

ROBOT FRAME WORK/

Introduction



Table des matières

1	Généralités	1
1.1	Introduction aux tests automatiques	1
1.2	Différents types de tests	1
1.3	Approches xDD : TDD, ATDD et BDD	2
2	Introduction à Robot Framework	5
2.1	Structure "usuelle" du repertoire de test	7
2.2	Description des fichiers de tests RFW	9
2.3	Installation et configuration de RFW	13
2.4	Les librairies	14
2.5	Options pour lancer RFW	15
2.6	Les variables automatiques	17
2.7	Structure conditionnelle	19
2.8	Les boucles for	19
3	La bibliothèque standard	21
4	Bonnes pratiques de codage	23

Table des figures

2.1	Eco-système de Robot Framework	6
2.2	Architecture de Robot Framework	8

Liste des tableaux

2.1	Squelette d'un test RFW	10
2.2	Passage d'arguments dans RFW	12
2.3	Passage d'arguments dans RFW II	12
2.4	Quelque Options de RFW	16
2.5	Les variables automatiques de RF	18

1

Généralités

1.1 Introduction aux tests automatiques

L'automatisation des tests joue un rôle crucial dans le développement logiciel parce que les tests manuels sont répétitifs, chronophages et sujets aux erreurs. L'utilisation d'outils d'automatisation de tests permettent d'augmenter le taux de couverture en test, d'améliorer la qualité des logiciels et minimiser l'apparition de bugs (ou defects). En gros de s'assurer que les fonctionnalités respectent les exigences du cahier de charges.

1.2 Différents types de tests

Dans l'environnement du test logiciel, on regroupe souvent les tests en plusieurs familles, dont voici quelques unes :

- **Tests de non regression** : permettent de vérifier que des modifications n'ont pas entraîné d'effets de bord de nature à dégrader la qualité d'une version antérieurement validée [2].
- **Tests d'intégration** : consistent à vérifier que tous les modules quand ils sont intégrés fonctionnent de façon attendue.
- **Tests d'acceptation** : déterminent si les exigences d'un cahier des charges sont respectées.

- **Tests fonctionnels et tests non fonctionnels** : pour les tests fonctionnels, l'objectif est de garantir que la fonctionnalité du système respecte les exigences demandées dans le cahier des charges.

Les tests fonctionnels peuvent être complétés par des tests non-fonctionnels qui permettent de vérifier que le logiciel fonctionnera correctement en conditions de production. Ce sont par exemple les tests de performance, de charge, de stress, de volume, de fiabilité, etc.

- **Test de charge, stress test** : le test d'une application sous une charge importante mais attendue est appelé test de charge. Lorsque la charge placée sur le système est élevée ou accélérée au-delà

de la plage normale, le test correspondant est connu sous le nom de stress testing.




- **Tests unitaires** : Les unités ont une API qui est utilisée lors de l'interaction avec d'autres unités et est également utilisée pour les tester. Le test unitaire teste la fonction correcte et élémentaire d'un module.
- **Sanity et smoke tests** : Les "sanity tests" et les "smoke tests" sont des moyens d'éviter de perdre du temps et de fournir des efforts inutiles en déterminant rapidement si une application est trop "imparfaite" pour mériter des tests rigoureux.

Le smoke testing vérifie les fonctionnalités critiques du système tandis que sanity testing vérifie les nouvelles fonctionnalités comme les corrections de bugs.

Les "smoke tests" sont un sous-ensemble des tests d'acceptation, tandis que les "sanity tests" sont un sous-ensemble des tests de régression. Les "smoke tests" vérifient l'intégralité du système de bout en bout tandis que les "sanity tests" vérifient uniquement un composant particulier.



1.3 Approches xDD : TDD, ATDD et BDD

- Le **TDD : Test Driven Development** est une approche d'écriture de tests avant l'implémentation du code où les tests pilotent le développement du produit. C'est une approche itérative dans laquelle : on ajoute un test - on regarde le test échouer - et on refactor le code pour réussir à nouveau le test. Une fois tous les tests réussis, le développement de la fonctionnalité est considérée comme terminée.
- La version collaborative de TDD est connue sous le nom d' **ATDD : Acceptance TDD** ou développement basé sur les tests d'acceptation. Ici les personnes de différents domaines, par exemple développeurs, testeurs, analystes métier, clients, utilisateurs finaux etc, collaborent ensemble et décident des tests d'acceptation avant l'implémentation. La principale différence entre TDD et ATDD est que le TDD se concentre davantage sur les tests unitaires tandis que ATDD se concentre davantage sur les tests d'acceptation.
- **BDD : Behavior Driven Development** combine les principes TDD et ATDD. BDD est un processus ou un style de développement logiciel agile qui définit le comportement attendu d'une application du point de vue d'un futur utilisateur.
BDD se concentre davantage sur le comportement de l'application plutôt que sur ses détails techniques. Il explique le comportement de l'application de telle manière que tout le monde (développeurs, clients, équipe support, ...) a la même compréhension.

