SoapUI Автоматизация тестирования

Содержание

SoapUI обладает большими возможностями по автоматизации тестов. Эта статья - дополнение к "основной":
https://rm.inforion.ru/projects/egis/wiki/Тестирование_веб-сервисов_How-To_Guide_от_простого_к_сложному
Здесь будет много скриптов и примеров. Начальные навыки программирования приветствуются, но не обязательны.
Связанная задача: 7742
Полезные примеры всяких скриптов и решений:
Learning SoapUI
Сборник примеров скриптов
NEW Подборка Groovy скриптов на гите

Подготовка, драйверы

Для базовой автоматизации (создания тест-кейсов, запуска подряд несколько запросов и простых скриптов) вам понадобится только сам SoapUI. Для продвинутых операций (подключение к БД, отправка на FTP) понадобится немного пошаманить:
Прежде всего нам понадобится:

- SoapUI 64 bit (гайд для версии 5.3.0). Если вы загружали SoapUI с сайта с большой кнопкой Download - скорее всего у вас версия x32 и лежит она в папке C:\Program Files (x86)\SmartBear. Нам же нужно скачать версию x64 вот здесь: https://www.soapui.org/downloads/latest-release.html
- драйвер JDBC для подключения к **postgreSQL**: https://jdbc.postgresql.org/download.html . Версия 4.2 (файл **postgresql-42.1.1.jar**);
- драйвер JBDC для подключения к **Oracle** приложен к этой статье - **ojdbc8.jar** ((если что, скачивать тут: http://www.oracle.com/technetwork/database/features/jdbc/jdbc-ucp-122-3110062.html , требуется регистрация)
- поставить **Java 1.8 версии 64**: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
- для продвинутых операций типа FTP загрузки нам также понадобится библиотека Apache commons Net (http://commons.apache.org/proper/commons-net/download_net.cgi) - файл **commons-net-3.6.jar** (в архиве commons-net-3.6-bin.zip), jar приложен к этой статье. Файл необходимо положить в папку C:\Program Files\SmartBear\SoapUI-5.3.0\bin\ext (см. ниже)

С версией 32 не пробовала, но возможно - заработает

После этого:

- запустить один раз SoapUI;
- закрыть SoapUI;
- перейти в папку C:\Program Files\SmartBear\SoapUI-5.3.0 и **переименовать папку jre в jre.ignore** (или переместить в другую папку);
- положить скачанный файл драйвера postgresql-42.1.1.jar в папки: C:\Program Files\SmartBear\SoapUI-5.3.0\bin\ext и в C:\Program Files\SmartBear\SoapUI-5.3.0\lib;
- открыть SoapUi. В логе (в нижней панели есть SoapUI Log) должны быть записи вида "Used java version: 1.8.0_121" и "Adding [C:\Program Files\SmartBear\SoapUI-5.3.0\bin\ext\postgresql-42.1.1.jar] to extensions classpath"

Что мы сделали и зачем: по умолчанию SoapUI работает а Java 7, драйвер postgresql-42.1.1.jar дружит с java 8 - в результате скрипты не выполнятся из-за разных версий java. Поэтому мы убрали встроенную в SoapUi java (переименовав папку), заставив его работать с java, которая установлена на компьютере (java 8).

Теперь для oracle:
1) положить файл ojdbc8.jar в папки C:\Program Files\SmartBear\SoapUI-5.3.0\bin\ext и в C:\Program Files\SmartBear\SoapUI-5.3.0\lib;
2) в Oracle должен существовать юзер, с которым можно подключиться к БД (sys подключиться не даст). Создаем пользователя (логинимся в оракл как sysdba, на листе скриптов localoracle:

```
create user testadmin identified by oracle;

grant dba to testadmin;
```

4) строка подключения к oracle, например: jdbc:oracle:thin:testadmin/oracle@192.168.70.90:1521:orcl , драйвер oracle.jdbc.driver.OracleDriver

Connection String для:
CAT(Oracle-192.168.65.47) - jdbc:oracle:thin:TEST3/oracle@192.168.65.47:1521:orcl
Тестовая(Oracle-192.168.70.90) - jdbc:oracle:thin:testadmin/oracle@192.168.70.90:1521:orcl
Тестовая(Postgres-192.168.70.91) - jdbc:postgresql://192.168.70.91:5432/ORA2PG_T?user=postgres&password=postgres
СГК(Oracle-sys2) - jdbc:oracle:thin:TESTADMIN/oracle@10.11.34.12:1521/orcl

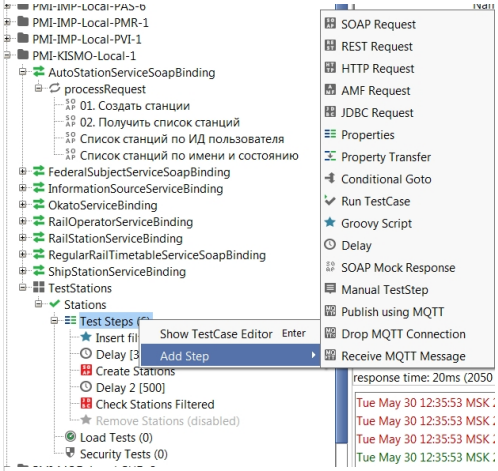
Пример тест-кейса

Ниже описан простой тест-кейс, который состоит из шагов:
1. создать фильтры в БД;
2. выполнить запрос с созданием станций;
3. подключиться к БД и проверить, что отфильтрованы нужные станции, ненужные не отфильтрованы;
4. (дополнительно) очистить базу от тестовых данных.

Создание Тест-кейса

Для примера я буду использовать мой существующий проект для тестирования Кисмо. В проекте уже созданы интерфейсы (AutoStationBindingSoap), запросы параметризированы (см. разделы выше), в свойствах проекта расставлены переменные (ip адрес, creatorId и т.д.). Все это описано в разделах выше, считаю, что с этим уже разобрались.

