MEDIA > ARTICLE

Ecrire ses keywords Robot Framework avec Python

8 min

Testing

19/11/2020

Voici une histoire inspirée de ma vie professionnelle :

Moi:

"J'ai bien avancé dans le développement de la communication Bluetooth entre notre frigo connecté et l'application PC avec une super bibliothèque en Python! Maintenant j'aimerais bien écrire des tests et les automatiser."

Des gens de mon équipe :

"Utilise Robot Framework! Tu verras: c'est cool!"

Moi!

"OK, je tente ça !"

Il y a plein de tutoriels Robot Framework sur le net, mais en général on te ntre comment te connecter une page web pour t'authentifier ou comment interroger un *endpoint* REST pour faire 2+2=4 (j'exagère à peine). Autant dire que j'étais très loin de me connecter en Bluetooth à mon frigo...

Dans cet article, je vais vous expliquer comment interfacer Robot Framework avec une bibliothèque en Python pour automatiser les tests d'un frigo connecté (qui est fictif, vous l'avez compris).

Robot Framework en bref

Je vais laisser Robot Framework se présenter, en citant sa page officielle :

Robot Framework is a generic open source automation framework. It can be used for test automation and robotic process automation (RPA).

[...]

Robot Framework itself is open source software released under Apache License 2.0, and most of the libraries and tools in the ecosystem are also open source. The framework was initially developed at Nokia Networks and was open sourced in 2008.

Robot Framework est principalement écrit en Python et le code est disponible sur GitHub.

On peut l'installer facilement avec pip puisque les packages sont disponibles sur PyPi :

\$ pip install robotframework





Les keywords : le nerf de la guerre

L'écriture de tests en Robot Framework repose sur l'utilisation de *keywords*. Les *keywords* (qui ressemblent des fois beaucoup à des *keyphrases*) permettent de réaliser une action ou une vérification.

Voici un exemple d'un test case trouvé sur le site officiel :

```
*** Test Cases ***
Valid Login
Open Browser To Login Page
Input Username demo
Input Password mode
Submit Credentials
Welcome Page Should Be Open
[Teardown] Close Browser
```

Chaque ligne est composée d'un *keyword* et d'éventuels paramètres. Il est possible d'écrire ses propres *keywords* à partir de *keywords* existants. On pourrait par exemple écrire le *keyword* suivant :

```
*** Keywords ***
Input Credentials

[Arguments] ${username} ${password}

Input Username ${username}

Input Password ${password}

https://www.younup.fr/blog/ecrire-ses-keywords-robot-framework-avec-python
```

On peut ainsi réécrire de manière simplifiée le test case précédent avec ce nouveau *keyword* :

```
*** Test Cases ***
Valid Login
Open Browser To Login Page
Input Credentials demo mode
Submit Credentials
Welcome Page Should Be Open
[Teardown] Close Browser
```

lci, je ne fais qu'écrire des *keywords* à partir de *keywords* existants. Mais comment faire si on veut créer des *keywords* vraiment nouveaux, genre des *keywords* pour se connecter en Bluetooth à un frigo et lui envoyer une nouvelle consigne de température ?

Sur la page officielle de Robot Framework, on peut lire :

Its capabilities can be extended by libraries implemented with Python or Java.

Très bien. On fait comment?

Créer sa bibliothèque de keywords avec Python

J'avoue que mes premiers pas pour interfacer Robot Framework avec ma bibliothèque Python existante ont été laborieux. La documentation officielle est très détaillée, mais elle est difficile à appréhender pour débuter. Elle tient clairement plus du *Reference Manual* que du *Getting Started*. Je me suis retrouvé à regarder des vidéos YouTube de *live coding* d'une boite de consulting indienne.





En fait, il existe plusieurs façons d'appeler du code Python depuis Robot Framework. La solution simple est suffisante dans beaucoup de cas. C'est celle que j'ai utilisée et c'est celle que je vais vous présenter ici. Elle est nommée "Static API" dans la documentation :

The simplest approach is having a module (in Python) or a class (in Python or Java) with methods which map directly to keyword names. Keywords also take the same arguments as the methods implementing them. Keywords report

failures with exceptions, log by writing to standard output and can return values using the return statement.

Petites remarques:

- Les fonctions Python dont le nom commencent par un _ sont cachées dans Robot Framework.
- Une fonction Python s'appelant input_credentials() peut être utilisée en Robot Framework comme Input Credentials , input credentials , Input credentials ou encore input_credentials (bref, vous avez le choix).
- si comme moi vous avez déjà une bibliothèque Python, il est certainement intéressant de ne pas l'importer directement dans Robot Framework mais de faire un wrapper intermédiaire pour exposer de jolis keywords bien pensés, comme suggéré dans cette discussion sur stackoverflow.

Exemple simple

Voici un exemple de bibliothèque Python prête à être importée dans Robot Framework, qu'on appellera fridge.py :

```
import robot.api.logger

///
mport robot.utils.asserts
```

```
# Variable that is supposed to be in the fridge firmware
temperature setpoint = 5
def ble send setpoint(value):
    """Pretend to send data to the fridge with the new
temperature setpoint.
    :param value: the setpoint value
global _temperature setpoint
    temperature setpoint = value
    print('Sending setpoint to the fridge: ',
temperature setpoint)
def ble read setpoint():
    """Pretend to read data to the fridge to get the
temperature setpoint.
    :return: the setpoint value
    global temperature setpoint
    print('Reading setpoint to the fridge: ',
temperature setpoint)
    return _temperature_setpoint
def change_temperature_setpoint(value):
    """Function exposed as a keyword to change the
temperature setpoint.
    :param value: the setpoint value
    \mathbf{n} \mathbf{n} \mathbf{n}
    value = float(value)
    robot.api.logger.info('Change temperature setpoint to
    % value)
     ble send setpoint(value)
```

```
def check_temperature_setpoint(expected):
    """Function exposed as a keyword to check the
temperature setpoint.

:param expected: the expected setpoint value
    """
    expected = float(expected)
    actual = _ble_read_setpoint()
```