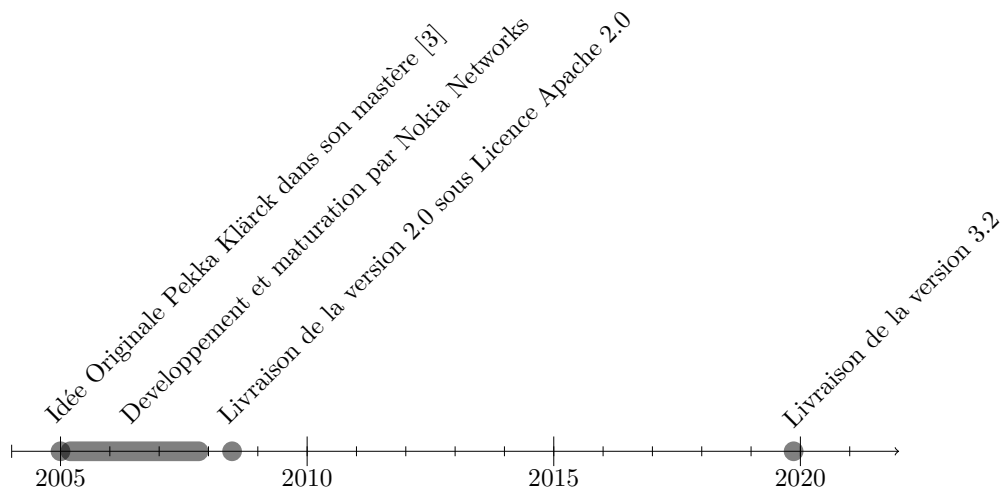
L'orchestration, integration des tests peut se faire avec des outils d'intégration continue comme Jenkins. , Gitlab , CircleCI .

2

Introduction à Robot FrameWork

Robot FrameWork (RFW)  est un interpréteur et séquenceur de tests de validation logicielle et d'automatisation des processus métier créé par Pekka Klärck et al. en 2005 [3], utilisé longuement en interne chez NSN  et finalement rendu open source en 2008.

RFW est simplement une surcouche qui permet de décorrélérer **l'implémentation en langage de programmation plus bas niveau du test** (" par exemple, écrire dans le premier champ puis cliquer sur le 3ème bouton dans le second formulaire, ...") et son **sens fonctionnel** ("Je m'identifie sur la page d'accueil") : en gros on se focalise plus sur la fonctionnalité souhaitée que le détail.



Pekka (Laukkanen) Klärck est un testeur, développeur et consultant Agile indépendant spécialisé dans l'automatisation des tests ; auteur original et le développeur

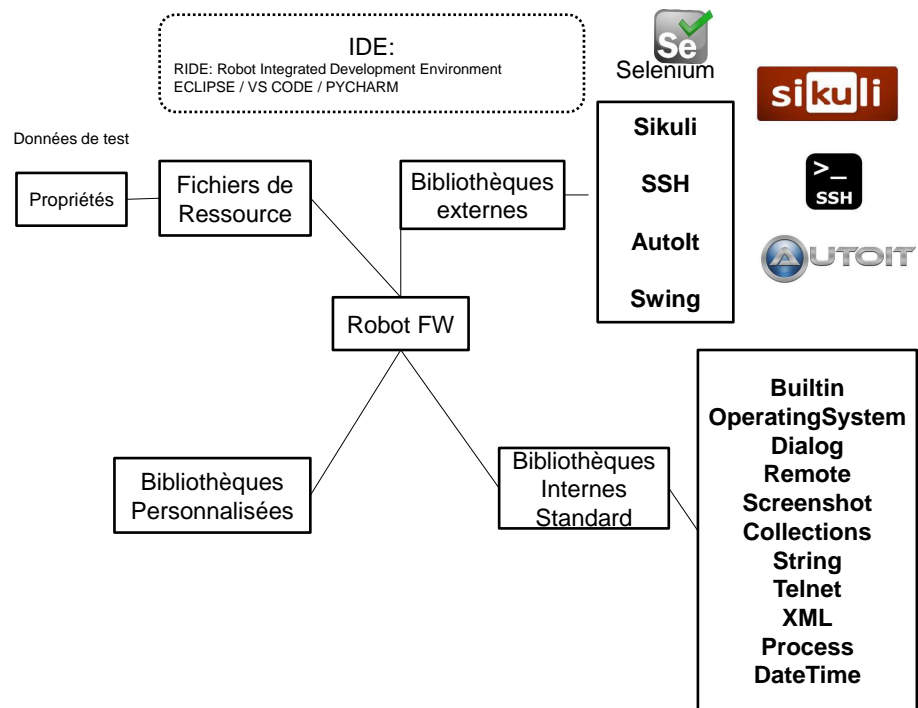



FIGURE 2.1 – Eco-système de Robot Framework principal de Robot Framework.

