mettez bien le @RobotFrameWork pour vous adresser au canal entier.

Bonjour @RobotFrameWork,
J'ai besoin d'automatiser mes tests avec robotframework.
Pouvez-vous me donner un accès à http://reportportal.maif.local ?
Voici les informations de mon projet :

- Nom du projet : maif_test
- URL GitLab du projet : http://gitlab-fabfonc.maif.local/...
- Mon nom
- Mon prénom
- Mon adresse mail
- L'équipe ou service dans laquelle je travaille

On vous communiquera un accès ainsi qu'on nom de projet que vous pourrez utiliser pour configurer la librairie robot framework.

Un poste de travail

- Un poste de travail Windows 7 ou Windows 10
- python 3.7 ou plus installé
- accès au repository Nexus http://nexus-fabfonc.maif.local/repository/pip-hosted/
- outils Git installés (Git Bash et git en ligne de commande)
- visual studio installé avec un terminal Git Bash configuré (IDE par défaut)
- les extension suivantes installées dans visual studio code :
    - Python
    - Robotframework intellisense FORK
- Un espace de travail persistant sur le poste de travail.

Vérifiez bien les extensions et que lorsque vous ouvrez un terminal dans visual studicode vous avez bien bash affiché.

Getting started

Pré-requis :

- **L'URL GitLab de votre projet**, c.f. section précédente concerné à la création du projet dans GitLab
- **Un accès reportportal et un nom de projet sans espace ni accents** fourni, c.f. section précédente concerné à la demande d'accès ReportPortal.
- **Un poste de travail windows 7 ou windows 10 suivant les prérequis**, c.f. section précédente concernant le poste de travail et ses prérequis.
- **Un espace de travail persistent**

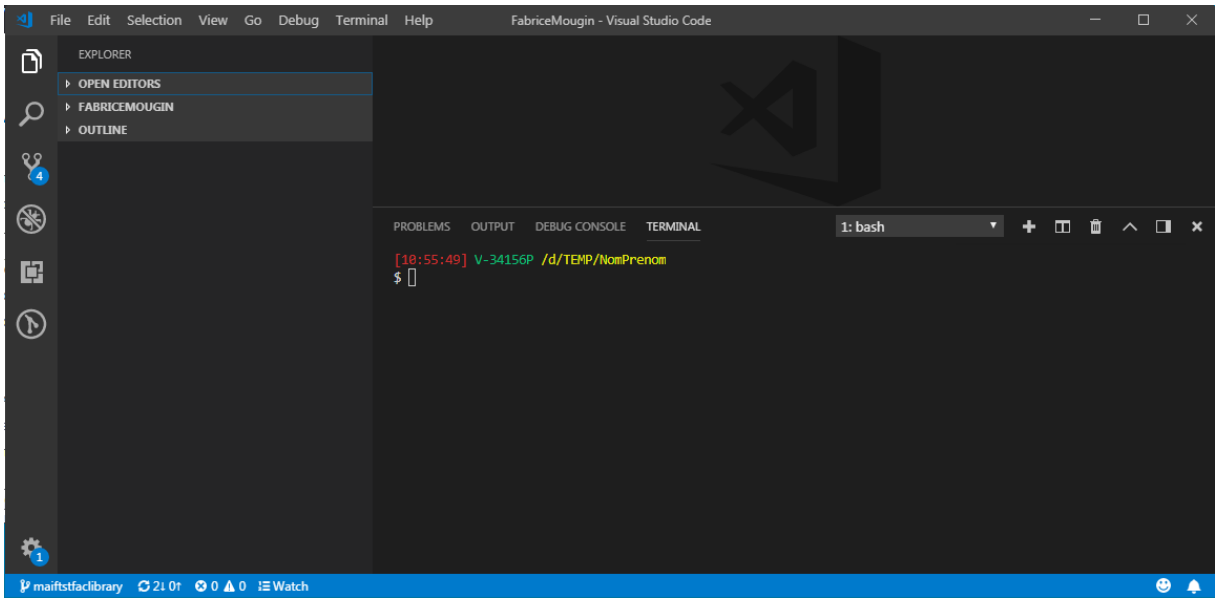
Procédure – ouverture IDE

Pour la suite, je vais considérer que l'espace persistent est d:\TEMP\ ceci vient de l'exemple concret de la machine Build internet – Windows 7 ou d:\ est un montage réseau.