1. Нажать ПКМ по существующему проекту и выбрать "New Test Suite". Созданный Test Suite появится в дереве проекта, его можно переименовать (ПКМ - Rename). Test Suite - набор тест-кейсов, тест-кейс - набор "шагов". TestSuite можно выполнять целиком (то есть все включенные тест кейсы). Например, можно создать один TestSuit, в котором есть 3 TestCase для каждого типа транспорта;
2. Нажать по созданному TestSuite ПКМ и выбрать New Test Case. TestCase появится в дереве TestSuite. Кейс можно переименовать (ПКМ - Rename);
3. Чтобы создать шаг теста, нужно нажать ПКМ по TestCase и выбрать тип создаваемого шага, например SOAP Request (наш обычный запрос) или JDBC Request:



Шаг: Подключение к БД и выполнение скрипта

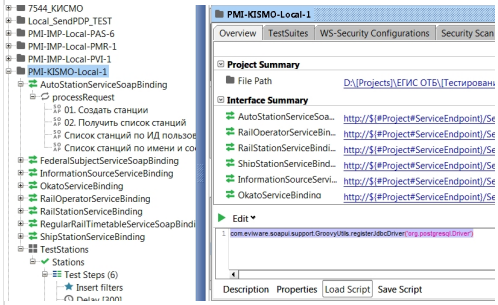
!Предложенные ниже скрипты и решения вероятно не являются самыми оптимальными, возможно все можно упростить и сделать красивше :) Практика, гуглирование и чтение документации в этом помогут

Первым шагом теста будет создание фильтров. Для этого мы будем использовать Groovy-скрипт. Groovy - это язык программирования (надстройка для Java, вики наше все) . Нам пока будут нужны только самые простые конструкции, которые можно нагуглить или спросить у коллеги-программиста.

Для этого шага в принципе можно было бы использовать JDBC Request (подключение к БД), но для развесистых SQL запросов это не всегда удобно.

Итак, прежде всего, нам нужно зарегистрировать наш драйвер postgresql для всего проекта (!в принципе это не обязательно, но если какой-то запрос к БД не работает по непонятной причине - прежде всего попробовать сделать это). Для этого:
1. открыть ProjectViewer (кликнуть дважды по проекту);
2. в нижней панели перейти на вкладку Load Script (скрипты, который выполняются при загрузке проекта;
3. ввести в поле:

```
com.eviware.soapui.support.GroovyUtils.registerJdbcDriver('org.postgresql.Driver')
```



Теперь в нашем тест-кейсе (Project/TestSuite/TestCase) создадим шаг Groovy Scipt (ПКМ - Add Step - Groovy Script) - шаг будет помечен звездочкой;
Откроется окно редактирования скрипта. Нам нужно подключиться к БД и выполнить несколько INSERT запросов для создания фильтров. Используем код:

Подключение к БД и вставка фильтров

Чтобы выполнить запрос нужно нажать зеленую кнопку "старт" в верхней панели окна редактирования запроса.

Первые три строчки кода - регистрация драйвера (еще разок на всякий случай), импорт классов для работы с SQL. В строке def sql - мы определяем подключение к БД (def - объявление переменной, sql - имя переменной), указываем строку подключения, логин, пароль и драйвер. sql.execute (имяПеременной.execute) выполняет по порядку все SQL команды. При успешном выполнении скрипта будет показано окошко Script-result false.

Этот скрипт, конечно, работает, но можно его немного улучшить. Так как мы можем перемещать наши тесты между разными площадками и IP, было бы здорово, если можно было поменять подключение в свойствах проекта (параметризация и свойства проекта описаны в разделах выше). Для этого в свойствах проекта создаем 3 новых переменные, я обозвала их:

pgCon	jdbc:postgresql://192.168.70.91:5432/ORA2PG_T
pgUser	postgres
pgPass	postgres

Скрипт Groovy немного изменим, добавив вызов этих переменных:

Подключение к БД и вставка фильтров

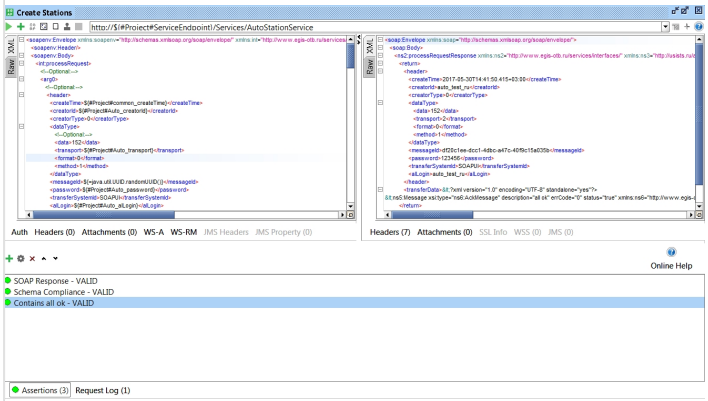
Так как вообще-то этот шаг по сути не входит в тест, правильнее было бы его запихать в Setup Script (два раза кликнуть по Test Steps, в нижней панели будет вкладка Setup Script - туда можно записывать все скрипты, которые нужно выполнить перед началом теста - например загрузить тестовые данные). Но для целей этого гайда я сделала отдельный шаг.

Шаг: Выполнение SOAP запроса и проверка

1. Теперь создадим шаг с SOAP запросом (ПКМ по тест-кейсу - Add Step - SOAP Request).
2. В проекте должен быть уже создан хотя бы один интерфейс: выбрать его из списка.
3. В следующем окне можно все оставить по умолчанию и сразу нажать OK. Я отмечаю еще вторую и третью галочки.
4. Откроется стандартное окно SOAP запросов. Содержимое левой панели сразу грохнуть и вставить свой запрос на создание станций.
5. Выполнение запроса точно также запускается зеленой кнопкой в левом верхнем углу панели.

Когда запрос выполнен, нам нужно **проверить, что он выполнен успешно**. Индикатором этого будет наличие "all ok" в возвращенном запросе. Поэтому нам нужно вставить такую проверку, которая найдет "all ok" в ответе, и выдаст ошибку, если "all ok" там нет.

