Automatisation de tests via SoapUI

Chantal Zimmer

Mars 2023

onepoint.
beyond the obvious

SoapUI est un logiciel open source permettant les tester les API, mal connu et pourtant très utile pour effectuer des tests techniques et même automatiser des scénarios de bout en bout. Je vous propose une présentation du périmètre d'action du logiciel suivie de l'explication de l'architecture d'un projet d'automatisation pour finir par deux exemples concrets d'utilisation.

Périmètre d'action du logiciel SoapUI

Le logiciel SoapUI permet de faire bien plus que de simples appels de web services SOAP ou REST.

Son périmètre d'actions comprend notamment:

- Appel de web services SOAP,
- Appel de web services REST,
- Requête en base de données,
- Transfert de variables,
- Script en langage Groovy,
- Assertion permettant de vérifier les données.

Cette liste est non exhaustive, personnellement j'ai encore beaucoup de fonctionnalités à expérimenter comme:

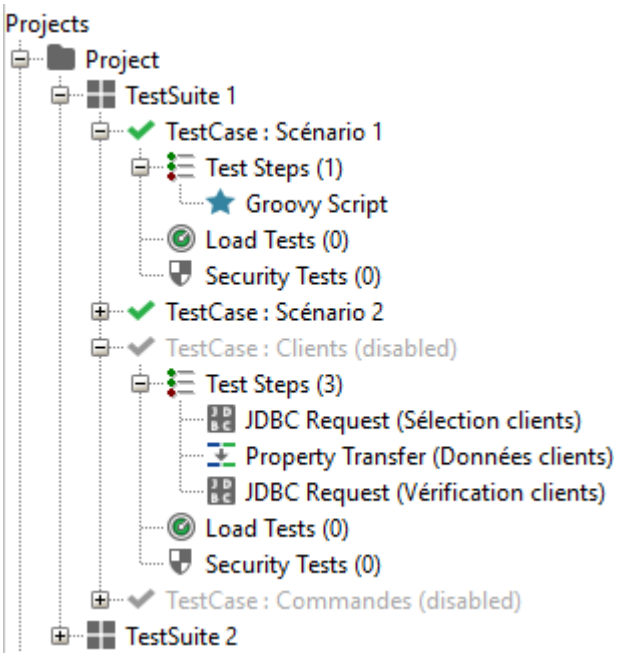
- Requête HTTP,
- Requête AMF,
- Conditional Goto,
- Protocole de messagerie MQTT

Architecture du projet

Le projet d'automatisation pour être facilement maintenable se doit d'avoir une architecture claire et optimisée ainsi que des données variabilisées.

a) Architecture

Pour nos projets, le choix s'est porté sur une architecture en briques. Cette organisation permet d'éviter au maximum la duplication des services, requêtes et scripts utilisés.



Le projet est composé d'un ou plusieurs TestSuite qui représentent le périmètre à tester.
A l'intérieur du TestSuite, chaque TestCase représente une brique. Nous avons des briques actives et inactives:

- ✓ Les briques actives correspondent aux scénarios à exécuter.
- ✓ Les briques inactives regroupent par type de fonctionnalité les différents appels, requêtes et scripts utilisés par les scénarios.

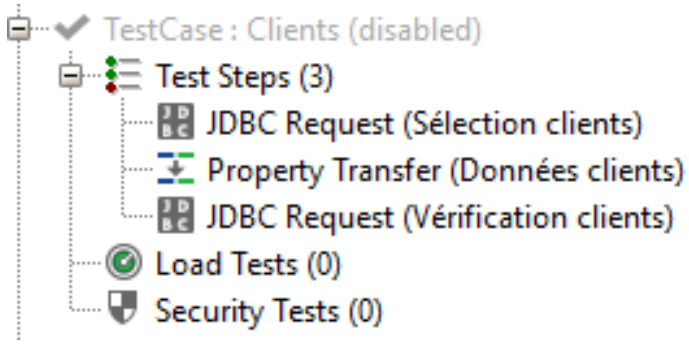
Chaque brique active comprend un seul TestStep qui est un script Groovy permettant d'exécuter les différentes étapes du scénario.

Groovy Script

```
def project = testRunner.testCase.testSuite.project

//////////////////////////////Sélection de clients....//////////////////////////////
testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Sélection clients)'] )
testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['Property Transfer (Données clients)'] )
testRunner.runTestStep( project.testSuites['TestSuite 1'].testCases['TestCase : Clients'].testSteps['JDBC Request (Vérification clients)'] )
```

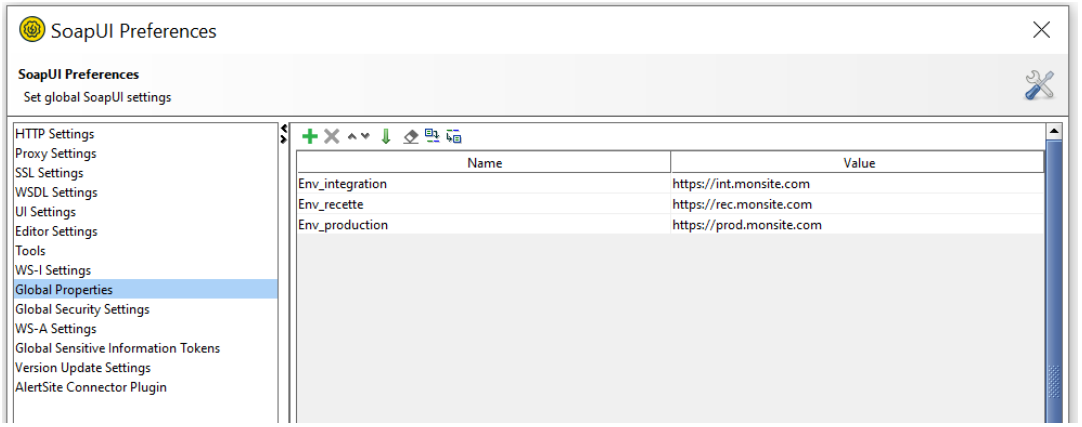
Chaque brique inactive comprend un ou plusieurs pas de test.



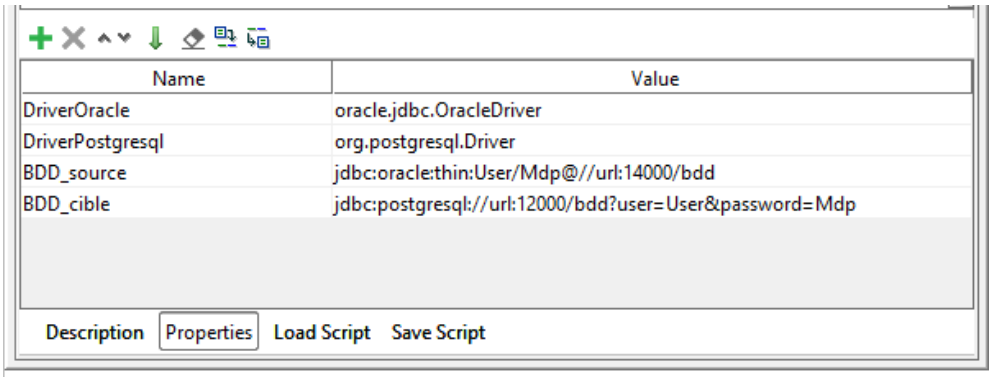
b) Variabilisation

Au niveau de la variabilisation des données, nous avons opté pour l'enregistrement des variables à différents niveaux.