Ouvrir un navigateur de fichier et créer vous un répertoire dans l'espace persistant sans espace ni accents ni caractères spéciaux Il faut bien entendu remplacer Nom et Prenom par votre nom et votre prénom en substituant les caractères indésirables :

D:\TEMP\NomPrenom\

Ensuite, lancez Visual Studio Code et ouvrez le répertoire D:\TEMP\NomPrenom dans le menu File > Open folder



Vérifiez bien que le répertoire est **/d/TEMP/NomPrenom** dans le terminal **1:bash**

Ce répertoire style UNIX correspond au répertoire **d:\TEMP\NomPrenom** windows.

Procédure – Git clone du projet

Vérifiez que vous êtes bien dans votre répertoire dans le terminal bash.

Dans le terminal lancer un git clone de votre projet :

```
git clone http://gitlab-fabfonc.maif.local/tnr_produits/tnr_<mon
projet>_rfw
```

Cette commande va créer un répertoire tns_<mon projet>_rfw dans /d/TEMP/NomPrenom

Il faut maintenant se déplacer dans le répertoire avec la commande cd :

cd tnr_<mon projet>_rfw

Pour ne pas avoir à remplir votre identifiant Git à chaque fois que vous faites une opération Git, veuillez lancer ceci dans le terminal à la suite des commandes précédentes (vous êtes dans tnr_<monprojet>_rfw) :

```
git config --global user.name "Prenom NOM"
git config --global user.email "votre adresse e-mail…"
git config credential.helper store
```

Il faut remplacer Prenom par le prénom avec une majuscule et le reste en minuscules.

Il faut remplacer le NOM tout en majuscules

Et bien sur les … par votre adresse mail.

Ici vous avez votre repository configuré automatiquement dans votre Visual Studio Code.

L'étape suivante sera de remplir votre projet avec une structure minimum pour commencer. Il y a une section qui suit dédiée à ce sujet.

Lorsque vous reviendrez travailler sur ce projet, la prochaine fois, vous aurez simplement le **cd tnr_<mon projet>_rfw** à faire dans votre GitBash, rien de plus. En effet, l'espace persistent gardera tout le reste.

Procédure – Installation de la librairie

Pour installer la librairie (déployée sur Nexus) tapez ensuite :

```
python -m pip install --index-url http://nexus-
fabfonc.maif.local/repository/pip-hosted/simple --extra-index-url
http://nexus-fabfonc.maif.local/repository/pip-proxy/simple --trusted-host
nexus-fabfonc.maif.local MaifTstFacLibrary --user
```

Une communication verbeuse de l'installation de la librairie et de ses dépendances suit.

Pour vérifier que la librairie est bien installée, veuillez taper :

```
python -m MaifTstFacLibrary --version
```

La sortie doit vous présenter ceci :

Version : MaifTstFacLibrary-M.m.n

C.f. l'explication sur le numéro de version dédié dans la documentation.

Ici vous avez une librairie complète installée dans votre système, vous êtes prêt à la configurer et à l'utiliser dans un projet.

Procédure – création du projet initial

Il est conseillé d'installer dans le projet vide, une structure de tests qui permet de vite démarrer :

Nous partons après le git clone du projet vide et le premier cd dans le projet.

Vous êtes dans votre IDE, visual studio code, dans un terminal Git bash, à la racine du répertoire de votre projet dans le terminal.