Les deux premières fonctions simulent une communication avec le frigo connecté pour lire et écrire la consigne de température. Une variable locale stocke cette valeur, simulant la mémoire du processeur du frigo. Comme leurs noms commencent par un *underscore*, elles ne seront pas importées comme *keywords*. Elles ne sont pas nécessaires ici, elles servent simplement à vous montrer le principe.

robot.utils.asserts.assert equal(expected, actual)

Les deux dernières fonctions en revanche seront bien importées comme des *keywords*.

Remarquez la conversion des arguments avec la fonction float(). En effet, les arguments sont transmis de Robot Framework vers Python comme étant des strings par défaut et ce n'est pas le format qui nous arrange ici. On transforme donc ces strings en nombres.

Maintenant, écrivons donc un petit *test case* pour se servir de nos super *keywords*!

```
*** Settings *** Library fridge.py
```

*** Test Cases *** It is possible to change the temperature setpoint of the fridge change_temperature_setpoint 4 check_temperature_setpoint 4 change temperature setpoint 8 Check temperature setpoint 8 Change_Temperature

[POINT 3.5 CHECK temperature_SETpoint 3.5]

Je vous l'avais dit : les underscores et la casse sont gérés en toute souplesse !

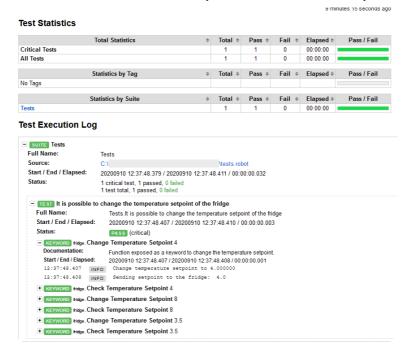
Enregistrons ces tests dans un fichier tests.robot (ce fichier sera considéré comme une test suite nommée Tests) et exécutons-les depuis notre terminal :

Si vous n'avez jamais utilisé Robot Framework, il génère un superbe rapport HTML avec tous les détails. Le résumé ressemble à ça :



Pour chaque test, on a le détail complet des opérations faites et on retrouve les logs faits depuis le code Python, la documentation des fonctions grâce aux

cstring, etc :





LE MEDIA

CONSULTANT

ENTREPRISE

d'imagination et pas mal de lecture de documentation. Vous avez toutes les bases pour aller plus loin dans l'écriture de vos tests automatisés.

A titre d'exemple des possibilités qui s'ouvrent à vous, je vais vous montrer comment "automatiquement" se connecter au frigo au début de chaque *test case* et se déconnecter à la fin de celui-ci. Car oui, avant de communiquer avec un système, il faut s'y connecter et on n'a pas envie de le faire manuellement dans chaque *test case*. Déjà parce que c'est pénible, mais surtout parce qu'il y a un risque d'oubli et donc d'erreur.

Pour faire ça, je rajoute 2 fonctions à ma bibliothèque [fridge.py] :

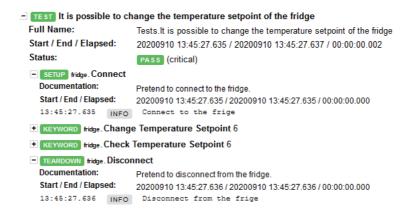
```
def connect():
    """Pretend to connect to the fridge."""
    robot.api.logger.info('Connect to the frige')

def disconnect():
    """Pretend to disconnect from the fridge."""
    robot.api.logger.info('Disconnect from the frige')
```

Je configure ma test suite pour que le setup d'un test case établisse la connexion et que le teardown procède à la déconnexion :

*** Settings *** Library fridge.py Test Setup Connect Test Teardown Disconnect *** Test Cases *** It is possible to change the temperature setpoint of the fridge Change temperature setpoint 6 Check temperature setpoint 6

Je constate dans le rapport que mes fonctions sont bien appelées :



Conclusion

Vous savez maintenant comment écrire des fonctions en Python et les importer dans Robot Framework pour les utiliser en tant que *keywords*.

La solution la plus simple est d'écrire un fichier mon_module.py et de l'importer dans Robot Framework dans la section *** Settings *** de votre fichier de tests avec la commande Library mon_module.py. Toutes les fonctions dont le nom ne commence pas par un *underscore* seront exposées en tant que *keywords*.

Allez, je vous laisse, j'ai un frigo à tester!





ça t'a plu ? Partage ce contenu



Pierre

Que la vie de Pierre, expert embarqué Younup, serait terne sans les variadic templates et les fold expressions de C++17. Heureusement pour lui, Python a tué l'éternel débat sur l'emplacement de l'accolade : "alors, à la fin de la ligne courante ou au début de la ligne suivante ?"

Homme de terrain, il est aussi à l'aise au guidon de son VTT à sillonner les chemins de forêt, dans une salle de concert de black metal ou les mains dans les soudures de sa carte électronique quand il doit déboguer du code (bon ça, il aime moins quand même!)

Son vœu pieux ? Il hésite encore... Faire disparaitre le C embarqué au profit du C++ embarqué ? Ou stopper la génération sans fin d'entropie de son bureau.

Publications conseillées

Vous en voulez plus sur cette thématique ? Découvrez nos autres contenus !







© Younup 2022

Mentions Légales

