Workey Selenium Tester

Cet article présente l'installation, la configuration et l'utilisation de la plateforme Workey d'automatisation des tests "workey-selenium-tester".

Installation

- Télécharger la dernière version dans GitBucket: http://jawa.c-log.lan:8087/workey/workey-selenium-tester/releases
- Dézipper le ficher workey-selenium-tester-1.5.1.zip dans un répertoire de votre système.
- Si besoin, importer le fichier des utilisateurs de tests resources\ldif\test-users.ldif dans le LDAP (les tests de validation s'attendent à trouver certains de ces utilisateurs).
- Éditer si besoin les fichiers admin-rest.properties et validation-selenium.properties

Scripts Groovy

Les scripts et WKY de validation Workey se trouvent dans le répertoire tests/validation. Le script tests/validation/000-setup.groovy est particulier car c'est lui qui déploie les WKY, créé les unités organisationnelles et effectue les assignations des acteurs à leur rôles. Il prend ses informations des fichiers tests/validation/000-assignments.json et tests/validation/000-organizational_units.xml

Plusieurs scripts Groovy ont été développés pour automatiser la création de formulaires, ordres et demandes dans SIRA. Ils se trouvent dans le répertoire groovy/. Les scripts utilisés pour tester le processus de blocage-v2 se trouvent dans le répertoire test/blocage-v2.

Les scripts Groovy présents dans le répertoire test/validation/ sont numérotés en essayant de respecter la convention arbitraire :

- 000: Administration (spécial)
- 001 à 599 : Fonctionnalités de base de Workey
- 600 à 899 : Fonctions spéciales / modules / etc.
- 900 à 999 : Connecteurs

Dans l'idéal, à chaque script Groovy doit correspondre un projet Workey associé avec le même nom (seule l'extension change).

Dans les cartons…

Il faudrait faire en sorte (avec une sorte de fichier Manifest) que le script Groovy puisse déployer lui-même le processus et faire les affectations avant de se lancer.

Lancement

Les options de workey-selenium-tester:

```
usage: workey-selenium-tester [options] scripts.groovy...
 -l,--loop <LOOP>         Number of time each groovy script has to be
executed
 -i,--input-file <FILE>   run tests found in FILE
 -t,--type <TYPE>         runner type WORKEY5, WORKEY6 or WORKEY7 (default
WORKEY6)
 -5,--workey5             Set the runner as Workey 5
 -6,--workey6             Set the runner as Workey 6 (default)
 -7,--workey7             Set the runner as Workey 7
 -v,--verbose             Prints more message from Selenium driver
 -h,--help                Prints this help message and exit
```

Pour exécuter un script Groovy:

```
.\bin\workey-selenium-tester.bat .\tests\blocage-v2\test00-treeview.groovy
```

Pour exécuter tous les tests contenus dans un fichier contenant un fichier .groovy par ligne :

```
.\bin\workey-selenium-tester.bat -i tests_workey.list
```

Une fois tous les scripts exécutés, les statistiques de réussite et d’échec sont affichées. Si des erreurs sont survenues, la liste des scripts problématiques est enregistrées dans le fichier “failed-scripts.list“.

Dans PowerShell, pour purger les processus des navigateurs robots :

```
stop-process -name chromedriver
```

Dans les cartons…

Prévoir un GUI pour faciliter le lancement des tests et pouvoir enregistrer le résultat des tests de manière synthétique (CSV, XML ou JSON).

Compilation

- .\gradlew.bat installZip
- .\build\install\workey-selenium-tester\bin\workey-selenium-tester.bat .\tests\blocage-v2\test00-treeview.groovy

ou bien,

- .\gradlew.bat distZip

Écriture de tests Workey Selenium

Configuration

Fichier de Propriétés

Le constructeur du robot Selenium prend en paramètre un chemin vers un fichier de propriétés. Les différentes propriétés possibles sont les suivantes :

- selenium.driver.host : URL de connexion vers le serveur Workey
- selenium.driver.type : Type de navigateur à utiliser: “chrome” (par défaut), “firefox” ou “edge“. Le driver Chrome est celui qui est privilégié.
- selenium.driver.timeout : Temps d’attente en seconde lors de la recherche d’éléments.