Pour installer les fichiers minimums, ouvrir un terminal GitBash dans VisualCode et tapez :

```
python -m MaifTstFacLibrary --gettingstarted all
```

Cette commande va installer dans votre projet les fichiers suivants :

```
Create JDD/sample.json
Create Jenkins/JenkinsFile
Create .gitignore
Create Helpers/install_helper.py
Create Helpers/run-test.py
Create Environment/credentials.ini
Create Environment/environments.ini
Create Environment/suite_variables.ini
Create Tests/PageObjects/Google/MPG_Google_Homepage.robot
Create Tests/PageObjects/Google/MPG_Google_ResultatRecherche.robot
Create Tests/PageObjects/Maif/MPG_Maif_Homepage.robot
Create Tests/PageObjects/Maif/MPG_Maif_Popup.robot
Create Tests/PageObjects/Maif/MPG_Maif_ResultatRecherche.robot
Create Tests/TestSuites/Google/TST_Google_RechercheSimple.robot
Create Tests/TestSuites/Maif/TST_Maif_RechercheSimple.robot l
```

Cette structure est définie dans une section dédiée.

La structure du template

Le repertoire Helpers

./Helpers/install_helper.py	L'installeur
./Helpers/run-test.py	Le runner

Le repertoire Jenkins

./Helpers/JenkinsFile	Un exemple de JenkinsFile

Le repertoire JDD

Contient des jeux de données JSON.

Le répertoire Logs

Ou se déversent tous les logs d'exécution en local.

Le répertoire des Tests

./Tests/Environment ou ./Environment

credentials.ini	Tous les login et mots de passe du projet cryptés ou non
environments.ini	Les variables URLs d'environement RECX RECL PPROD PPCOR
products.ini	Le ou les produits testés
proxies.ini	Les proxies ajoutés en plus de ceux livrés par la librairie
run_targets.ini	Les cibles en plus de celles livrées par la librairie
launch_variables.ini	Les variables de lancement
project_variables.ini	Les variables du projet et toutes les variables par défaut.

./Tests/TestSuites/*/*.robot

Les tests à jouer.

Ceux-ci ne doivent contenir que des keywords métier et la structure macro du ou des tests.

./Tests/PageObjects/*/*.robot

Le répertoire contenant les page objects.

C'est ici ou sont les keywords techniques contenant les modules utilisés par les tests ou d'autres « page object », ils contiennent les variables techniques, expression xpath, JSON path, référence CSS et les manipulation avec le navigateur.

La migration se fera par accompagnement par projet avec accompagnement si besoin, dans le cadre d'une épopée Stream Outils.

./Technique

Répertoire dédié aux librairies développées par les projets si il y en a.

Les TAGS

Voici un guide sur l'écriture des tags dans les scripts robot framework. Ceux-ci sont décrit dans la balise [Tags] de vos tests.

Le **tag identifiant unique**, comme par exemple 01_05_Habitation_AHA.

Il permet de lancer un seul test précis et de les identifier à postériori. Dans certains projets, ils sont préfixés par des numéros pour les classer. Ces numéros peuvent correspondre à une numérotation des .robot et des testcases par exemple.

Le **tag de domaine**, par exemple Habitation, il sera présent dans tous les tests Habitation. Il te permet de lancer tous les tests d'un domaine. Tag obligatoire pour les tests transverse.

Ensuite, il y a **les tags CI/CD** pour l'intégration continue.

Le **tag ETAT:...**, ,obligatoire, avec plusieurs états :

- developpement : le script est en cours de rédaction initlale
- validation : le script est fonctionnel, un run OK est passé, il est maintenant en cours de validation. C'est une phase d'observation initiale ou après un retour d'incident.
- stable : en production
- instable : retiré volontairement de la prod car régression ou investigation en cours. Une fois le problème réglé, il repassera en production ou en validation selon le risque.