1) Для того чтобы вставить проверку или assertion, нужно нажать зеленый плюсик (рядом с кнопкой запуска запроса) или раскрыть панель Assertion в нижней части этого окна и нажать плюсик там, см скриншот (по умолчанию нижняя панель Assertion закрыта, и открывается только когда есть ошибки):



2) В окне **Add Assertion** представлен список возможных проверок. Нам сейчас нужна простая проверка - Property Contains > Contains. Выбрать ее и нажать Add.
3) В появившемся окне ввести в поле Content текст "all ok" (можно с кавычками) и нажать OK.
4) Assertion появится в нижней панели и будет зеленой, если проверка пройдена, или красной с указанием ошибки. Assertion можно переименовать (ПКМ - Rename):

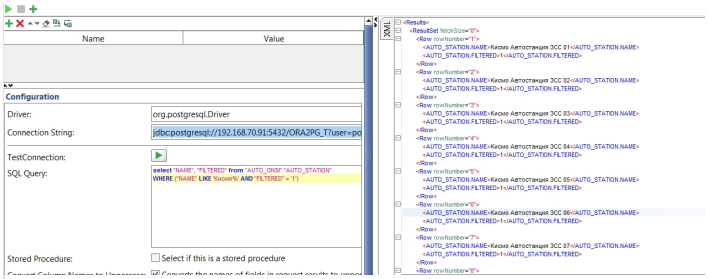
Шаг: Подключение к БД и проверка фильтрации станций

Теперь, когда станции созданы, нам нужно проверить, что:
1. станции есть в БД;
2. нужные станции отфильтровали (статус 1);
3. ненужные станции не отфильтровались (статус 0).

В моем случае есть несколько индикаторов успешности выполнения (это зависит от теста, данных, которые записываются, и ожидаемого результата), например:
1. станции, начинающиеся на указанное имя, должны существовать;
2. станций отфильтрованных не должно быть больше 6;
3. указанные станции должны быть отфильтрованы (6 штук);
4. указанные станции не должны быть отфильтрованы (другие 6 штук).

Для подключения к БД создадим JDBC Request (Add Step > JDBC Request). В окне запрос нужно указать параметры подключения в формате:
Driver: org.postgresql.Drive
Connection Sting: jdbc:postgresql://192.168.70.91:5432/ORA2PG_T?user=postgres&password=postgres

Кнопка Test Connection запускает проверку подключения. В поле ниже вводится SQL запрос к БД. Результаты запроса возвращаются в XML формате, вот так:



В данном случае запрос должен выбрать все станции, начинающиеся на Кисмо и с Filtered=1:

```
select "NAME", "FILTERED" from "AUTO_ONSI"."AUTO_STATION" WHERE ("NAME" LIKE 'Кисмо%' AND "FILTERED" = '1')
```

Точно также, как и в предыдущем шаге, нужно вставить Assertion (Проверку). Здесь представлены примеры разных проверок, на самом деле все они не нужны в данном случае, использую просто для примера:
Простые проверки:

- Простая проверка Property Content > Contains с текстом rowNumber . Проверяет, что по запросу вообще что-то найдено.
- Простая проверка Property Content > Contains с текстом Row rowNumber="6" . Проверяет, что есть по крайней мере 6 результатов.

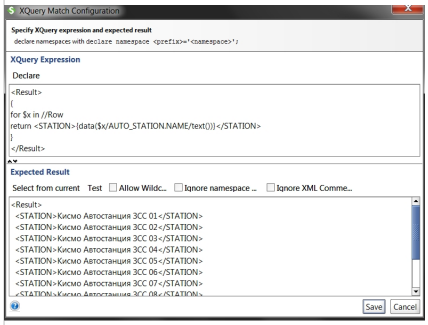
Следующая проверка будет скриптом (Script > Script Assertion). Эту же проверку так же можно разбить на 6 простых проверок (Property Content > Contains) с именем каждой станции, но я хочу все сразу. Скрипт содержит все имена станций, который должны присутствовать в результатах:

```
responseData = messageExchange.getResponseContent()
log.info(responseData)
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 01</AUTO_STATION.NAME>")
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 03</AUTO_STATION.NAME>")
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 05</AUTO_STATION.NAME>")
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 07</AUTO_STATION.NAME>")
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 09</AUTO_STATION.NAME>")
assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция 3СС 11</AUTO_STATION.NAME>")
```

Следующая проверка посложнее и основана на **Xquery** - фактически, аналог SQL запроса к XML. Нам нужно, чтобы в возвращенных от базы данных результатах были только указанные станции, и никаких других. Для этого используем Property Content > **XQuery Match**. Откроется окно редактирования проверки. В верхней части нужно написать запрос, в нижней части - то, что мы хотим увидеть. Например, для того чтобы вывести все станции, которые сейчас возвращены в результате запроса к БД, используем скрипт:

```
<Result>
{
for $x in //Row
return <STATION>{data($x/AUTO_STATION.NAME/text())}</STATION>
}
</Result>
```

Этот скрипт проходит по каждому узлу Row (в xml это Row rowNumber="1", Row rowNumber="2" и т.д.), смотрит в узел AUTO_STATION.NAME, берет оттуда текст (в нашем случае это название станции) и выводит все обнаруженные станции в список. Протестировать скрипт можно нажав кнопку Select From Current - в нижней части отобразится результат.



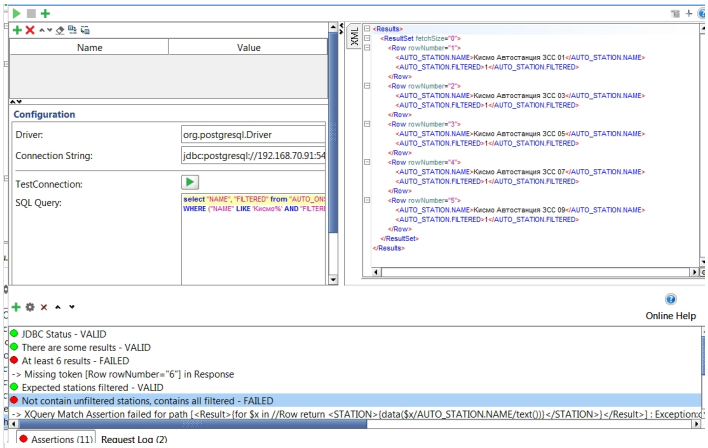
Мне нужно, чтобы в результате было выведено только 6 конкретных станций, поэтому в нижней части Expected Result я введу то, что ожидаю увидеть (только эти станции должны быть возвращены по запросу к БД и никаких лишних):

```
<Result>
  <STATION>Кисмо Автостанция 3СС 01</STATION>
  <STATION>Кисмо Автостанция 3СС 03</STATION>
  <STATION>Кисмо Автостанция 3СС 05</STATION>
  <STATION>Кисмо Автостанция 3СС 07</STATION>
  <STATION>Кисмо Автостанция 3СС 09</STATION>
  <STATION>Кисмо Автостанция 3СС 11</STATION>
</Result>
```

Такая проверка покажет ошибку, если каких-то станций нет, или какие-то станции лишние. Xquery Match очень мощный инструмент, с его помощью можно разбирать возвращенные XML-запросы, делать условные проверки и многое другое.