Propriétés spécifiques Chrome

- selenium.driver.chrome.profile : Réutilise le profil Chrome au lieu d’en créer un nouveau à chaque exécution. Cela permet notamment de réutiliser des extensions (valeur “true“/”false“, défaut à “false“).
- selenium.driver.chrome.headless : Lance Chrome sans interface graphiqe (valeur “true“/”false“, défaut à “false“).
- selenium.driver.chrome.allow-notifications : Autorise les notifications à s’afficher (valeur “true“/”false“, défaut à “false“).
- selenium.driver.chrome.maximized : Lance Chrome avec la fenêtre maximisée (valeur “true“/”false“, défaut à “false“).
- selenium.driver.chrome.disable-gpu : Lance Chrome sans accélération graphique, permet parfois de corriger des bugs visuels (valeur “true“/”false“, défaut à “false“).
- selenium.driver.chrome.disable-infobars : Désactive l’affichage d’une barre indiquant que Chrome est lancé en mode automatique (valeur “true“/”false“, défaut à “false“).

Propriétés avancées

- selenium.driver.skip-pause : Ignore les pauses manuelles. Utile lorsque les tests doivent être complètement automatiques (valeur “true“/”false“, défaut à “false“).
- screenshot.dir : Répertoire où seront enregistré les copies d’écran (par défaut “output“)
- selenium.proxy : Créé un proxy permettant d’enregistrer les communications au format HAR avec les méthodes runner.newHar(harFile) et runner.saveHar() (valeur “true“/”false“, défaut à “false“).
- selenium.driver.path : Chemin d’accès aux drivers des navigateurs (par défaut “drivers“)
- test.language : Code du langage utilisé pour les données créées par JFairy (par défaut “fr“)
- runner.type : Type de robot à utiliser (valeur “WORKEY_5“, “WORKEY_6” ou “WORKEY_7“)

Déploiement et initialisation des processus

Le script “validation/000-setup.groovy” est un script spécial qui permet de définir les gestionnaires de workflow et de processus, de déployer automatiquement des fichiers WKY, de créer des unités organisationnelles et d’affecter des acteurs à des rôles et à des unités.

Il prend ces informations à partir du fichier “admin-rest.properties” dans lequel est défini un chemin vers un fichier JSON (voir par exemple, “tests/validation/000-assignments.json“)

Dans les cartons…

Cette partie va vraisemblablement être modifiée pour être un peu plus flexible.

Script Groovy

Le “Runner” Workey

L’application dispose de plusieurs robots spécialisés dans différentes versions de Workey. Ils peuvent être forcés avec l’option “--type” de workey-selenium-tester.

- com.workey.selenium.SeleniumScriptRunner : Workey 5 / JBoss
- com.workey.selenium.SiraScriptRunner : Workey 5 / JBoss / Fonctions spéciales SIRA (Gencods, création des formulaires, etc.)
- com.workey.selenium.Workey6ScriptRunner : Workey 6 / Tomcat
- com.workey.selenium.WorkeyStoreScriptRunner : Workey 7 – nouvelle interface/ Tomcat

Principales propriétés

- runner : le “robot” Workey, couche de haut niveau qui sait se connecter à l’application, ouvrir des vues, créer des document, remplir des formulaires, etc
- runner.driver : le moteur Selenium, pour effectuer des actions de bas niveau qui ne seraient pas proposé par le runner.
- runner.logger : instance de logger
- runner.fairy : Générateur de données factices mais vraisemblables (nom de personne, adresses, société, mail, etc). Pour plus d’informations: JFairy.
- runner.textProducer : Générateur de texte aléatoire
- runner.properties : Propriétés chargées lors de l’exécution du script

Principales méthodes Workey

- runner.getCredentials("XXX") : récupère le login/mot de passe à partir des propriétés systèmes “XXX.login” et “XXX.password“. Si ces propriétés n’existent pas, le login “XXX” est utilisé, avec le mot de passe “test“.
- runner.workeyLogin(Credentials) : Connexion à l’application.

```
// Connexion à workey avec l'utilisateur test001
runner.workeyLogin(new Credentials('test001', 'pa$$w0rd'));
```

- runner.workeyLogout() : Déconnexion
- runner.openView(viewName) : Ouverture d’une vue (par son libellé en v6 et v7, par son nom interne en v5).

```
// Ouverture de la vue "A Faire" en v6
runner.openView('A Faire')
```

- runner.openDocumentFromView(documentIdentifier) : Ouverture d’un document depuis une vue. Le paramètre doit être de préférence une chaîne de caractère permettant d’identifier sans ambigüité le document à ouvrir.

```
// Ouverture d'un document depuis la vue courante
runner.openDocumentFromView('DocumentTest-1C5-FKI-5CT')
```

- runner.createNewDocument(documentLabel) : Création d’un nouveau document, en indiquant le libellé avec lequel il apparaît dans le menu de création.

```
runner.createNewDocument('Demande de congés')
runner.createNewDocument('Note de frais [Valideur]')
```

- runner.setFieldValue(fieldName, value): Change la valeur d’un champ.