Le **tag TYPE:...** ,obligatoire, avec une seule valeur l'instant, TNR, mais il y aura aussi :

- TF : tests fonctionnels
- TNF : tests non fonctionnels
- TA : tests d'acceptance
- API : tests d'API

Le **tag TRUST:** oui ou non, il permet de déterminer les tests de confiance. C'est la suite de test qui est lancé en premier pour savoir si on lance tout le reste. Si les tests de confiance échouent, cela n'est pas la peine d'aller plus loin. Ca permet de gagner du temps. Il est facultatif.

Le **tag PRODUIT:** nom du produit sans caractère spécial …, il permet de déterminer le produit testé.

Le tag **PRIORITE** : Px, il permet de définir une priorité :

- P1 : le plus prioritaire, critique
- P2 : important mais pas critique
- P3 : le reste

Cette priorité a été définie avec votre PO et/ou votre BA (risques produits).

Avec cette méthode de classement, il sera possible sans hésitation de lancer tous les tests critiques, sur le produit Nora, de confiance, en production par exemple sans avoir à solliciter l'équipe pour déterminer la liste.

La structure d'un test

```
*** Settings ***
Library           MaifTstFacLibrary
Library           SeleniumLibrary
Library           Collections

Resource    ../../PageObjects/Google/MPG_Google_Homepage.robot
Resource    ../../PageObjects/Google/MPG_Google_ResultatRecherche.robot

Documentation    Recherche simple sur Google
*** Test Cases ***
Google_RechercheSimple
    [Tags]                  RECHERCHE    GOOGLE  CONFIANCE:oui   PRODUIT:Google  ETAT:stable
    [Setup]                 Open TstFac Browser

    Page Navigation Accueil
    Page Recherche
    Page Retour Accueil

    [Teardown]              Close TstFac Browser

*** Keywords ***

# La structure d une etape
Page Recherche
    Action Soumission Recherche
    Action Verification Recherche
```

Il est important de bien vérifier les extensions visual studio code afin de bénéficier de la coloration syntaxique et de la possibilité de pouvoir suivre les liens vers les fonctions (appelées keywords) robot framework avec control + clic.

Dans un test, il ne doit y avoir que des keywords qui sont lisibles par quelqu'un de non scripteur. Il sera exclu par exemple d'y mettre des références au DOM ou des keywords trop technique.

Regardons ce test plus en détail :

Settings

```
*** Settings ***
Library           MaifTstFacLibrary
Library           SeleniumLibrary
Library           Collections       You, a month ago • Fix for env

Resource    ../../PageObjects/Google/MPG_Google_Homepage.robot
Resource    ../../PageObjects/Google/MPG_Google_ResultatRecherche.robot

Documentation    Recherche simple sur Google
```

Dans les Settings, il y a l'import de la librairie et de toutes les librairies nécessaires au test.

Il est conseillé si il y en a beaucoup, de les externaliser dans un fichier all.robot dans un sous répertoire Technique et d'y faire référence dans tous vos tests.

Dans ce répertoire Technique, vous pourrez aussi y mettre les librairies techniques qui n'ont pas été encore intégrées au framework parce que trop spécifique ou pas encore normalisée ou bien par manque de temps.

On trouve aussi dans Settings, tous les PageObjects de votre test.

Et enfin, une balise documentation de la suite de tests.

*** Test Cases ***

```
*** Test Cases ***
Google_RechercheSimple
    [Tags]                  RECHERCHE    GOOGLE  CONFIANCE:oui   PRODUIT:Google  ETAT:stable
    [Setup]                 Open TstFac Browser

    Page Navigation Accueil
    Page Recherche
    Page Retour Accueil

    [Teardown]              Close TstFac Browser
```

Cette balise contient les tests proprement dit.

La balise [Tags], défini les tags du test, il y a une section dédiée à ce sujet dans le document.

Les balises [Setup] et [Teardown] sont respectivement le points d'entrée et de sortie de la librairie.

Ils doivent à minima contenir Open TstFac Browser et Close TstFac Browser. Ces deux mots clés sont responsables de la mise à disposition d'un navigateur respectant les fonctionnalités demandées.