Comme frameworks de tests analogues on pourrait citer Gauge  ou Cucumber .






👍 Robot Framework utilise une approche mots clés. Elle encapsule la mise en œuvre de tests derrière les actions de haut niveau. Cette abstraction permet de rendre les tests lisibles, faciles à créer et à maintenir. Les tests sont construits comme une séquence de mots clés qui par la suite sont traduits en scripts python de plus bas niveau.

👍 RFW est Open source , hautement extensible, à support étendu à partir de bibliothèques tierces pour de nombreux types d'objectifs différents. Avec RFW on a la possibilité de créer autant d'abstraction pour permettre au langage naturel d'être utilisé au plus haut niveau afin que tout le monde puisse comprendre les rapports et les journaux de test -> très bons rapports et journaux de tests.

👍 Grande communauté qui contribue à créer des bibliothèques diverses et variées. Le site de référence pour trouver des informations sur RFW est robotframework.org. Il existe aussi un canal slack pour participer aux discussions techniques (dans la section support et contact de robotframework.org).

La Figure 2.1 résume l'écosystème de RFW. Avec ses différents compo-

sants :

- **La bibliothèque standard**, qui contient plusieurs fonctions liées au système d'exploitation, à la gestion des chaînes de caractère, aux structures de données, aux process, à la datation,
- RFW possède de nombreux outils et bibliothèques externes de tests comme **Sikuli**, **SSH** pour la gestion de tunnel SSH, **AutoIt**, **SWING**. L'une des plus utilisée est probablement **Selenium WebDriver**. Selenium  est une suite d'outils d'automatisation de tests d'interfaces graphiques web¹ : ses principaux composants sont Selenium IDE, Selenium RC, Selenium WebDriver et Selenium Grid [1]. Selenium Server (Grid) par exemple est utilisé pour lancer des tests en remote sur un webdriver selenium.
AutoIT : permet de manipuler des applications MS Windows.
FTP Library : permet d'effectuer des tâches FTP en simulant un client FTP.
- **Des bibliothèques personnalisées (customisées)** écrites en python ou en java ². RFW s'exécute également avec Jython qui est une implémentation de java en python et sur Iron Python, une implémentation sur .NET.
- **Editeurs** : RIDE (RFW IDE est un éditeur natif et autonome de RFW) ou d'autres IDE comme Pycharm , Eclipse , VSCode  avec des plugins adaptés .
- **Les données de tests** stockées dans des formats variables, les tables based formats (tsv, csv, xlsx) ou dans des fichiers python.

2.1 Structure "usuelle" du repertoire de test

Une hierarchie repandue des repertoires de tests consiste à avoir l'organisation suivante ; représentée dans la figure ci-dessous :