Далее можно например еще создать проверки на то, что конкретная станция НЕ содержится в запросе (Property Content > Not Contains).

Например, вот результат выполнения проверки после записи станций на нашей тестовой (где почему-то коряво работает фильтрация ФС). Отфильтровалось только 5 станций, проверки это обнаружили:



!

Примеры выполнения конкретных операций

Разбор XML-файла ответа веб-сервиса

Tip: извлечение xml из transferData в одну строку

```
transferData = new XmlHolder((new XmlHolder(messageExchange.getResponseContentAsXml()
transferData.getNodeValue("//*:data/@name") //работаем с xml в transferData
```

Веб-сервисы возвращают ответ в XML формате, данные же помещаются в CDATA в поле tranferData. Предположим, мы создали 3 оператора, обновили их запросили сервис (GET) созданных операторов и хотим проверить, что и как обновилось, правильно ли были помечены старые записи как удаленные.

Для этого есть два подхода:

- проверить в БД (JDBC Request);
- разобрать XML-ответ.

В каких-то случаях быстрее будет использовать запрос к БД, но иногда полезнее разобрать полученный ответ. Так же это можно использовать для более продвинутой автоматизации (например, сделать запрос на ОКАТО, прочитать значение и использовать это значение в следующем запросе).

Для начала разберем простой пример - проверим количество возвращенных записей и короткое имя перевозчика. Это ответ от сервиса с тремя операторами: Один оператор

Чтобы разобрать этот XML нам надо:

- извлечь из transferData/CDATA xml-ку с данными;
- посмотреть атрибут recordCount;
- посмотреть атрибут shortName.

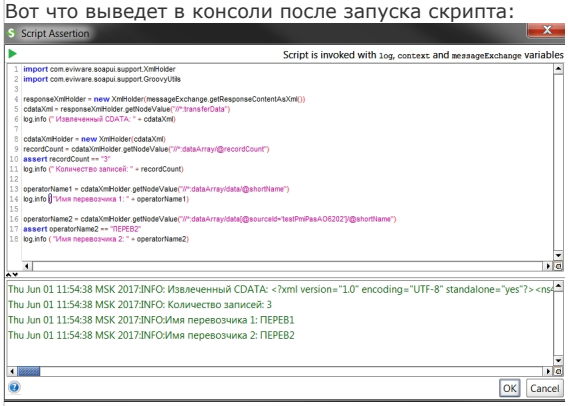
Есть довольно длинный официальный гайд как провернуть это без использования Groovy-скрипта, используя вместо этого фичу Property Transfer, но по-моему это как-то коряво.
Мы используем Groovy-скрипт.

В шаге теста, где мы запрашиваем сервис данные (GET), добавляем Assertion типа Script > Script Assertion. Используем следующий код (смотри комментарии). Как узнать путь (все эти //*:tranferData и @shortName) описано ниже.

Код для извлечения значений атрибутов из XML

Код для получения id из XML

Вот что выведет в консоли после запуска скрипта:



Как узнать путь до конкретной переменной, что такое вообще путь в XML (XPath)? XML-структуру можно представить как дерево папок:
-верхняя папка: <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
--вложенная папка: <soap:Body>
----следующая вложенная папка: <ns2:processRequestResponse>
------еще одна <return>
--------внутри которой папки <header> и <tranferData>.

Можно составить путь до папки самому, а можно использовать замечательный плагин для Notepad++ - XML Tools > Current XML Path (выделив нужный узел, например <trasferData> - поставить туда курсор). Получится что-то вроде:

```
/soap:Envelope/soap:Body/ns2:processRequestResponse/return/transferData
```

Это очень длинный путь со всякими непонятными ns2 и soap (это пространства имен). Нам это все не надо, весь длинный путь мы сокращаем, заменив все символом *. То есть скрипт будет раскрывать все "папки" структуры XML подряд пока не найдет указанную:

```
//*:transferData
```

Точно также можно узнать путь во вложенном в CDATA xml (для этого нужно скопировать в новый файл Notepad++.
Например, dataArray: /ns4:Message/dataArray
Заменяем: //*:dataArray
Поле data: /ns4:Message/dataArray/data
Можно так или так: //*:dataArray/data или //*:data

Чтобы сослаться на конкретный **атрибут** поля **data** используется указатель @имяатрибута, например, ссылаемся на shortName:

```
//*:dataArray/data/@shortName
```

А если нужно сослаться на атрибут поля с конкретным атрибутом (например, нужно узнать shortName поля с конкретным sourceId), то пишем в таком формате:

```
//*:dataArray/data[@sourceId='testPmiPasAO6202']/@shortName
```

В квадратных скобках указываем @имяАтрибута и значение.

Можно сослаться на атрибут уровнем выше, например, нужно найти ПДП с определенной фамилией, и посмотреть его status, то есть:

```
<data xsi:type="ns7:AutoPDP" status="1" .....>
            ...
            <surname value="КисмоАвтоГ"/>
           ...
</data>
```

Вот так:

```
"//*:dataArray/data/surname[@value='КисмоАвтоА']/../@status"
```

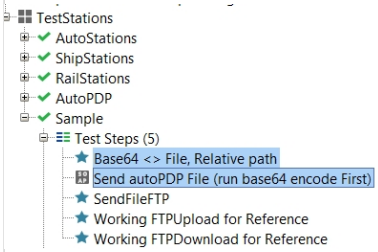
Автоматическая сериализация в base64 в запросе

TODO: улучшить параметризацию пути (перенести часть в test-case?)

Отправка ПДП в сервис ParserManagerSoapBinding выполняется в формате base64. Для этого мы берем содержимое файла, открываем его в блокноте, все выделяем и жмем base64 encode. Муторно. Как сделать, что оно все само?
Вот так.