Si Open TstFac Browser ou Close TstFac Browser n'est pas appelé, ce comportement n'est pas supporté. Il peut y avoir un effet de bord de certaines options qui lancent des thread (comme – watchconsole True), si Open TstFac Browser n'est pas invoqué, le test reste bloqué car un thread est toujours en mémoire et personne n'est venu le terminer. Il faut alors tuer la tâche pour continuer.

Le reste décrit les tests métiers. Ils ne doivent contenir que des KeyWords métier, lisibles par une personne non scripteur.

*** KeyWords ***

```
*** Keywords ***

# La structure d une etape
Page Recherche
    Action Soumission Recherche
    Action Verification Recherche
```

Cette balise contient les KeyWords qui peuvent être découpés en sous KeyWords métiers toujours lisibles par un non scripteur.

La structure d'un page object

```
*** variables ***
${xpath_recherche_icone}       xpath=//i[@class="maificon maificon-loupe"]
${xpath_recherche_texte}      css=input[name=searchedText]
${xpath_recherche_bouton}      css=button[name=send]

*** keywords ***
Action Popup Recherche
    Wait Until Element Is Visible  ${xpath_recherche_icone}
    Click Element   ${xpath_recherche_icone}
    Wait Until Element Is Visible  ${xpath_recherche_texte}

Action Soumission Recherche
    Input Text  ${xpath_recherche_texte}  Electrique
    Wait Until Element Is Visible  ${xpath_recherche_bouton}
    Click Button    ${xpath_recherche_bouton}
```

Une ressource page object est une bibliothèque de keyword de navigation technique orienté métier. Les test cases s'appuient sur un ensemble de page objects.

Il contient des variables technique de l'application testée, par exemple des expression XPath ou CSS désignant des éléments de page à tester.

Il contient des KeyWords technique de navigation dans l'application testée.

Les page objects peuvent être organisés en fonctionnalités ou en entités de page.

En mode d'organisation par entité de page, par exemple, tous les KeyWords d'interaction avec le bandeau commun devront se trouver dans un page object contenant le mot bandeau. Si les résultats de recherche se trouvent dans un type de page bien particulier, idem, ils devront se retrouver dans un page object contenant le mot résultat de recherche.

Si l'application est organisée plutôt en étapes fonctionnelles par exemple, les page objects pourront porter le nom de l'étape fonctionnelle testée.

De manière générale, les page objects visent à diviser et organiser le code de test pour que les développeurs retrouvent intuitivement les keywords sans avoir à les chercher.

Les options de lancement de robot framework (runner)

Les options sont utilisables via le runner, en ligne de commande. Elles sont liées à des fonctionnalités.

Pour avoir la liste, veuillez taper :

```
./Helpers/run-test.py --help
```

Utilisation des variables de launch

Lorsqu'on lance un test souvent, dans le cas d'un scripteur qui travaille toujours en local sur firefox avec report portal et il ne veut pas du format de logs année mois jours (option –archive False) par exemple. Il doit taper ceci :

```
./Helpers/run-test.py –environment RECX --target local --archive False --
browsername firefox --reporter Disabled -i MAIFDEMO
```

Les variables de launch sont un fichier de configuration qui peut définir une grappe de fonctionnalités. Le fichier est situé ici :

```
Tests/Environment/launch_variables.ini
```

Ou

```
Environment/launch_variables.ini
```

Voici un exemple de contenu :

```
[Local]
RN_TARGET=local
RN_BROWSER=firefox
RN_LOG_MODE=noarchive
RN_REPORTER=Disabled

[Grid]
RN_TARGET=Selenoid
RN_BROWSER=firefox
RN_LOG_MODE=archive
RN_REPORTER=Report_portal

[RECX]
RN_ENVIRONMENT=RECX

[PPCOR]
RN_ENVIRONMENT=PPCOR
```

Maintenant le testeur pourra lancer ceci avec le même effet :

```
./Helpers/run-test.py –launchvariables Local,RECX -i MAIFDEMO
```

Les variables de launch pourront servir par exemple pour déterminer des grappes de variables pour la PROD, pour une exécution Jenkins . A définir de manière plus précise lors de la bascule des projets.