1. Selenium est comparable à Cypress qui demande une plus grande connaissance de javascript.

2. Python est le langage principal utilisé pour étendre RFW.

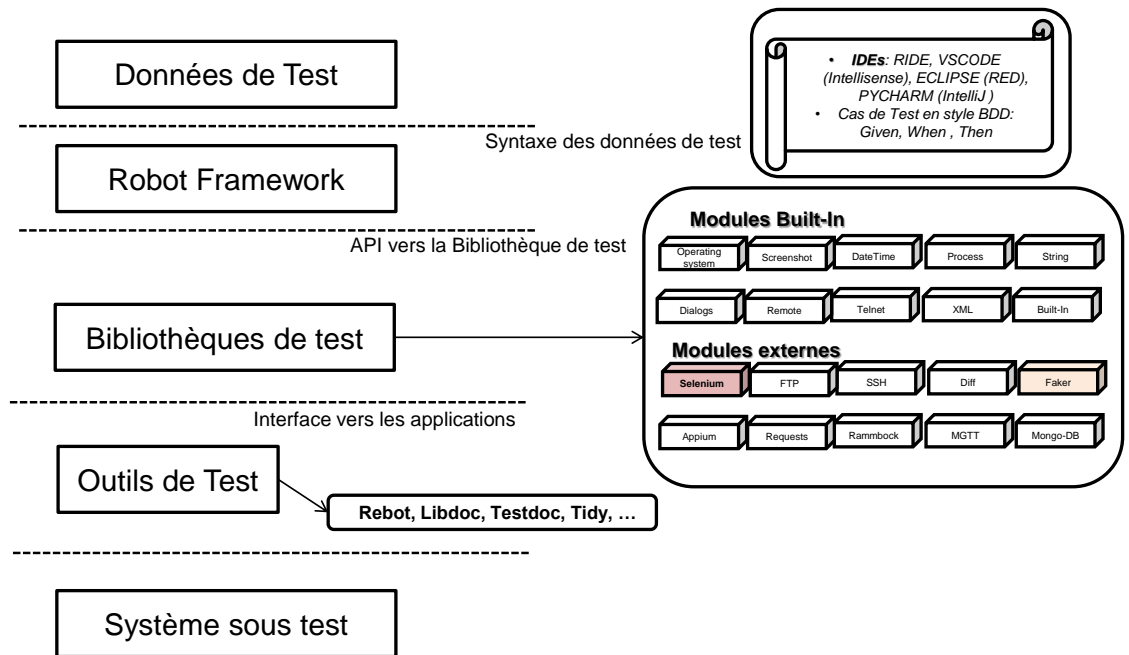
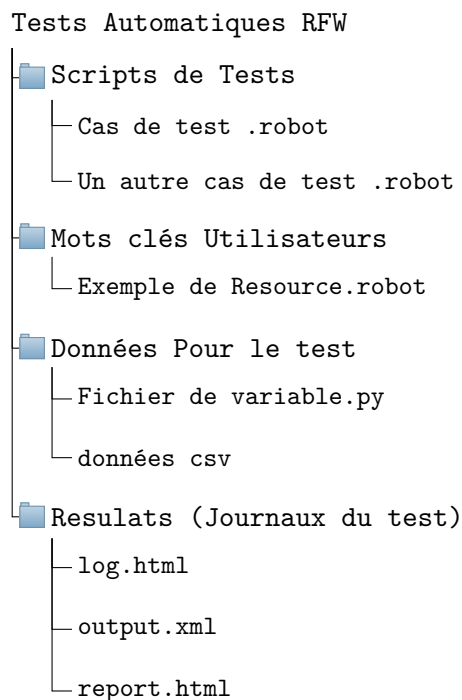


FIGURE 2.2 – Architecture de Robot Framework



- un repertoire pour **les scripts de test** : c'est ici que l'on definit le scénario de test avec les différentes étapes du test. On fait appel à des mots clés definis dans des fichiers de ressources, des librairies builtin ou des librairies externes.
- un repertoire où sont definis **les mots clés utilisateurs**. Les mots clés utilisateurs sont des mots clés créés soit à partir d'autres mots clés natifs à RFW ou à partir de scripts python.
- un repertoire où sont contenues les données du tests.
- un repertoire pour les **journaux et sorties du test**. Le fichier html par exemple contient tout ce qui se passe au cours des tests : les différentes valeurs d'une variable, retour de keywords, capture d'écran, log de dictionnaire...

2.2 Description des fichiers de tests RFW

Les tests Robot sont scriptés dans un fichier avec l'extension .robot. Dans n'importe quel fichier test robot, il y a des sections obligatoires, leur ordre n'est pas important. Les sections sont identifiées par un motif de trois astérisques (***) qui entoure un entête : Settings, Variables, Keywords, Test Cases, Tasks.



Une suite de tests est un fichier qui contient un ou plusieurs cas de test. Un cas de test est une suite de mots-clés avec ou sans arguments. Les mots clés sont soit natifs à la framework ou écrits en Python  (ou Java .

TABLE 2.1 – Squelette d'un test RFW

Settings			
Documentation	Un exemple de test générique		
Resource	chemin/resource.robot		
Library	DateTime		
Library	LibrairieExterne		
Variables	VariableDuTest.py		
Suite Setup	Action de setup de la suite		
Suite Teardown	Action de Teardwon de la suite		
Variable			
&{dict}	clef1=valeur1		clef2=valeur2
@{list}	elt1	elt2	elt3
\${scalar}	valeur		
*** Test Cases ***			
Nom de Test			
[Tags]	Tag1		Tag2
[Setup]	Action a faire avant le test		
Exemple d'appel de mot clef			
[TearDown]	Action a faire après le test		
*** Keywords ***			
Exemple d'appel de mot clef			
	Mot clef provenant d une librairie externe		
	Mot clef provenant d une librairie interne		

Les différentes sections possibles dans un fichier de script robot, sont les suivantes :