Для начала поймем, что нужно сделать:
1. взять файл, который лежит в какой-то папке. Путь желательно относительный, например в подпапке где лежит файл проекта SoapUI;
2. считать содержимое файла (в UTF-8 формате, чтобы не было кракозябр!);
3. кодировать в base64;
4. записать эту кодированную штуку в какую-нибудь переменную;
5. вставить переменную в наш запрос Soap.

Так как невозможно все это проделать в одном шаге, у нас будет два шага:
1. скрипт Groovy - закодировать файл и записать получившиеся кракозябры в переменную;
2. сам запрос - содержит эту самую переменную.



Файл проекта лежит в какой-то папке. Создадим подпапку (например, testfiles), а в ней еще одну папку, например kismoPMI, а там папку для транспорта, например auto.
Вообще-то дерево папок не очень важно, так как мы вынесем путь в свойства, и вы сможете задать свою структуру. Для пример я буду использовать такую:
--папка SoapUI <<< тут лежит файл проекта
----testFiles <<< это общая папка для всех тестовых файлов
------kismoPMI <<< папка для определенной задачи
--------auto <<< сегмент, тут лежит файл pdp.csv

Сначала нам нужно получить путь к файлу проекта (чтобы тест можно было запускать с любой машины и из любой папки). В Custom Properties проекта создадим свойство testFilesFolder и запишем туда путь к тестовым файлам, например \testFiles\kismoPMI\auto

```
import com.eviware.soapui.support.GroovyUtils

/*Прочитаем значение переменной testFilesFolder в Custom Properties проекта */
```

```
def testFilesFolder = testRunner.testCase.testSuite.project.getPropertyValue( "testFilesFolder" )
log.info(" Путь к папке с тестовыми файлами относительно файла проекта: " + testFilesFolder)

/*Эти две строчки прочитают путь до папки с проектом и запишут его в переменную projectPath*/
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
def projectPath = groovyUtils.projectPath
log.info(" Путь к папке проекта на диске: "+ projectPath)
```

После этого нам нужно прочитать файл из папки. Мы берем путь к проекту (projectPath), прибавляем к нему путь из свойств проекта (testFilesFolder) и в конце указываем постоянный путь (/pdp.ccv). В результате скрипт прочитает файл, расположенный на компьютере в папке /<путь к папке проекта>/testFiles/kismoPMI/auto/pdp.csv.

```
def pdpFile = new File(projectPath + testFilesFolder, "/pdp.csv" ).getText("UTF-8");
log.info(" Содержимое файла: " + pdpFile)
```

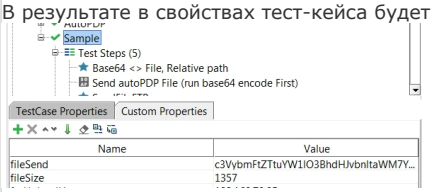
В запросе SOAP, который отправляет файл, также указывается размер файла. Там может быть рандомное число, можно его не менять. Но можно получить размер нашего файла и записать его в переменную fileSize (переменная в Custom Properites тест-кейса):

```
def fileSize = pdpFile.length().toString()
testRunner.testCase.setPropertyValue( "fileSize", fileSize )
```

Затем нужно взять содержимое нашего файла, перевести его в строку UTF-8 (иначе будут кракозябры), кодировать в base64 и сохранить получившуюся строку переменную fileSend в свойствах тест-кейса. (Custom Properties):

```
String pdpFileEncoded = pdpFile.getBytes( 'UTF-8' ).encodeBase64()
testRunner.testCase.setPropertyValue( "fileSend", pdpFileEncoded )
```

В результате в свойствах тест-кейса будет сохранены два значения - размер файла и строка base64:



Целиком скрипт:

Теперь эти переменные можно использовать в запросе. Создаем шаг тест-кейса с SoapUI запросом к PasrerManagerRouter... и делаем вот такой запрос:

Запрос с base64

Как видно в файле запроса я использую переменные и формулы для генерации даты и случайного archiveId. Попорядку:

- <fileData>${#TestCase#fileSend}</fileData> - здесь мы достаем нашу сериализованную скриптом строку из переменной fileSend и вставляем в fileData
- size="${#TestCase#fileSize}" - берем посчитанный скриптом размер файла из переменной fileSize
- fileName= - так как обработчику неважно имя, можно не менять. А можно тоже сгенерировать автоматически, как - описано тут, во второй части
- createdAt - тут длинная формула, которая всего навсего ставит в createdAt текущую дату - 2 часа;
- archiveId="${=java.util.UUID.randomUUID()}"- а это обычная формула генерации рандомного UUID, чтобы руками не менять.

Теперь, если нужно загрузить какой-то файл, складываем его в папку (auto/pdp), запускаем скрипт, а затем запускаем запрос.

Сериализация/Десериализация base64, проверки

Дано: при тестировании ПУД в операции создания пользователя требуется отправлять данные, закодированные в base64. Ответ тоже приходит в base64. Надо как-то эту информацию извлекать, кодировать туда сюда.
Этот же способ можно использовать, когда требуется некоторую xml-ку закодировать в строку base64 (требуется для тестирования некоторых методов).

Данные пользователя в декодированном виде выглядят вот так (например): Показать

Можно их сразу закодировать в строку base64 и так и использовать в тесте, но в этом случае не всегда получится что-то быстро поменять (например, права или логин). Поэтому перед выполнением шагов теста можно вставить шаг скрипта, в котором мы запихнем эти данные в читаемом виде и закодируем их скриптом в переменную. Вот так:

Показать

Если надо сгенерировать много всяких base64 имеет смысл написать метод, а потом его вызывать сколько надо раз Пример с методом

В запросе просто заираем готовую строку base64 из свойства тест-кейса.

ПУД частенько возвращает ответы в base64. Чтобы разобрать ответ строку надо декодировать. Для этого декодированный байт-код нужно перевести в читаемую строку. Вот так (скрипт вставлен в Script Assert в шаге Soap): Показать

Далее можно сравнить ответ и с некоторой строкой. Например, в ПУД убедиться, что он вернул именно те изменения, которые мы записали. Самое простое - сравнить две строки. Это можно сделать сразу в base64 или сравнивать декодированные строки. Если получилось сравнить строки прямо в base64 - замечательно. К сожалению, это не всегда работает(Поэтому можно извернуться и 1) декодировать полученный ответ 2) сравнить ответ с декодированным отправленным запросом, обрезав все лишние символы (концы строки, пробелы и табуляцию. При этом, сравнивать напрямую не получится, так как вредный ПУД может вернуть записи в разном порядке. Поэтому сравнивать декодированную строку надо посимвольно, для этого: символы в декодированном ответе сортируем и сравниваем получившиеся строки
В примере ниже также добавлено условие, по которому всю байду с base64 делаем, если в ответе не возвращено ошибки "Пользователь х не найден".