Pour avoir la liste complète des variables de launch possible, veuillez taper :

```
./Helpers/run-test.py --launchvariables help
```

Pour avoir un exemple complet de fichier de variables de launch, veuillez taper :

```
./Helpers/run-test.py --launchvariables example
```

Utilisation des variables du projet

Ces variables sont les variables qui concernent tout le projet. Le runner y puise les variables qui ne sont pas définies dans les options ni dans les variables de launch.

Elles sont dans :

```
Environment/project_variables.ini
```

La section Project contient pour l'instant juste le nom du projet. C'est celui qui est définit dans report portal.

La section ReportPortal, contient les varaibles qui seront injectées dans report portal.

La section Runner, contient les variables par défaut non spécifiées dans une launch variable ni une option.

Voici un exemple de variables de projet :

```
[Project]
PROJECT_NAME=rfw

[ReportPortal]
; Petit texte qui se retrouve dans la description
DEFAULT_RP_DESCRIPTION=Run de mon projet TNR
; Front ou Backoffice ou TNR ...
DEFAULT_RP_RUN_NAME=TNR
; Tags separes par une virgule
DEFAULT_RP_ATTRIBUTES=DEBUG,MAP
; Tags cle:valeur separes par une virgule
DEFAULT_RP_ADDITIONAL_KEYS=domaine:nondefini,application:nondefini
[Runner]
DEFAULT_RN_TARGET=Selenoid
DEFAULT_RN_BROWSER=firefox
DEFAULT_RN_LOG_MODE=archive
DEFAULT_RN_REPORTER=Report_portal
```

Utilisation de report portal

L'utilisation de report portal nécessite un compte sur la machine suivante et un projet défini :

`http://reporportal.maif.local`

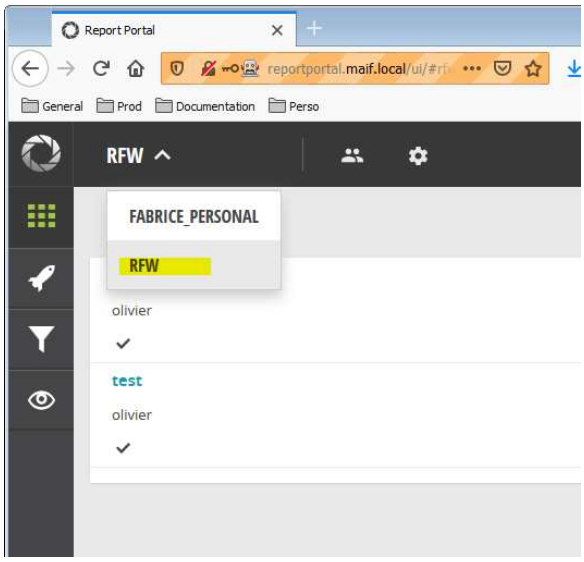
Pour activer cette option, il suffit d'ajouter l'option suivante et un projet défini :