- ***** Settings ***** : Ici on inclut :
 - La documentation de la suite de test (sur une ou plusieurs lignes).
 - Les fichiers ressources, qui sont des fichiers contenant les mots clés customisés définis dans d'autres fichiers .robot ; ou .ressource, pour des besoins de généricité. Ils sont susceptibles d'être appelés de multiples fois dans nos tests.
 - Variables : le ou les fichiers où sont stockées les données des tests.
 - Les actions (mots clés) à effectuer lors du TearDown/Setup de la suite.
 - Les librairies requises : où certains mots clés sont définis en utilisant python ou java.
- ***** Variables ***** : Ici on définit(initialise) les variables. On peut avoir des scalaires, des listes ou des dictionnaires.
Les variables définies dans un autre fichier sont importées dans la section *****Settings*****.
- ***** Keywords ***** : c'est dans cette section que se trouve la définition des mots clés du test.
Les mots clés sont utilisés pour créer de nouveaux mots clés de niveau supérieur en combinant des mots clés existants. Chaque ligne sous le nom de mot clé doit être indentée pour être considérée comme une instruction.
La syntaxe de création de mots clés utilisateur est très proche de la syntaxe de création de cas de test.
- La façon la plus simple de spécifier des arguments (en dehors de ne pas les avoir du tout) est d'utiliser uniquement des arguments positionnels.

```
*** Keywords ***
```

```
One Argument
```

```
[Arguments]          ${arg_name}
Log      Got argument ${arg_name}
```

```
Two Arguments
```

```
[Arguments]    ${arg1}    ${arg2}
Log      1st argument: ${arg1}
Log      2nd argument: ${arg2}
```

La syntaxe est telle que le paramètre [Arguments] est d'abord donné, puis les noms des arguments sont définis dans les cellules suivantes. Chaque argument se trouve dans sa propre cellule. Le mot-clé doit être utilisé avec autant d'arguments qu'il y a de noms d'arguments dans sa signature.

- Robot Framework propose une autre approche pour transmettre

TABLE 2.2 – Passage d’arguments dans RFW

Exemple de mot clé Utilisateur I			
Mot clé	Action	Argument	Argument
Open Login Page	Open Browser	http ://host/login.html	
	Title Should Be	Login Page	
Title Should Start With	[Arguments]	`\${expected}`	
	`\${title}`=	Get Title	
	Should Start With	`\${title}`	`\${expected}`

TABLE 2.3 – Passage d’arguments dans RFW II

Exemple de mot clé avec des arguments embarqués			
Select `\${animal}` from list	Select Item From List	animal_list	`\${animal}`

des arguments aux mots clés utilisateur que de les spécifier dans les cellules après le nom du mot clé. Cette méthode est basée sur l’incorporation des arguments directement dans le nom du mot clé, et son principal avantage est de faciliter l’utilisation de phrases réelles et claires comme mots clés.

Ces types de mots clés sont utilisés de la même manière que les autres mots clés, sauf que les espaces et les tiret du huit sont interprétés dans leurs noms. Ils ne sont cependant pas sensibles à la casse comme les autres mots clés.

Les arguments incorporés ne prennent pas en charge les valeurs par défaut ou le nombre variable d’arguments comme le font les arguments normaux.



Une partie délicate, décrite ici de l’utilisation d’arguments incorporés consiste à s’assurer que les valeurs utilisées lors de l’appel du mot clé correspondent aux arguments corrects. Il peut y avoir un amalgame entre deux mot clés, lorsque le mot clé avec argument incorporé, coïncide avec la définition d’un autre mot clé (une fois l’argument évalué).

- ***** Test Cases ***** : Ici on va écrire le ou les scripts de tests qui consistent en une sequence de mot clé. Cette partie comportent entre autre des mots clés entourés par deux crochets :
 - [Tags] Fournissent beaucoup de fonctionnalités, comme par exemple des statistiques sur les résultats de tests, les options de lancement des tests, i.e, include/exclude.

- [Setup] qui permet de configurer l'état initial avant le test.
- [Teardown] effectuer le nettoyage une fois le test terminé.
- [Template] qui sont des mots clés spéciaux pour réaliser des tests pilotés par des données.

2.3 Installation et configuration de RFW

Tout d'abord avec une version à jour de pip³,

```
python -m pip install --upgrade pip
```

il faut récupérer une version de python à partir de la version 2.7, ajouter le repertoire où il est installé, et le repertoire des scripts dans le PATH.

Ensuite on installe la dernière version de RFW,


```
pip install robotframework
```

ou une version spécifique :

```
pip install robotframework==3.1.2
```


Installation de la L'IDE natif :

```
pip install robotframework-ride
```

Ci-dessous la procédure d'installation quelque packages interressants en python dont pandas  :

```
python -m pip install pyautogui
python -m pip install pandas
python -m pip install requests
python -m pip install selenium
python -m pip install robotframework-seleniumlibrary
python -m pip install robotframework-faker
python -m pip install xlrd
python -m pip install urllib3
```


Se referer à la documentation officielle de chacun de ces parckages pour plus de détails <https://pypi.org/>.