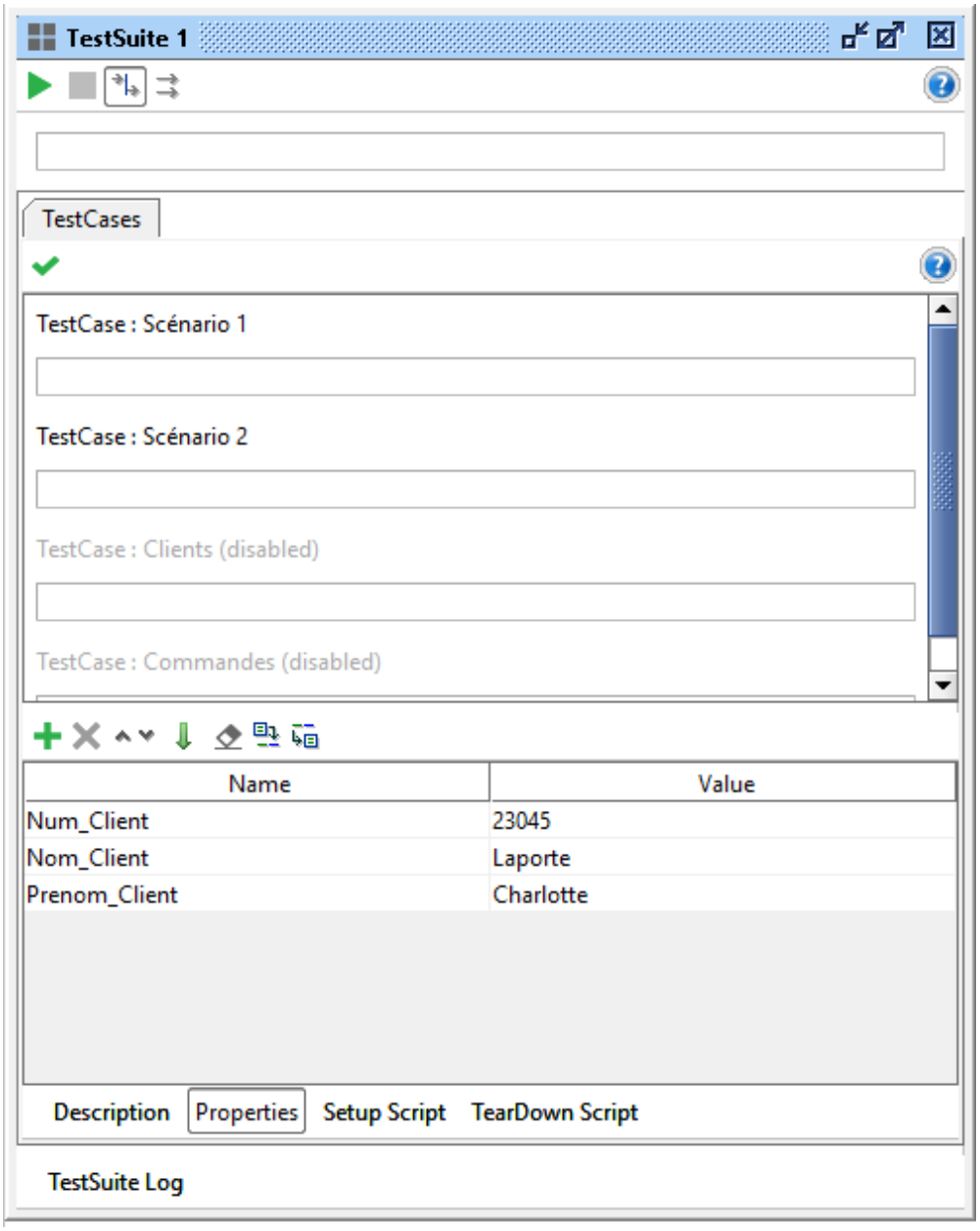
Les données communes à différents projets comme les url des différents environnements sur lesquels les tests doivent être exécutées sont stockées au niveau « Global Properties »



Les données communes à plusieurs périmètres comme les url de connexion aux bases de données ou les url des différents services se trouvent au niveau des « Properties du Project »



Les données sélectionnées via les requêtes ou récupérées via les services, sont enregistrées au niveau des « Properties du TestSuite » pour être accessibles depuis tous les TestCase.



Exemples concrets:

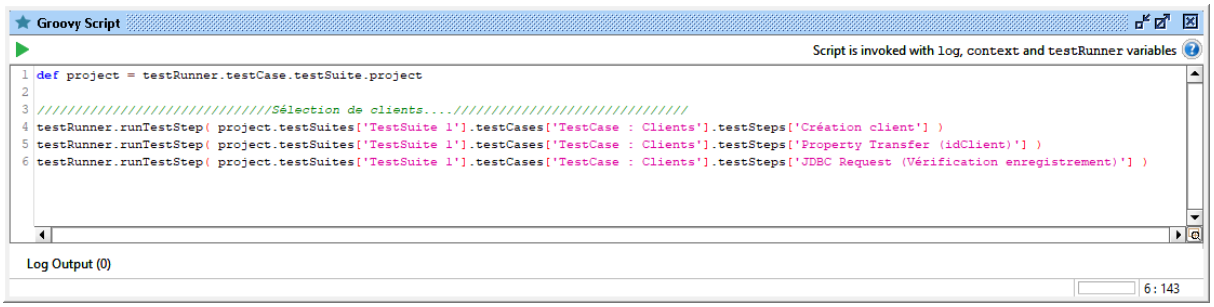
a) Automatisation de TNR

Prenons l'exemple de l'automatisation de 2 services REST pour compléter nos TNR.

1. Un service de création de client
2. Un service de recherche de commandes par idClient

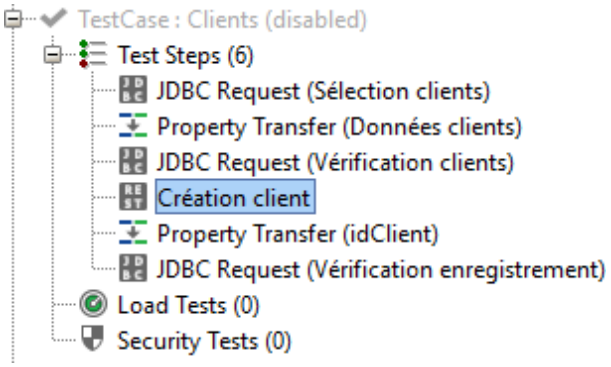
Dans notre 1er scénario, nous allons mettre en place le script groovy permettant d'appeler :

- Le service de création de client
- Le stockage de l'idClient renvoyé par le service
- Une requête en BDD permettant de vérifier l'enregistrement