`--reporter Report_portal`

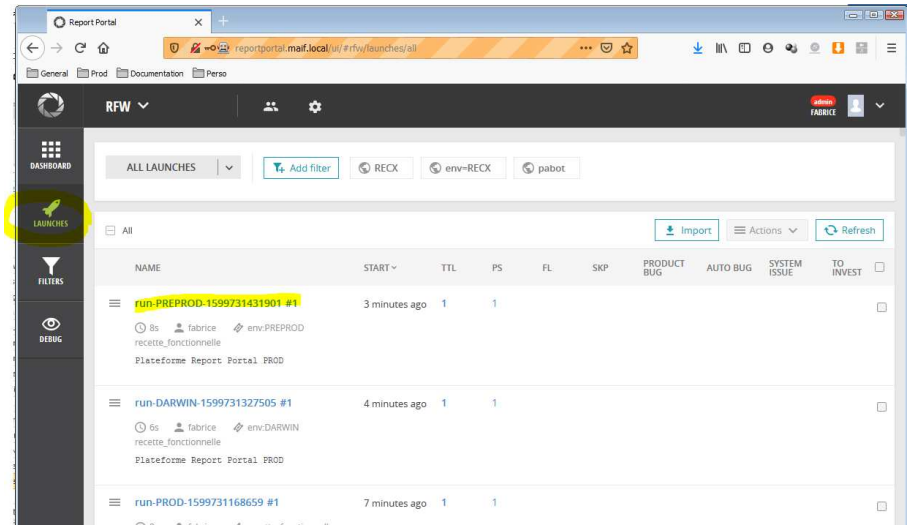
Pour avoir la liste des reporter, il suffit de lancer l'option avec la valeur help.

`./Helpers/run-test.py --reporter help`

Pour visualiser la mesure, il faut aller dans report portal et choisir la file de run :



Les launches sont visibles ici :



Cas d'utilisation

Pour tous les cas d'usage :

- se mettre dans visual studio code
- dans un git bash
- dans le répertoire de votre projet contenant :
    - Helpers
    - Logs
    - Tests/Environment
    - Tests/TestSuites
    - Tests/PageObjects

Execution d'un test en local avec firefox

```
./Helpers/run-test.py –i MAIFDEMO
```

Cette ligne de commande va lancer le test dont le tag est MAIFDEMO avec le navigateur par défaut, en local.

Execution sur la grid

```
./Helpers/run-test.py -i MAIFDEMO --target Selenoid
```

Cette ligne de commande va lancer le test sur la Grid Selenoid de production au lieu de le lancer en local.

Execution sur la grid avec parallelisme 2

```
./Helpers/run-test.py --target Selenoid --parallel 2 Tests/
```

Cette ligne de commande va lancer tous les tests sur la Grid Selenoid de production avec un parallélisme de 2.

Execution avec une configuration proxy sur la grid

Si vous avez besoin de voir tous les cas d'usage, vous pouvez lancer le runner avec l'option -X et vous aurez tous les cas d'utilisation maintenus dans les tests d'intégration. Ce sont des exemples de lancement, si vous voulez les lancer, il faut préalablement aller chercher le template d'exemples évolués associé : http://gitlab-fabfonc.maif.local/tnr_produits/poc_rf/template_maiftstfaclibrary

Fonctionnalités avancées

La toolbox

C’est une librairie de keywords fréquemment utilisés par les projets.

Elle sera maintenue, documentée et étendue au cours du temps.

Voici son contenu à ce jour :

- coller_element(xpath)
- set_cookie(valeur)
- current_browser()
- ouvrir_tab(url)
- change_tab(numeroonglet)
- fermer_tab(numeroonglet1, numeroonglet2)
- press_enter()
- press_tab()
- press_uparrow()
- press_space()
- press_downarrow()
- press_key_directly(touche)
- press_end()

Si des mots clés manquent , veuillez avertir le canal Automatisation des tests > @Rework framework RBW.

Logs de la console firefox

La librairie est capable de se connecter par web socket à Selenoid et de vérifier la présence de mots clés dans les logs de la console firefox ou bien de faire le même contrôle en local.