 RFW est sensible à la casse, et deux espaces minimum sont nécessaires pour séparer les mots clés des arguments. Le nombre maximum de mot-clé imbriqués est de 42⁴, Il est là pour éviter que l'exécution ne plante en raison d'erreurs de récursivité. L'ordre dans lequel les sections

3. c'est toujours utile de metre à jour pip, On peut configurer le gestionnaire de paquets pip si nécessaire Pour Win et linux resp. : "%APPDATA%\pip\pip.ini", "\$HOME/.config/pip/pip.conf"

4. à la date d'écriture de ce document : février 2020

sont écrites n'est pas important. la syntaxe de RFW s'intègre avec les pas de tests Gherkins⁵.

Le Gherkin est une syntaxe utilisée à la base dans l'outil Cucumber  pour faire du BDD ("Behaviour-Driven Development") en écrivant des scénarios de test avec une structure plus humainement lisible et compréhensible. Gherkins utilise un ensemble de mots clés spéciaux : Given, When, Then, But , And et But.

- *Given* un mot clef qui exécute une precondition,
- *When* un mot clef qui effectue une action determinante,
- *Then* un mot clef qui attend une certaine sortie en cas de bon fonctionnement du produit.
- *And* permet de chainer et ou de décomposer les actions et les étapes du test.
- *But* permet de rendre le test plus fluide à la lecture en ajouter par exemple un résultat qui ne devrait pas se réaliser.

2.4 Les librairies

Les bibliothèques de test fournissent les capacités de test réelles à Robot Framework en fournissant des mots clés. Il existe plusieurs bibliothèques standards intégrées et une multitude de bibliothèques externes développées séparément qui peuvent être installées en fonction des besoins.

Une fonction écrite en python peut être appelée en tant que mot-clé dans un fichier robot en remplaçant lors de son appel les "_" par des espaces.

RFW supporte aussi l'interprétation de cmd python inlinées. On utilise le mot clé « Library » pour importer les librairies natives

```
*** Settings ***
Library      BuiltIn
```

ou externes.

```
*** Settings ***
Library      myproject.MyLib          WITH NAME      MyLib
```

Plus loin dans le code nous pourrions accéder aux différents keywords ainsi :

```
*** Keywords ***
My Keyword
    MyLib.Mon Keyword      argument
```

Toutes les bibliothèques de test implémentées en tant que classes peuvent accepter des arguments. Ces arguments sont spécifiés dans la table Setting

5. Inventeurs : Matt Wynne, Aslak Hellesoy & Co.

après le nom de la bibliothèque et lorsque Robot Framework crée une instance de la bibliothèque importée, il les transmet à son constructeur. Les bibliothèques implémentées en tant que module ne peuvent pas prendre d'arguments.

Le nombre d'arguments requis par la bibliothèque est le même que le nombre d'arguments acceptés par le constructeur de la bibliothèque.

```
*** Settings ***
Library      MyLibrary      10.0.0.1      8080
```

Appel de

```
from example import Connection

class MyLibrary:

    def __init__(self, host, port=80):
        self._conn = Connection(host, int(port))

    def send_message(self, message):
        self._conn.send(message)
```

Les bibliothèques implémentées en tant que classes peuvent avoir un état interne, qui peut être modifié par des mots-clés et des arguments au constructeur de la bibliothèque.

Robot Framework tente de garder les cas de test indépendants les uns des autres : par défaut, il crée de nouvelles instances de bibliothèques de test pour chaque cas de test. Cependant, ce comportement n'est pas toujours souhaitable, car les cas de test doivent parfois pouvoir partager un état commun.

Les bibliothèques de test peuvent contrôler la création de nouvelles bibliothèques avec un attribut de classe `ROBOT_LIBRARY_SCOPE`.

Cet attribut doit être une chaîne et il peut avoir les trois valeurs suivantes :

- CAS DE TEST Une nouvelle instance est créée pour chaque scénario de test.
- TEST SUITE Une nouvelle instance est créée pour chaque suite de tests.
- GLOBAL Une seule instance est créée pendant toute l'exécution du test et elle est partagée par tous les cas de test et suites de tests.