Показать

Как загрузить файл по FTP

Скриптом SoapUI можно загрузить через FTP. То есть без всяких там Filezilla. Для этого правда нужна библиотека commons-net-3.6.jar (приложена к статье). Ее нужно положить в папку C:\Program Files\SmartBear\SoapUI-5.3.0\bin\ext и перезапустить SoapUI. В этой библиотеки есть всякие полезные классы и методы, в том числе FTPClient. Загрузка файла выполняется скриптом Groovy (TestCase > Add Step > Groovy скрипт).

У меня этот скрипт уже получился довольно развесистый, он использует предыдущие наработки, поэтому нужно прочитать предыдущие разделы.
По порядку что нам нужно сделать:

1. сгенерировать имя файла (чтобы не нужно было менять руками дату). Имя файла будет состоять из префикса (20000_ , записывается в свойствах тест-кейса) и рандомной даты за последний месяц + разрешение .csv;
2. записать получившееся имя в переменную (мы же потом захотим получить на этот файл квитанцию? так что надо запомнить имя)
3. отправить файл по FTP. Параметры подключения к FTP серверу я также вынесла в свойства тест-кейса.

В результате в свойства тест-кейса у меня следующее:

ftpUploadHost	192.168.70.95
ftpUploadPort	21021
ftpUploadUser	auto_test_ru
ftpUploadPass	123456
ftpFilePrefix	20000
ftpPdpFileName	20000_2017_05_18_16_28_18_912.csv

Скрипт разбит на две части. В первой мы генерируем имя файла. Генерация даты отдельно описана тут . Во второй части создаем FTPClient и подключаемся к серверу с указанными параметрами (адрес, порт, имя пользователя, пароль). В первых строках скрипта можно видеть много import - это мы импортируем нужные библиотеки. Под катом длинный скрипт с подробными комментариями. Это рабочий скрипт, и если нужно загрузить другой файл - нужно только изменить путь к файлы в свойствах проекта и имя файла (его тоже можно в свойства вынести, чтобы вообще скрипт не менять).

См. новую версию скрипта ниже.

Старая

Новая версия скрипта генерирует дату (за последние 12 ч.), берет существующий файл на ФС, переименовывает его и закидывает на FTP. Загрузка реализована в виде метода, поэтому ее можно использовать для xml и ftp. Параметры подключения к серверу записаны в одну переменную, из которой извлекается строка и разбивается на нужные параметры по символу ",".

Новая версия скрипта

Генерация рандомных строковых данных на примере ФИО

Есть несколько способов генерации рандомных данных, мы рассмотрим два: рандомный набор букв/цифр/символов или рандомный выбор из подобранного набора данных. Первый способ проще реализовать, но генерируется часто нечитаемый набор, типа "Грывалжсва". Второй способ сложнее, но зато можно подобрать более менее приличные, читаемые данные.

Также если нам нужно создать много записей, то шаги с генерацией и записью данных можно зацикливать, это описано здесь.

Помимо ФИО (поля name, surname..) в БД записываются также нормализованные значения этих полей, которые тоже передаются в запросе. Можно пойти простым путем и просто перевести все символы строки в верхний регистр. Однако, для части проверок может потребоваться именно нормализация/транслитерация (например, для проверок поиска по БД ПОП). Для этого тоже можно написать скрипт на Groovy.

Сначала рассмотрим генерация рандомной строки из набора символов. Для этого:

- напишем метод, который принимает на вход алфавит (в нашем случае - русский) и количество символов;
- сгенерируем фамилию, имя и отчество и сделаем первые буквы заглавными;
- напишем метод для транслитерации;
- сохраним все в свойства тест-кейса.

Генерация строки из символов

Пример сгенерированных ФИО: `Цсщвлпрхп Тткимщшцс Ютшудчжкс`. Не очень созвучно, но уж что есть. Также при записи в переменную можно добавлять какой-нибудь префикс, например "Тест", чтобы позже находить даные в БД (например `testRunner.testCase.setPropertyValue("Name","Тест" + pdName)`).

Следующий пример с выбором данных из списка. В данном примере я опущу метод транслитерации. делаем следующее:

- формируем списки возможных имен, фамилии и отчеств;
- записываем в свойства проекта рандомно выбранные ФИО.

Выбор случайного элемента из списка

Пример ФИО: Диранов Спазмалгон Илиносивич

Отдельные примеры скриптов для выполнения тривиальных операций

Генерация случайного числа (между) или случайной даты (между)

Генерацию случайно числа удобно использовать, например, для рандомных идентификаторов (`sourceId`), даты и т.п.
В Java ниже версии 1.7 используется класс Random, Math.random, однако в версиях старше 1.7 рекомендуемый способ - `ThreadLocalRandom`. Ниже я приведу примеры со старой реализацией, и с новой (рекомендуемой). SoapUI работает с версией java 1.8 и выше, так что смело можно использовать `ThreadLocalRandom`.

Генерация рандомного числа между указанными значениями (min ... max) выполняется по формулам:

```
# с java 1.7
import java.util.concurrent.ThreadLocalRandom

ThreadLocalRandom.current().nextInt(min, max + 1)

# до Java 1.7
Math.random()*(max - min))+min
```

Например, случайное число между 5 и 10:

```
# с java 1.7
ThreadLocalRandom.current().nextInt(5, 10 + 1)

# до Java 1.7
Math.random()*(10 - 5))+5
```

Результат для ThreadLocalRandom: 8
Результат Math.random: 7.356781231

Чтобы округлить число до целого, нужно добавить еще Math.Round. Например, число между 1001 и 1140:

```
Math.round((Math.random()*(1140 - 1001))+1001)
```

Результат: 1125
Для ThreadLocalRandom дополнительно ничего делать не нужно. Правда, круто?