Attention

Le type de la donnée “value” doit être une chaîne de caractère.

- runner.setFieldValue(fieldName, value, index): Si le champ est multivalué, ou s’il fait partie d’une table dynamique, change la valeur du champ à l’index indiqué (commence à 0)
- runner.getFieldValue(fieldName): Récupère la valeur du champ.
- runner.getFieldValue(fieldName, index): Récupère la valeur du champ à l’index indiqué si le champ est multivalué, ou s’il fait partie d’une table dynamique (commence à 0)

Méthodes utilitaires annexes

- runner.waitForField(fieldName): Le script attend que le champ soit visible dans le document.

Conseil

Cette fonction est utile pour s’assurer qu’un document a terminé d’être chargé.

- runner.randomId(prefix) : Génère une chaîne de caractère aléatoire du type “prefix-ABC-DEF-GHI“. Cela permet notamment de retrouver facilement des documents testés si cette valeur est utilisée dans le sujet du document.
- runner.close() : Ferme la fenêtre active du navigateur.
- runner.quit() : Quitte le navigateur

- runner.newTab() : Création d’un nouvel onglet
- runner.switchTab(): Bascule cyclique d’un onglet à l’autre
- runner.switchToFrame(frame): Sélectionne le <frame> HTML courante. La plupart des méthodes de WorkeyScriptRunner se chargent de basculer d’une frame à l’autre si besoin.
- runner.expectAlert(message) : Si une alerte JS est émise, cette fonction permet de la valider et continuer le test.
- runner.pause(message) : Met le script en pause en affichant un message. L’utilisateur doit appuyer sur Entrée pour continuer le test. Cela permet de donner le temps à l’utilisateur d’effectuer des actions non testable (ouverture et vérification d’un fichier externe, par exemple)
- runner.pause(temps_ms) : Marque une pause dans le traitement du script. Utile pour être sûr que certains traitements aient le temps de s’effectuer (JS, etc).
- runner.screenshot(file): Prend une capture d’écran de la fenêtre courante du navigateur et l’enregistre dans un fichier.

Tests JUnit

Les tests se font avec la classe org.junit.Assert

```
// Vérification de la valeur d'un champ
Assert.assertEquals("Valeur de champ correcte",
    "Valeur Correcte", runner.getFieldValue('Champ_a_tester'));
```

- Assert.assertTrue
- Assert.assertFalse
- Assert.assertEquals
- Assert.assertNotNull

Pour vérifier qu’un objet est visible dans le navigateur:

```
Assert.assertTrue("Element #workey-button visible.",
    driver.findElement(By.id('workey-button')).isDisplayed());
```

Expressions XPath

Pour tester une expression XPath dans Chrome, avec la console:

```
document.evaluate("//input[@id='Field_XXX']",
    document, null, XPathResult.ANY_TYPE, null ).iterateNext()
```

Exemple de script

```
import org.junit.Assert;
import org.openqa.selenium.By;
import com.workey.selenium.WorkeyRunnerFactory;

// Création du robot Workey. Le type (v5, v6 ou v7) est déterminé
// automatiquement.
runner = new WorkeyRunnerFactory('validation-selenium.properties').runner;
```

```
driver = runner.driver;
logger = runner.logger;

// Connexion avec le login et mot de passe définis dans les propiétés
// user1.login et user1.password
runner.workeyLogin(runner.getCredentials('user1'));

// Création d'un numéro de test aléatoire pour retrouver le document
// facilement lors de la suite des tests
testId = runner.randomId('TestSelenium');
logger.info("Création du document \"${testId}\"." );

// Création du nouveau document
runner.createNewDocument('Document TestSelenium')

// Attente du chargement complet du document
runner.waitForField('Id_Test');

// Vérification de l'état et du formulaire
runner.assertFormAndState('Formulaire_TestSelenium', '__CREATED');

// Remplissage des champs
runner.setFieldValue('Id_Test', testId);
runner.setFieldValue('String', 'Valeur de mon champ');

// Sélection de l'état suivant et soumission du document
runner.selectNextState('Cree');
runner.clickSubmitButton();

// Ouverture du document dans la vue (peut parfois nécessiter
// de rafraichir la vue)
runner.openDocumentFromView(testId);

// Vérification de la valeur du champ
runner.waitForField('Id_Test');
Assert.assertEquals('Champ String', 'Valeur de mon champ',
    runner.getFieldValue('String'));

// Sélection du dernier état et soumission
runner.selectNextState('Termine');
runner.clickSubmitButton();

// Déconnexion et fermeture du navigateur
runner.workeyLogout();
runner.quit();
```