2.5 Options pour lancer RFW

TABLE 2.4 – Quelques Options de RFW

version longue	version courte	Description
-variable	-v	fixe la valeur d'une variable lors du lancement du test
-Variablefile	-V	passer un fichier de variables
-Outputdir	-d	spécifie le dossier des fichiers de sortie
-log, -report, -output	-l, -r, et -o resp.	chemins des log, report.html, ou output
-include, -exclude	-i, -e	Filtre les tests à exécuter par rapport à leurs tags
-logtitle, -reporttitle		Précisent le nom du fichier log.html ou report.html
-test, -suite	-t, -s	Précisent quel test ou test suite on désire lancer
-loglevel	-L	TRACE, DEBUG, INFO, WARN, ERROR and NONE
-critical -non critical	-c, -n	Renseignent quels tags font faire échouer les tests

meme séquences de mots clés vs différents types de données à l'entrée

2.6 Les variables automatiques

Robot Framework a ses propres variables qui peuvent être utilisées comme scalaires, listes ou dictionnaires en utilisant la syntaxe `${SCALAR}`, `@{LIST}` et `&{DICT}`, respectivement. De plus, les variables d'environnement peuvent être utilisées directement avec la syntaxe `%{ENV_VAR}`. Les variables pouvant être définies à partir de la ligne de commande au démarrage des tests, il est facile de modifier les variables spécifiques au système.

Les variables Robot Framework, de la même manière que les mots-clés, ne respectent pas la casse, et les espaces et les traits de soulignement sont également ignorés.

Les variables automatiques

TABLE 2.5 – Les variables automatiques de RF

Nom de la variable	Description	Visibilité et portée
<code>{TEST NAME}</code>	nom du test en cours	Cas de test
<code>{TEST TAGS}</code>	tags du cas de test en cours	Cas de test
<code>\$(TEST DOCUMENTATION)</code>	documentation du cas de test	Cas de test
<code>\$(TEST STATUS)</code>	statut du présent test PASS ou FAIL	teardown du test
<code>\$(TEST MESSAGE)</code>	message du test courant.	teardown du test
<code>\$(PREV TEST NAME)</code>	nom du test précédent ou la chaîne vide	partout
<code>\$(PREV TEST STATUS)</code>	statut du test précédent PASS or FAIL	partout
<code>\$(PREV TEST MESSAGE)</code>	Erreur plausible du précédent test	partout
<code>\$(SUITE NAME)</code>	le nom complet de la suite de test	partout
<code>\$(SUITE SOURCE)</code>	chemin absolu du fichier ou repertoire de la suite de test	Partout
<code>\$(SUITE DOCUMENTATION)</code>	Documentation de la suite de test courant	partout
<code>&\$(SUITE METADATA)</code>	Les métadonnées de la suite de test courante	partout
<code>\$(SUITE STATUS)</code>	statut de la suite de test courante FAIL ou PASS	Teardown de la suite
<code>\$(SUITE MESSAGE)</code>	Le message complet de la suite de test courante	Teardown de la suite
<code>\$(KEYWORD STATUS)</code>	le statut de du mot clé courant : PASS ou FAIL	teardown utilisateur
<code>\$(KEYWORD MESSAGE)</code>	Le message possible du mot clé courant	Teardown du mot clé
<code>\$(LOG LEVEL)</code>	Niveau de logging courant	partout
<code>\$(OUTPUT FILE)</code>	chemin absolu au fichier d'output	partout
<code>\$(LOG FILE)</code>	chemin absolu au fichier de log	partout
<code>\$(REPORT FILE)</code>	chemin absolu au fichier de report	partout
<code>\$(DEBUG FILE)</code>	chemin absolu au fichier de debug	partout
<code>\$(OUTPUT DIR)</code>	chemin absolu au repertoire des fichier de sortie	partout

2.7 Structure conditionnelle

Sur robot FW la structure conditionnelle est réalisée grace au mot clé

```
Run keyword if ${value} == "une certaine donne" Mot cle\\
ELSE IF ... condition (mutliple)\\
ELSE
```

2.8 Les boucles for

Les boucles for dans RFW ne peuvent être imbriquées de façon explicite.

La nouvelle syntaxe permet de parcourir les elements d'une liste :

```
— FOR  ${element} IN  @{List of element}
      Appel d un mot clef
      END
```

ou d'une range de valeur :

```
— FOR  ${INDEX}  IN RANGE  0  500
      Appel d un mot clef
      END
```

On peut parcourir deux liste de façons simultanée avec FOR IN ZIP :

```
— FOR  ${elm1}  ${elmt2}  IN ZIP ${List 1}  ${List 2}
      Appel d un mot clef  ${elm1}  ${elm2}
      END
```

— Parfois, il est utile de parcourir une liste et de garder une trace de votre emplacement dans la liste. Robot Framework a une syntaxe FOR index ... IN ENUMERATE ... spéciale pour cette situation. Cette syntaxe est dérivée de la fonction enumerate () intégrée de Python.