Прямо в теле запроса SOAP можно использовать в таком формате:

```
// с java 1.7
${=import java.util.concurrent.ThreadLocalRandom; (int)(ThreadLocalRandom.current().nextInt(1000, 1900+1))}

// до Java 1.7
${=(int)(Math.round((Math.random()*(1140 - 1001))+1001))}
```

Например, для поиска рандомного IS по базе (для нагрузочного теста):

```
<filter xsi:type="BinaryFilter" operation="eq" negate="false">
            <attr xsi:type="Attribute" name="gid" />
            <right xsi:type="Constant">
                <value xsi:type="xs:long" xmlns:xs="http://www.w3.org/2001/XMLSchema">${=(int)(Math.round((Math.rand
            </right>
        </filter>
```

В скрипте рандомное число присваивается, например, переменной. Вот так:

```
// с java 1.7
import java.util.concurrent.ThreadLocalRandom

randomNumberNew = ThreadLocalRandom.current().nextInt(1000, 1900+ 1)
log.info ("Случайное число: " + randomNumberNew.toString())

// до Java 1.7
randomNumber = Math.round(Math.random()*(59 - 1) + 1)
```

Также может пригодится генерация случайной даты. Например, мне это понадобилось для генерации названия файла для загрузки по FTP. Формат имени файла включает год, месяц, день, часы, минуты, секунды и мс. Чтобы было как положено, сгенерируем случайную дату, которая попадает в период между сегодняшней датой (и временем) и датой месяц назад. Это выполняется скриптом Groovy, нужно будет импортировать несколько нужных библиотек. Вот так (читаем комментарии):

Надо вот так

Не надо так больше

Результат, например: 2017_05_30_20_55_57_541

Установка и получение свойств из проекта или тест кейса

Свойства проекта или тест-кейса (properties) удобно использовать для сохранения и извлечения каких-то общих значений.
Например, можно посмотреть идентификатор станции, сохранить его в свойство тест-кейса, а потом использовать его для создания расписания.
Официальная документация по свойствам

Есть два типа скриптов: скрипт-шаг (Add Step - Groovy Script) и script assertion (скрипт проверка - который добавляется в шаге теста для проверки результатов). У этих скриптов немного разный контекст, поэтому назначение и получение свойств выполняется немного по разному.

Для начала обычный скрипт-шаг (**Groovy script**). Вот так получаем значения из свойств проекта:

```
def someVar = testRunner.testCase.testSuite.project.getPropertyValue( "propertyName" )
```

где property.. - название свойства проекта, someVar - какая-то переменная

Установка и получение свойств на всех уровнях:

```
// Установка свойств на уровне Проекта
 testRunner.testCase.testSuite.project.setPropertyValue( "projectProperty" ,"Значение свойства")
// Установка свойств на уровне Тест-суита
 testRunner.testCase.testSuite.setPropertyValue( "testSuiteProperty" ,"Значение свойства")
// Установка свойств на уровне Тест-кейса
 testRunner.testCase.setPropertyValue( "testCaseProperty" ,"Значение свойства")
```

```
//Получение свойства на уровне Проекта
def someVar1 = testRunner.testCase.testSuite.project.getPropertyValue( "projectProperty" )
//Получение свойства на уровне Тест-суита
def someVar2 = testRunner.testCase.testSuite.getPropertyValue( "testSuiteProperty" )
//Получение свойства на уровне Тест-кейса
def someVar3 = testRunner.testCase.getPropertyValue( "testCaseProperty" )
```

Для того чтобы установить свойство тест-кейса из **Script Assertion**, нужно немного по другому:

```
messageExchange.modelItem.testStep.testCase.setPropertyValue( "propetyName", "Привет, Мир!"  )
```

будет создано свойство тест-кейса с именем propetyName, со значением Привет, Мир!.

Установка и получение свойств из script assertion:

```
// Установка свойств на уровне Проекта
messageExchange.modelItem.testStep.testCase.testSuite.project.setPropertyValue( "projectProperty" ,"Значение свойства
// Установка свойств на уровне Тест-суита
messageExchange.modelItem.testStep.testCase.testSuite.setPropertyValue( "testSuiteProperty" ,"Значение свойства")
// Установка свойств на уровне Тест-кейса
messageExchange.modelItem.testStep.testCase.setPropertyValue( "testCaseProperty" ,"Значение свойства")

//Получение свойства на уровне Проекта
def someVar4 = messageExchange.modelItem.testStep.testCase.testSuite.project.getPropertyValue( "projectProperty" )
//Получение свойства на уровне Тест-суита
def someVar5 = messageExchange.modelItem.testStep.testCase.testSuite.getPropertyValue( "testSuiteProperty" )
//Получение свойства на уровне Тест-кейса
def someVar6 = messageExchange.modelItem.testStep.testCase.getPropertyValue( "testCaseProperty" )
```

Переменной testCaseProperty будет назначено значение свойства тест-кейса с MyProp

Создание, удаление файлов и пути до файлов

В тесте может потребоваться создать какой-то файл в директории. Для этого нам надо:
1. сделать путь до папки относительным (чтобы работало на любом компьютере);
2. создать папку, если ее нет
3. создать файл

Чтобы получить путь до файла проекта:

```
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
def projectPath = groovyUtils.projectPath
log.info(" Путь до папки проекта: " + projectPath)
```

Путь к папке с проектом сохраняется в переменную projectPath. Мы можем создавать файл прямо в корне папки, тогда файлы будут сохранятся рядом с .xml файлом проекта.
Если же нужно создать подпапки, то можно, например, задать дополнительную структуру в отдельной переменной (или достать ее из свойств проекта). Например:

```
def testFilesFolder = ( '\\files\\pmi\\' )
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
def projectPath = groovyUtils.projectPath
log.info(" Путь до папки проекта: " + projectPath + testFilesFolder)
```

В результате будет сформирован путь: <путь к папке с проектом>/files/pmi.

Так как папка может не существовать на момент запуска теста, нужно это проверить, и если папки нету - создать ее:

```
def testDir = new File(projectPath + testFilesFolder)
if( !testDir.exists() ) {
  testDir.mkdirs()
}
```

Чтобы создать в этой папке файл нужно также проверить, нет ли там уже его.
В зависимости от особенностей теста, может потребоваться создать новый файл или добавлять записи к существующему.
Код ниже всегда создает новый файл с указанным именем. Если файл уже есть - он удаляется.

```
def pdpFile = new File(projectPath + testFilesFolder + "kismoAutoPdp_generated.csv")
if (pdpFile.exists()) {
    pdpFile.delete();
}
```

Будет создан файл в директории: <путь к папке с проектом>/files/pmi/kismoAutoPdp_generated.csv.