Pour activer cette fonctionnalité, il faut :

- être sur la GRID ou en local
- ajouter l'option --watchconsole True

Un exemple du template MaifTstFacLibrary est :

```
./Helpers/run-test.py -c --debug 5  -E SELENOIDLOGS --target
Selenoid_PREPROD --watchconsole True -i MAIFDEMO .
```

Les keywords qui permettent de vérifier les logs sont :

- Log Observe : démarre l'observation des logs, ce mot clé doit être positionné avant Log Raise et Log Check.
- Log Raise Regex <expression régulière> : vérifie que l'expression régulière est bien présente dans les logs et lève une exception sinon.
- Log Raise <expression texte> : vérifie que l'expression texte est bien présente dans les logs et lève une exception sinon.
- Log Check Regex <expression régulière> : vérifie que l'expression régulière est bien présente dans les logs et renvoie faux sinon.
- Log Check <expression texte> : vérifie que l'expression texte est bien présente dans les logs et renvoie faux sinon.
- Log Get Line <expression texte> : vérifie que l'expression texte est bien présente dans les logs et renvoie la ligne trouvée, None sinon.
- Log Get Line Regex <expression régulière> : vérifie que l'expression régulière est bien présente dans les logs et renvoie la ligne trouvée, None sinon.

Le résultat de l'observation est dans Logs/logme.log si besoin.

Envoi d'informations à NeoLoad

La librairie est capable d'envoyer des information à l'outil NeoLoad.

Le mode automatique ne nécessite pas de modification du code. Il faut cependant que le test :

- Utilise Open TstFac Browser : ce mot clé lance le start neoload
- Utilise New Step pour déclarer les keywords métiers principaux, ce mot clé lance une nouvelle transaction ou bien New Transaction <nom de la transaction> si le mot clé New Step n'est pas utilisé dans la suite des tests.
- Utilise Close TstFac Browser : ce mot clé fait un stop NeoLoad

Les options pour l'activer :

- --neoloadmode : définit les modes
- --neoloadurl : l'URL du serveur NEOLOAD

Les modes sont :

- run ou dryrun : active le mode réel ou le mode test
- auto ou rien : active le mode automatique ou celui qui necessite l'utilisation des KeyWords NeoLoad Start, NeoLoad Stop et Neoload New Transaction

L'exécution se fait comme ceci :

```
./Helpers/run-test.py -c --neoloadmode='auto,dryrun' --
neoloadurl=http://localhost/ -i MAIFDEMO .
```

Pour une exécution plus personnalisée, les mots clés suivants peuvent être utilisés hors mode auto :

- NeoLoad Start
- NeoLoad Stop
- Neoload New Transaction <nom de la transaction>

Avec l'option --neoloadmode='dryrun' ou --neoloadmode='run'.

JSON et JSON Path

La librairie intègre des mots clés pour l'utilisation de Json et Json Path :

- Json Load <chemin> : charge un Json et renvoie un pointeur vers une structure Json
- Json Save <pointeur vers Json> <chemin vers fichier> : sauvegarde le Json
- JsonPath Match <Expression Json Path> <pointeur vers Json> : renvoie un sous Json Filtré

Un exemple est utilisé dans le test Tst_Google_RechercheSimple.robot, décrit dans le page object MPG_Google_Setup.robot.

Rétrocompatibilité robot et pybot/pabot

Pour une utilisation avec un outil tiers tel que rfswarm ou tout autre outil utilisant robot framework, parfois il est imposé de travailler avec robot et il est donc impossible d'utiliser le runner. Le runner est très pratique pour les scripteurs mais pas indispensable.

Pour lancer un script en mode compatibilité avec robot par exemple, il faut simplement reprendre l'option, la passer en majuscules et ajouter TF_ en préfix et la passer en variable d'environnement.

Voici un exemple concret d'usage.

Le lancement à migrer est :

```
./Helpers/run-test.py -i MAIFDEMO --target Selenoid
```

En mode de retro compatibilité robot:

```
TF_TARGET=Selenoid robot -i MAIFDEMO .
```

En mode de retro compatibilité pabot :

```
TF_TARGET=Selenoid pabot -i MAIFDEMO .
```

En mode de rétrocompatibilité, nous n'avons pas toute les fonctions. Par exemple, Report_portal n'est pas supporté, ni l'extension NeoLoad, ni la fonctionnalité watchconsole.