For-in-enumerate

```
FOR  ${index}  ${item}  IN ENUMERATE  @{LIST}
    My Keyword  ${index}  ${item}
END
```


3

La bibliothèque standard

Elle possède plusieurs modules dont voici quelque exemples :

- *OperatingSystem* est la bibliothèque standard de Robot Framework qui permet d'effectuer diverses tâches liées à l'OS dans le système où RFW est exécuté. Il peut, entre autres, exécuter des `cmd`, créer et supprimer des fichiers et des répertoires, vérifier si des fichiers ou des répertoires existent ou contiennent quelque chose et manipuler les variables d'environnement.
- *Builtin* : La première bibliothèque historique de RFW, elle est importée de façon implicite, fournit les mots clés génériques les plus utilisés. Mots clé : `Log`, `Fail`, `Sleep`, `Run Keyword if`, `Should be Equal`, `Set Global Variable`, `Evaluate`, ...
- *String* est la bibliothèque standard de Robot Framework pour manipuler des chaînes (par exemple, remplacer la chaîne à l'aide d'expressions régulières, fractionner en lignes) et vérifier leur contenu.
- *DateTime* est une bibliothèque standard de Robot Framework qui prend en charge la création et la conversion de valeurs de date et d'heure, ainsi que d'effectuer des calculs simples avec elles (par exemple, soustraire l'heure de la date, ajouter l'heure à l'heure).
- *Collections* fournit toutes les fonctionnalités liées à la gestion des dictionnaires et des listes.
- *Remote Library* : permet aux bibliothèques de test de Robot Framework d'être exécutées en tant que processus externes. Un cas d'utilisation important pour cette bibliothèque est l'exécution de bibliothèques de test sur des machines différentes de celles où RFW lui-même est exécuté.
- D'autres bibliothèques existent aussi comme *Dialogs* qui permet à l'utilisateur d'interagir pendant l'exécution du test, *xml*, *Telnet* pour exécuter des `cmds` sur des serveurs en utilisant le protocole Telnet. *Screenshot* pour les captures d'écran, intéressant surtout en cas de fail pour voir dans quel état est resté le système.

4

Bonnes pratiques de codage

Cette section met en évidence quelque bonnes pratiques pour la rédaction de tests avec RFW pour avoir des tests évolutifs et maintenables. Elle est inspirée de la page et est loin d'être exhaustive.

- **Pour le nommage** adopter des noms pour les tests suites et les tests qui soient explicites, facile à comprendre, cohérent avec le contexte, descriptifs. Favoriser la réutilisation des mots clés en factorisant le plus possible.
- pour le chainage des mots clés ou des Tags

\# Préferer cette syntaxe

```
*** Settings ***
```

```
Suite Setup      Run Keywords
...              Login To System      AND
...              Add User              AND
...              Check Balance
```

#à cette syntaxe:

```
*** Settings ***
```

```
Suite Setup      Login To System, Add User And Check Balance
```

- **Pour la documentation** elle peut-être facultative si le nom du test ou du test suite est assez explicite. Doit généralement contenir des informations de contexte sur l'utilité des tests, sur l'environnement d'exécution mais ne doit pas être trop détaillée sur les tests... Pour ajouter des informations représentées par des paires de données, préférer l' utilisation des metadata (par exemple version : 1.0, Os : Linux)
- **Architecture des suites de test** les tests dans une suite sont reliés entre eux : ils possèdent souvent la même initialisation. On convient de ne pas mettre plus de 10 tests dans une suite pour des besoins de lisibilités à moins que ce soit des tests pilotés par des

données.

- **Logique des tests** : les mots clés utilisateur peuvent contenir une logique de programmation plutôt basique (boucle for, if / else, repeat, imbriquation). La logique complexe se trouve dans les bibliothèques plutôt que dans les mots clés utilisateurs.
- **Les temps d'attente** Les temps d'attente sont un moyen très "fragile" de synchroniser les tests. Les marges de sécurité provoquent en moyenne des temps d'attente trop long. Au lieu de mettre un time out, c'est mieux d'utiliser un mot-clé qui interroge une certaine action.
- C'est mieux d'utiliser le data-driven style pour éviter la répétition des workflows et mieux représenter les business rules avec un codage du Style Gherkins.
- Éviter les dépendances entre les tests et randomiser l'ordre de lancement des tests.

Bibliographie

- [1] Satya Avasarala. *Selenium WebDriver practical guide*. Packt Publishing Ltd, 2014.
- [2] Silvain EVRARD. Non régression, une notion simple mais complexe.
- [3] Pekka Laukkanen. Data-driven and keyword-driven test automation frameworks. *M. Tech thesis, Department of Computer Science and Engineering Software Business and Engineering Institute, Helsinki University of Technology*, 2006.