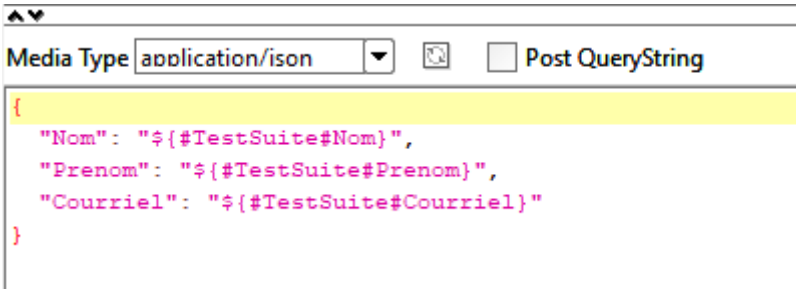


Dans notre brique inactive « Clients », nous allons ajouter :

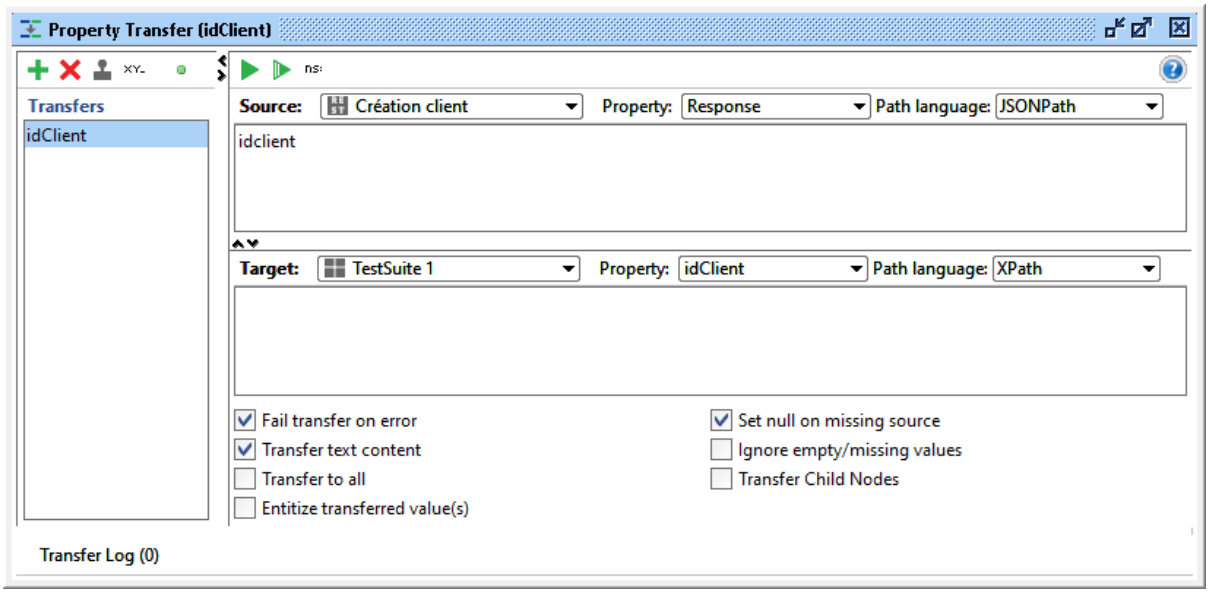
- Le service de création de client



Les données en entrée du service REST sont variabilisées et stockées dans le TestSuite:

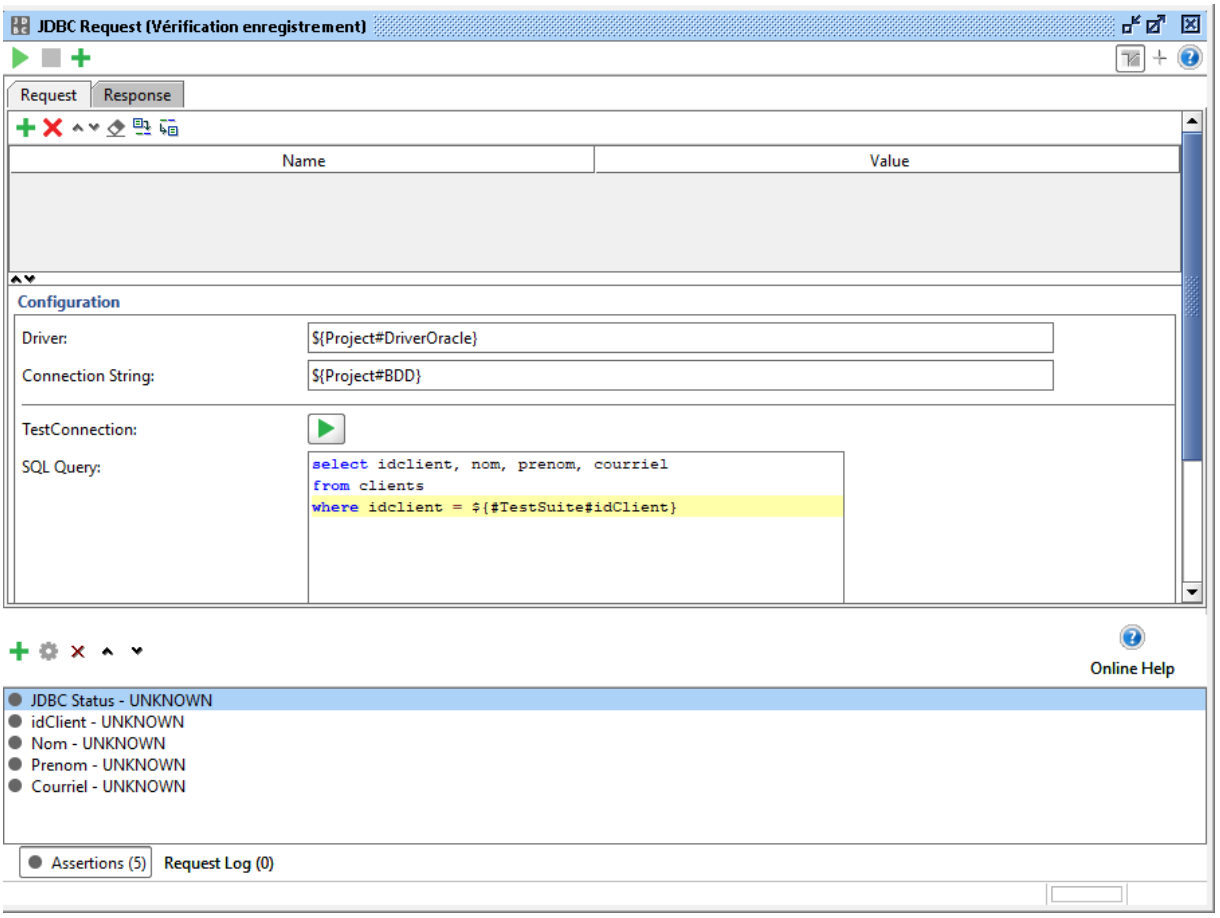


- Un step permettant de stocker l'idClient

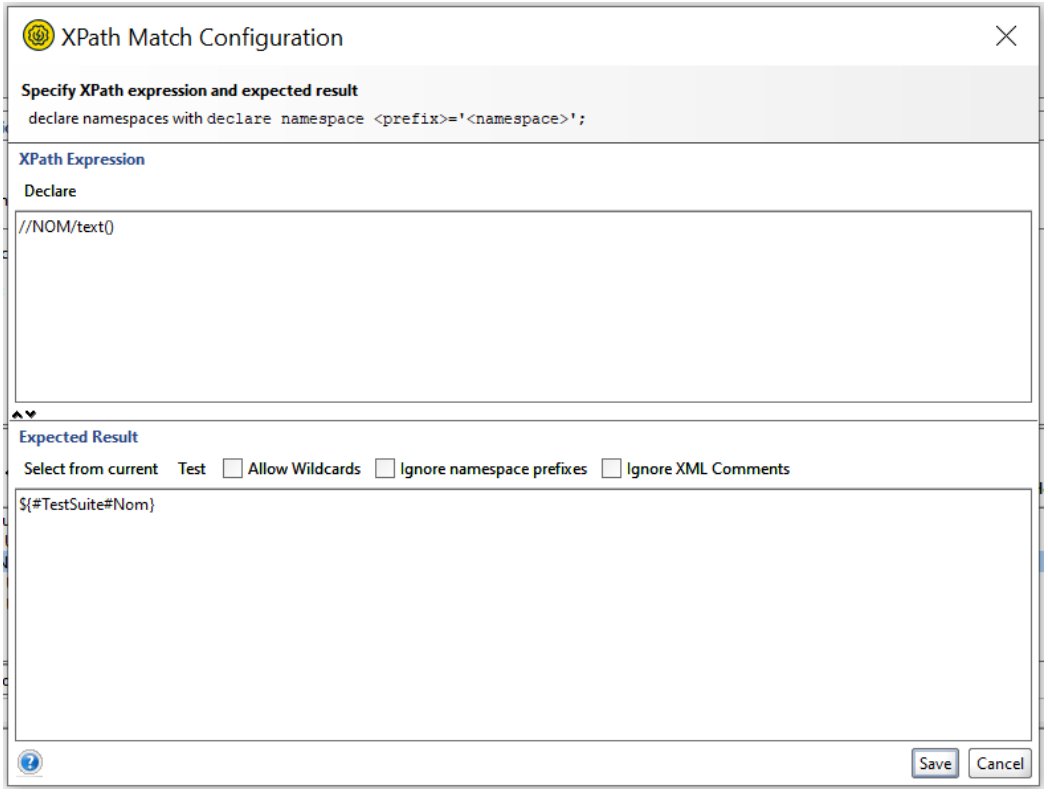


A l'aide de ce step, l'idClient est récupéré dans le retour du service REST et stocké dans le TestSuite.

- La requête permettant de récupérer les informations d'un client

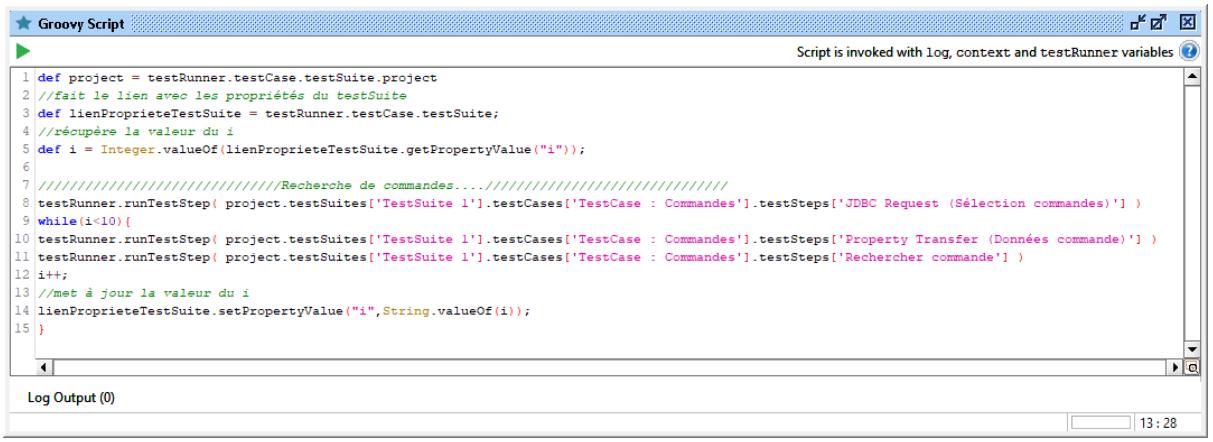


Chaque information récupérée est vérifiée au sein des assertions, sa valeur doit être identique à celle stockée dans le TestSuite:



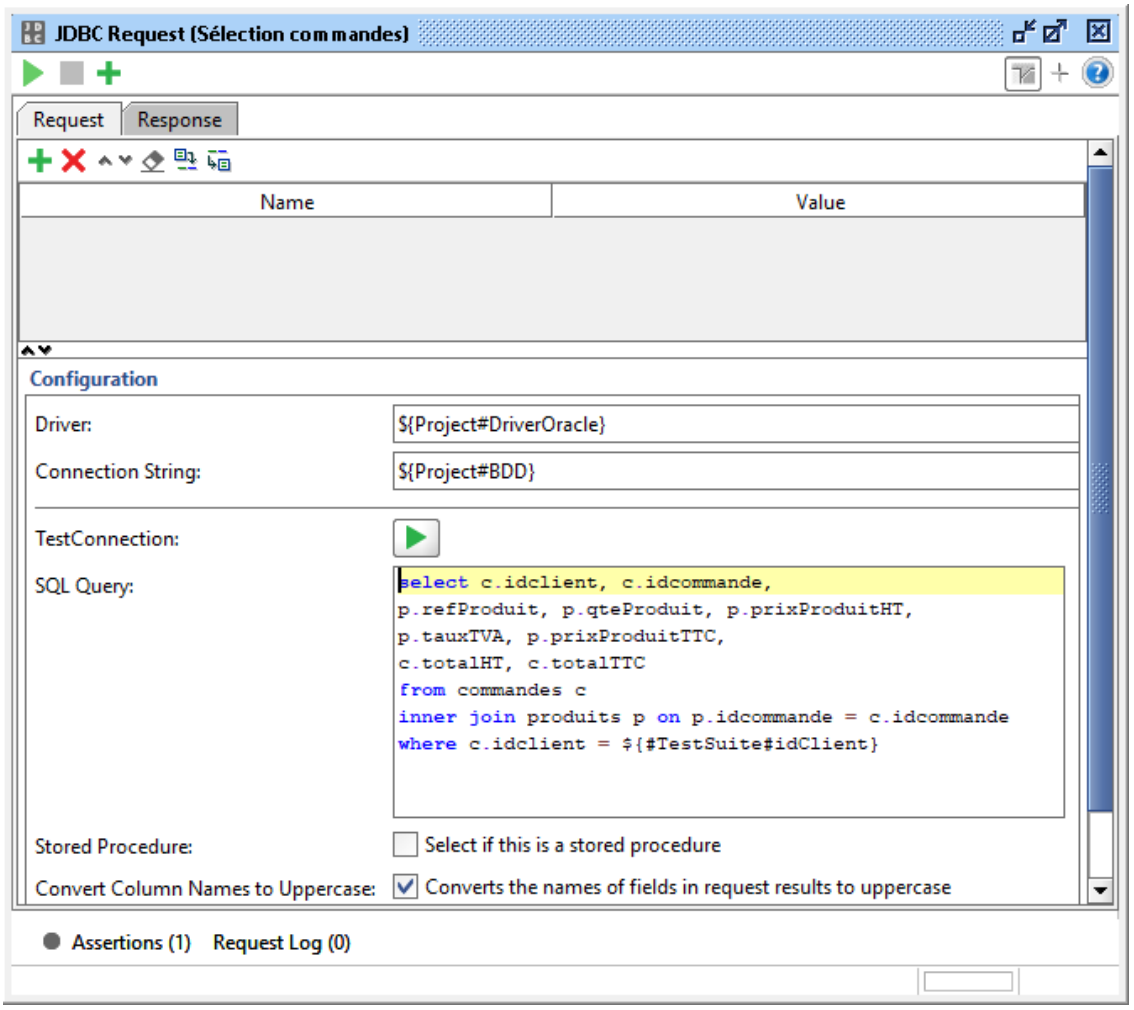
Dans notre 2ème scénario, nous allons mettre en place le script groovy permettant d'appeler:

- Une requête en BDD permettant de récupérer les commandes d'un client
- Le stockage des données d'une commande
- Le service de recherche de commandes par idCommande

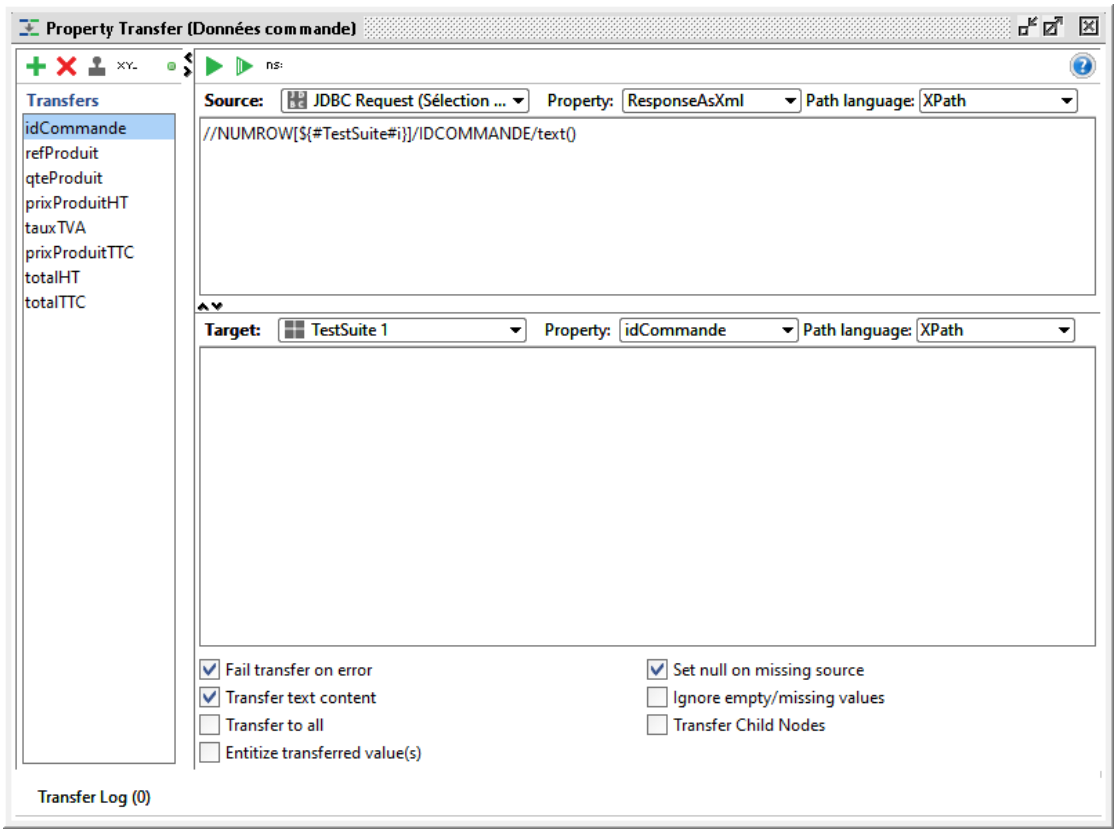


Dans notre brique inactive « Commandes », nous allons ajouter:

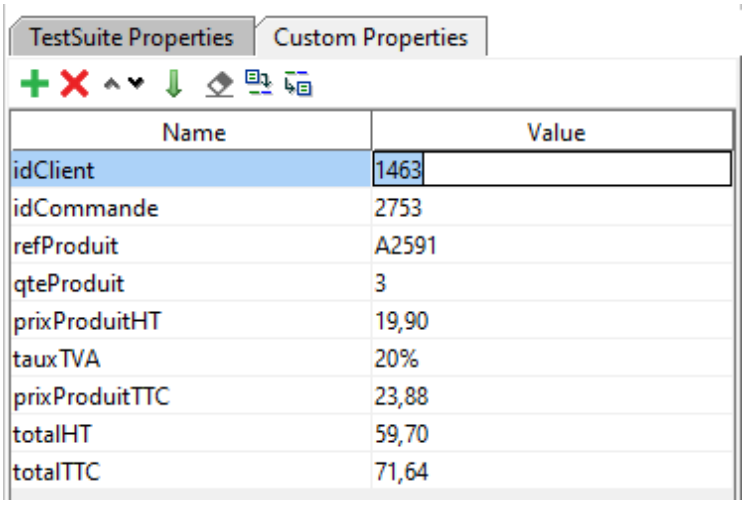
- La requête permettant de récupérer les commandes d'un client



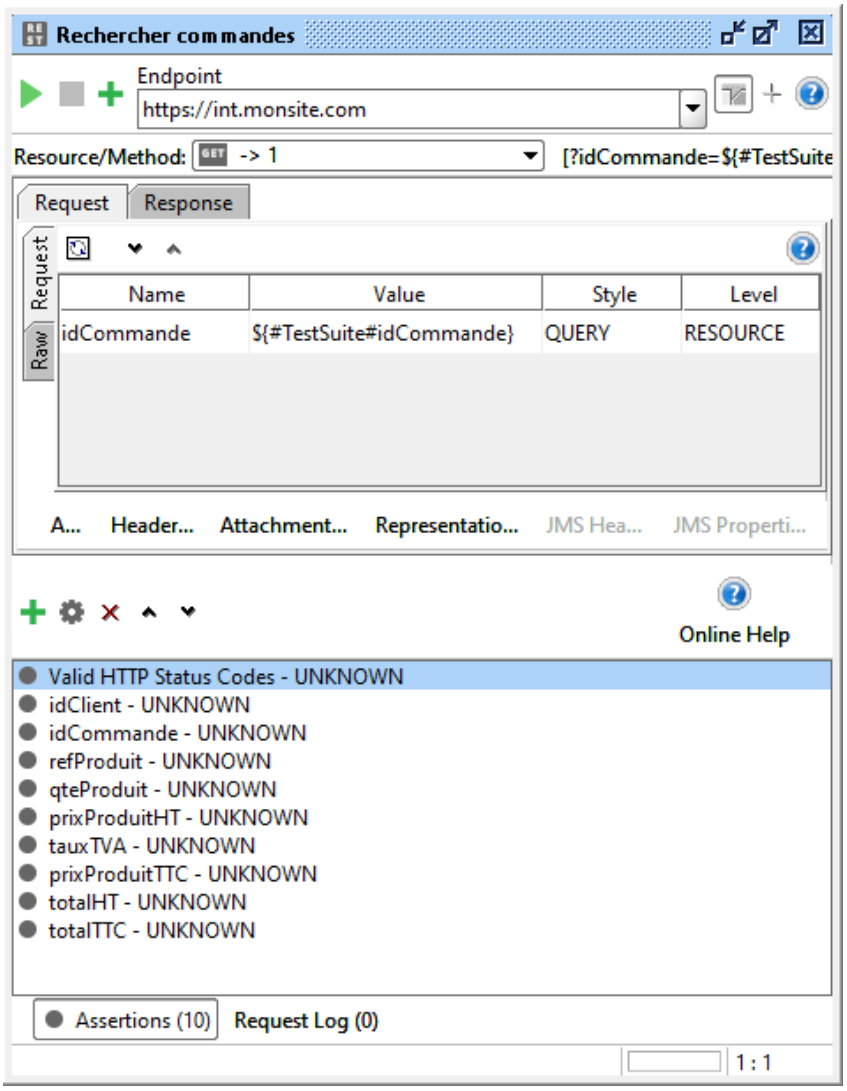
- Un step permettant de stocker les informations d'une commande



On stocke les données de la 1ère commande pour les comparer à celles de la recherche.



- Le service de recherche de commande par idCommande



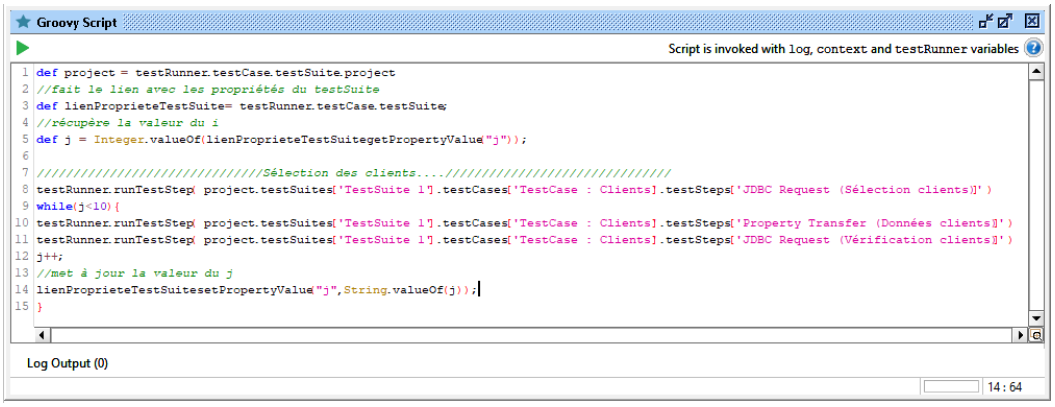
b) Automatisation des tests d'une migration de BDD

Prenons l'exemple de la migration de notre BDD contenant :

1. La table « Clients »
2. La table « Commandes »

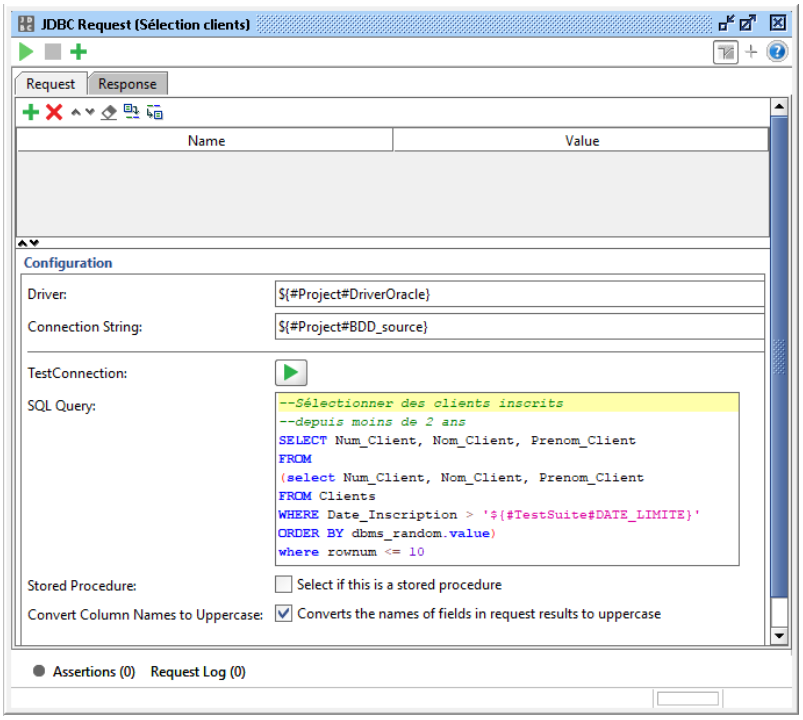
Dans notre 1er scénario, nous allons mettre en place le script groovy permettant d'appeler:

- La requête permettant de sélectionner les clients du périmètre à extraire dans la BDD source
- Le step permettant de stocker les données de chaque client
- La requête permettant de vérifier que les clients sont bien enregistrés dans la BDD cible

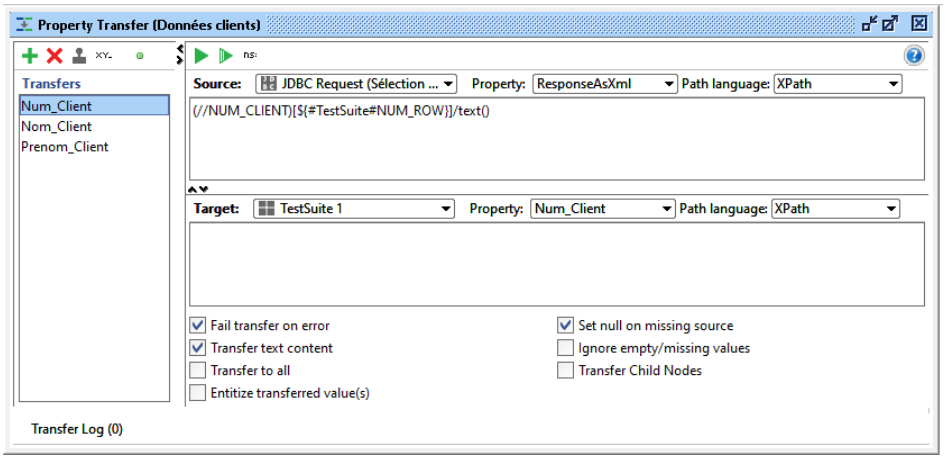


Dans notre brique inactive « Clients », nous allons ajouter:

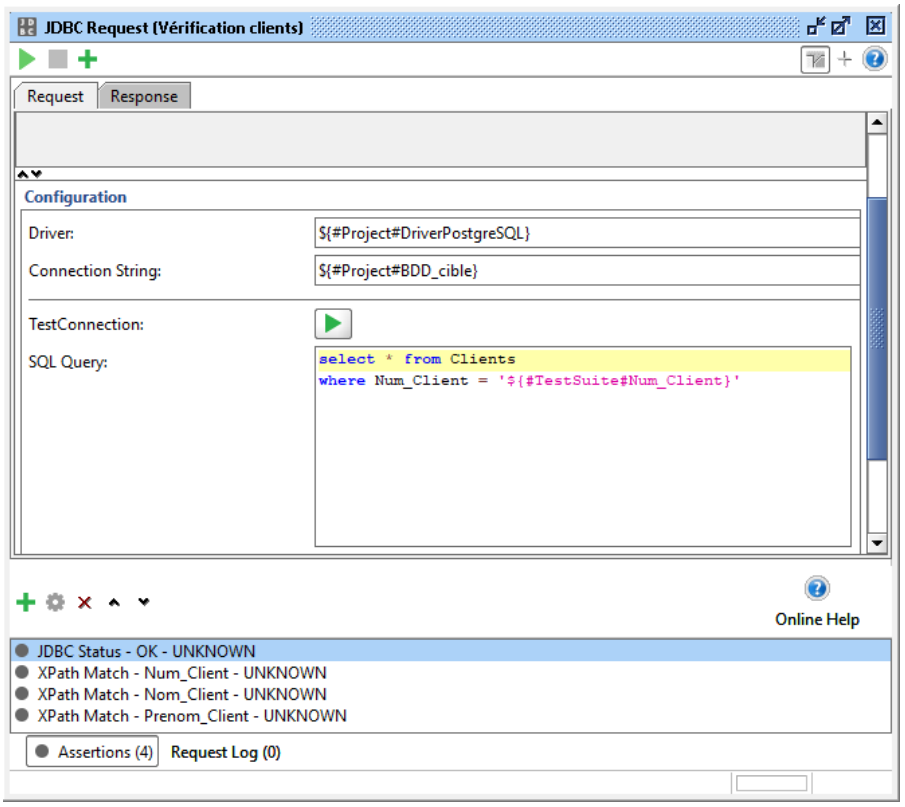
- La requête permettant de sélectionner les clients dans la BDD source



- Le step permettant de stocker les données de chaque client



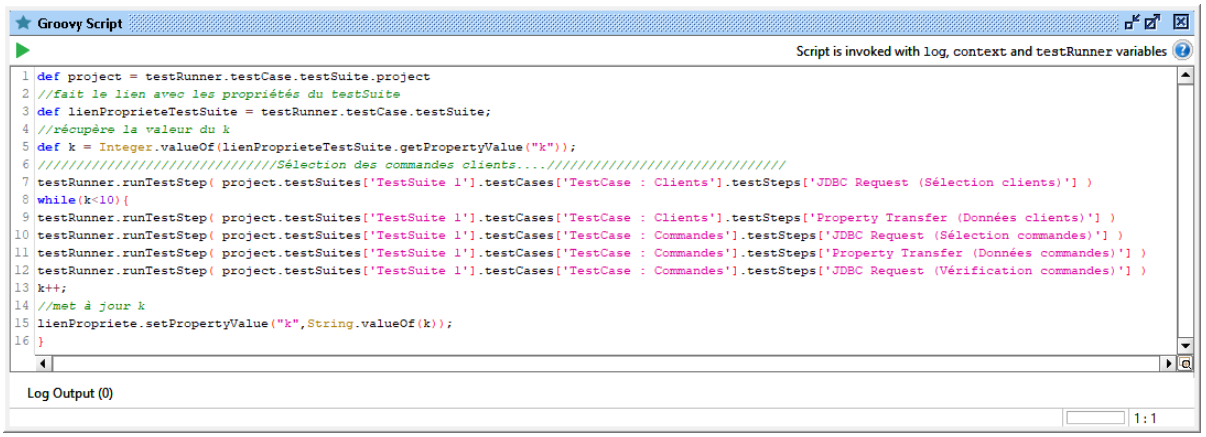
- La requête permettant de vérifier l'enregistrement des clients dans la BDD cible



Pour compléter ce scénario, nous pouvons ajouter une requête permettant de vérifier qu'aucun client hors périmètre n'est présent dans la BDD cible.

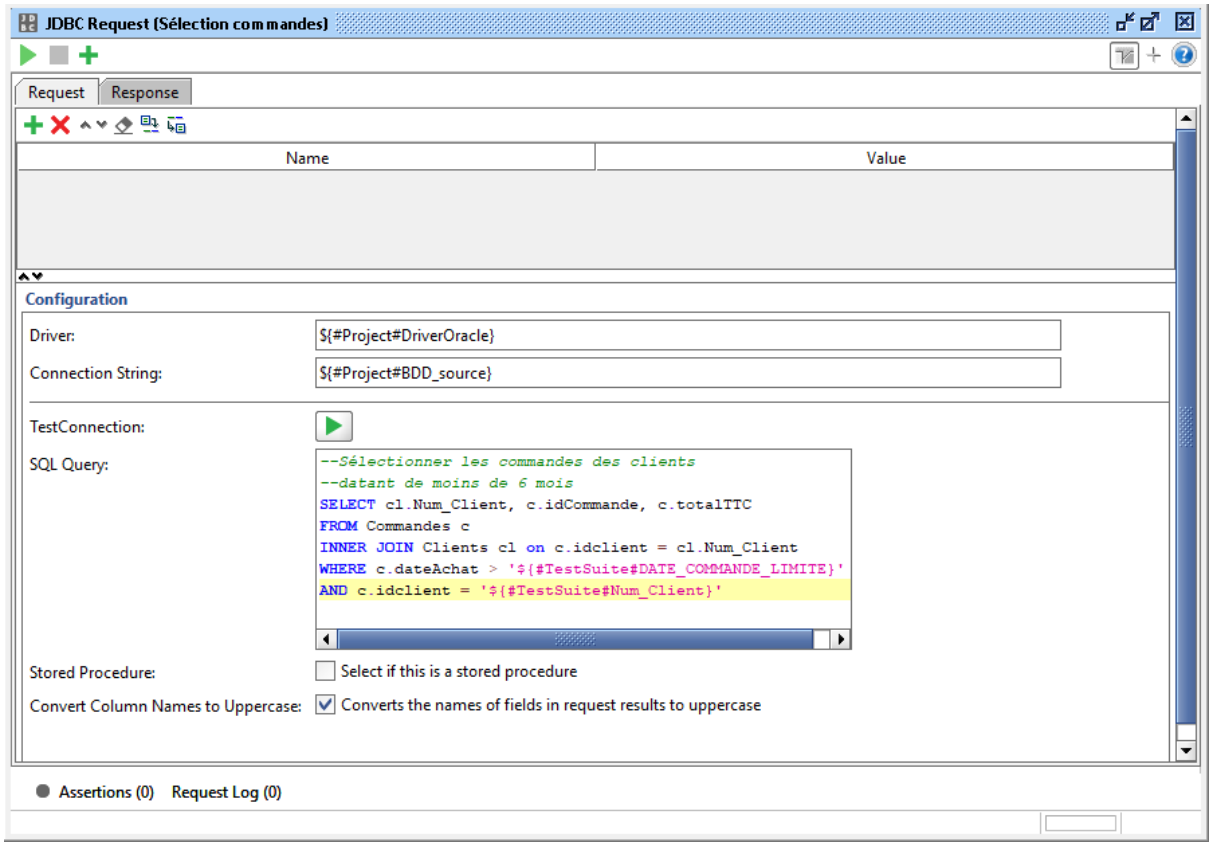
Dans notre 2ème scénario, nous allons mettre en place le script groovy permettant d'appeler:

- La requête permettant de sélectionner les commandes des clients extraits et datant de moins de 6 mois
- Le step permettant de stocker les données de chaque commande
- La requête permettant de vérifier que les commandes sont bien enregistrées dans la BDD cible

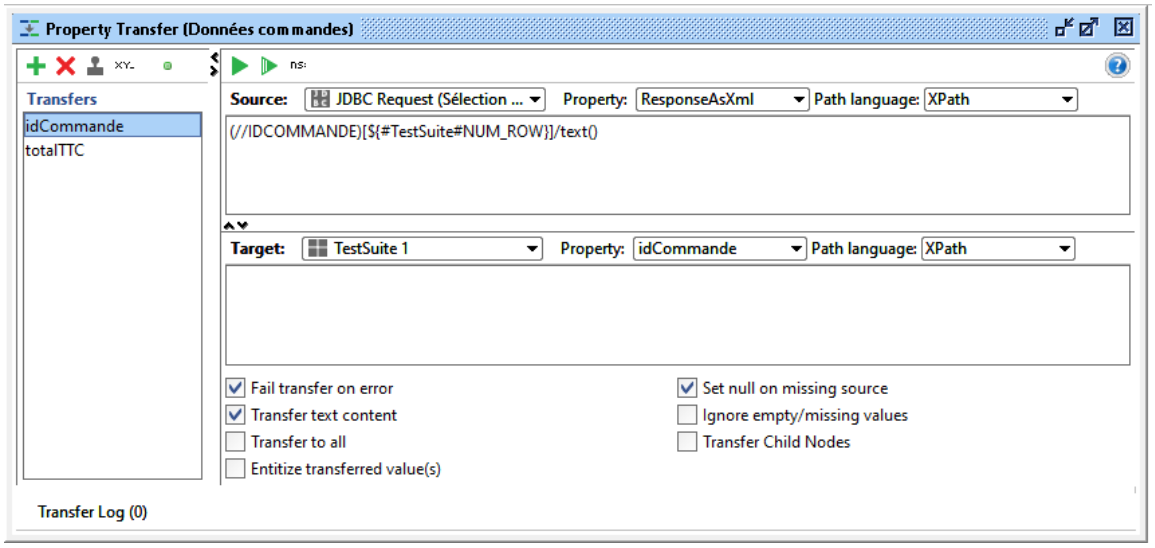


Dans notre brique « Commandes », nous allons ajouter:

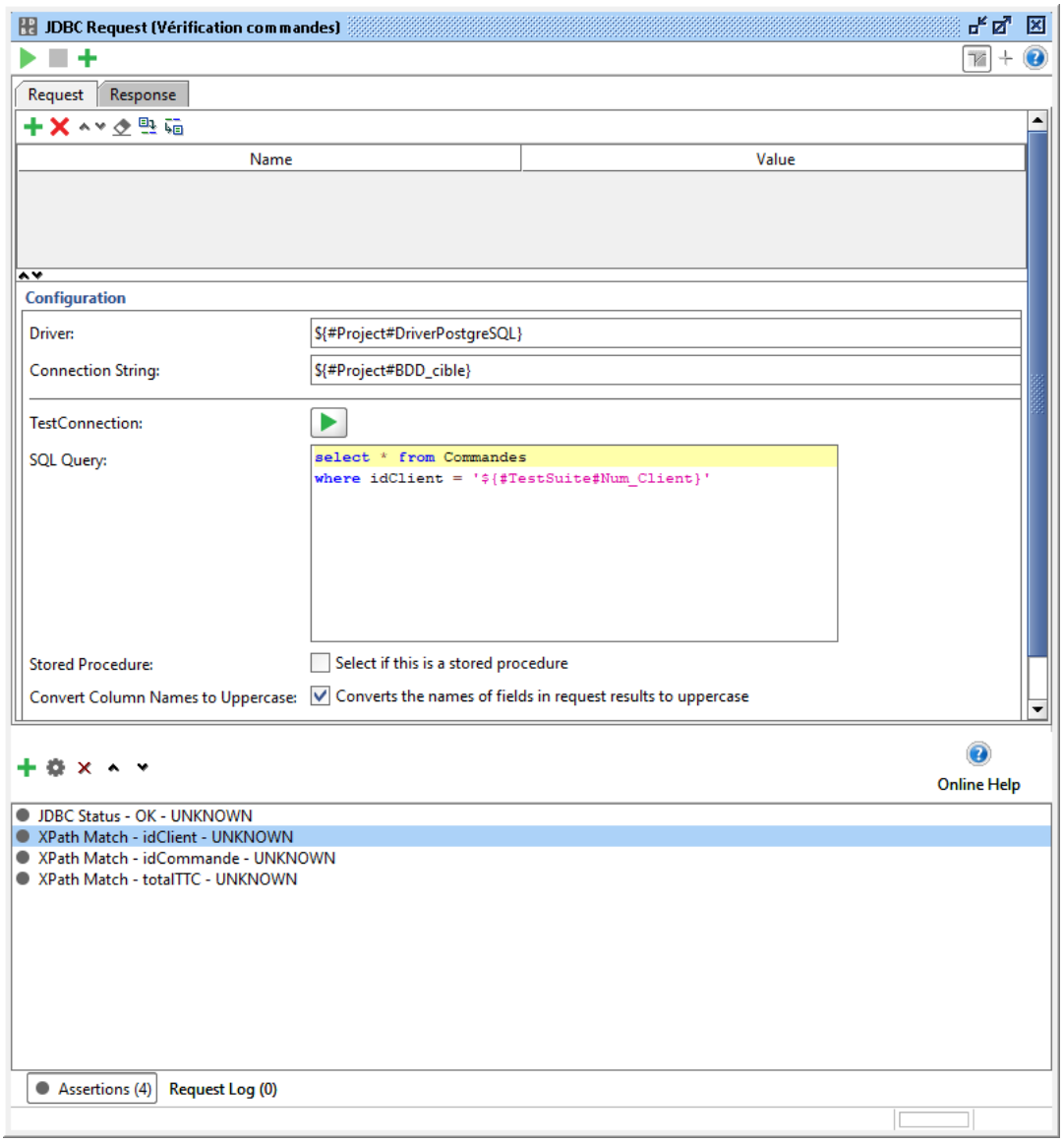
- La requête permettant de sélectionner les commandes dans la BDD source



- Le step permettant de stocker les données de chaque commande



- La requête permettant de vérifier l'enregistrement des commandes dans la BDD cible



Pour compléter ce scénario, nous pouvons ajouter une requête permettant de vérifier qu'aucune commande datant de plus de 6 mois ou n'appartenant pas à un client du périmètre n'est présente dans la BDD cible.

En fonction de la livraison des différents travaux, si les extractions des tables « Clients » et « Commandes » sont livrées au même moment, un seul scénario peut être envisagé.

- Sélectionner les clients de la BDD source
- Stocker les données du 1er client
- Sélectionner les commandes du 1er client
- Stocker les données des commandes du 1er client
- Vérifier l'enregistrement des données du 1er client

- Vérifier l'enregistrement des données des commandes du 1er client
- Boucler pour vérifier l'ensemble des extractions clients/commandes

Amis testeurs, lancez-vous dans l'automatisation avec SoapUI !