Чтобы добавить в созданный файл какие-то записи, можно использовать команду append (будет добавлять строки в конец файла). Например, ниже код создает две записи - заголовок файла ПДП и одну запись ПДП. В конец каждой строки добавляется конец строки (\n)

```
def pdpHeader = "surname;name;patronymic;birthday;docType;docNumber;documentAdditionalInfo;departPlace;arrivePlace;r
def pdpLine1 = "КисмоАвтоА;Алефтин;Тестовый;1971-11-15;0;8602328040;;Кисмо Автостанция ЗПФС 01;Кисмо Автостанция ЗПФ
pdpFile.append(pdpHeader +"\n" + pdpLine1 + "\n", "UTF-8")
```

Примечание: ${ttDatePdp} - это переменные, который содержат сгенерированную дату.

Если требуется создать файл с фиксированным контентом, то правильный способ это сделать вот так. Этим способом можно записывать в файлы в правильной кодировке:

Пример создания файла на ФС в UTF

Прочитать файл upload.xml и закодировать в base64: Прочитать файл

Активировать или деактивировать шаги теста по условию

Некоторые шаги тест-кейса иногда требуется активировать или деактивировать в зависимости от условия.
Например, если оператор с egisId существует в базе - то для теста используем его, соответственно шаги по созданию оператора деактивируем.
Если оператора нет - активируем шаги по созданию оператора. Такую операцию можно выполнить только в Groovy Script (то есть в Script Assertion это не сработает, ну или надо как-то контекст поменять, но как я не знаю). Поэтому, например для проверки оператора, можно сделать так:

1. шаг теста с SOAP запросом оператора по egisId;
2. если возвращено 0 записей, то в свойство тест-кейса ставим createOperator=true
3. если возвращена запись - ставим createOperator=false
4. в шаге тест-кейса Groovy Script смотрим значение свойства createOperator, и в зависимости от значения активируем или деактивируем шаги.

Для активации шага теста:

```
def oNameList = testRunner.testCase.getTestStepList().name
testRunner.testCase.getTestStepByName("Имя шага тест-кейса").setDisabled(false)
```

Для деактивации:

```
def oNameList = testRunner.testCase.getTestStepList().name
testRunner.testCase.getTestStepByName("Имя шага тест-кейса").setDisabled(true)
```

getTestStepList требуется для получения текущего состояния и списка всех шагов тест-кейса.

Например вот так: Активация шагов теста

Операции с БД в скрипте Groovy

При подключение к БД из Groovy Script шага можно получить, например, значение какого-то поля.

Например, найдем станцию в базе и извлечем значение колонки ASID (идентификатор станции):

```
def sql = Sql.newInstance("jdbc:postgresql://192.168.70.91:5432/ORA2PG_T","postgres","postgres","org.postgresql.Driv
def stations = sql.rows '''
SELECT "ASID" FROM "AUTO_ONSI"."AUTO_STATION" WHERE "FOREIGN_ID" LIKE 'testKismoPdpAs%' ORDER BY "FOREIGN_ID" ASC;
'''
def stationId1 = stations[0][0]
def stationId2 = stations[1][0]
log.info(" ASID станции: " + stationId1)
```

Запрос выше возвратит массив значений - array (то есть несколько станций). Конструкция `stations[0][0]` вытаскивает значение первой строки первого ряда, [1][0] - второй строки первого ряда и т.д. Чтобы записать это значение из базы, например, в свойство проекта, нужно перевести его в строковое значение. Вот так:

```
testRunner.testCase.setPropertyValue( "station01Id", stations[0][0].toString() )
```

Иногда требуется получить некий список значений и с каждым значением из списка выполнить какие-то действия.
Например, найти в базе все расписания (TIMETABLE) с определенным названием (в том числе сгенерированные), получить их TTID и по этому TTID удалить записи из смежных таблиц. Для этого можно использовать вот такой цикл:

```
def timetable = sql.rows '''
SELECT "TTID" FROM "AUTO_TIMETABLE_NEW"."TIMETABLE" WHERE "NAME" = 'Кисмо Авторасписание ЗПФР 01';
'''
for(i=0; i<timetable.size; i++) {
    def ttid = timetable[i][0]
    sql.execute "DELETE FROM \"AUTO_TIMETABLE_NEW\".\"AUTO_ROUTE_POINT\" WHERE \"TTID\" = '${ttid}'; DELETE FROM \"A
}
```

Пример с подключением немножко покороче.
В одну переменную записываем всю строку подключения в формате, например:
`jdbc:oracle:thin:@192.168.70.90:1521:orcl,testadmin,oracle,oracle.jdbc.OracleDriver` (без кавычек).
Затем в скрипте разобьем строку по символу ",", и выполним запрос. Дополнительно, для sql соединения можно (даже нужно) указывать таймаут, а после выполнения - закрывать соединение:

Пример с очисткой БД пользователей ВЗО после теста

Как зациклить тест (цикл), и как поменять Endpoint у всех шагов

Дано: один и тот же сервис развернут на нескольких серверах, везде нужно проверить.
В принципе можно менять IP сервиса вручную, но можно сделать так:

1. записать список IP всех серверов где развернут сервис в переменную;

2. повторять все шаги теста для каждого IP-адреса из списка.

Testcase:
1. Groovy скрипт, в котором определяем логику и меняем IP-шники
2...n Все наши шаги теста, которые надо повторять
n+1. Groovy скрипт, который проверяет, есть ли еще в списке IP, и если есть - отправляет в начало теста (1)

В переменных тест-кейса запишем например:

```
IP:192.168.70.95:9080,192.168.70.95:9081,192.168.70.95:9082
counter:0
```

В первом скрипте запишем Показать

В последнем скрипте нужно проверить текущее значение счетчика, сравнить его с количеством IP в списке. Если значение счетчика меньше (еще не все IP проверили), то повторить весь тест с первого шага (шаг вызывается по имени). Если список закончился, установить счетчик на 0 и закончить тест.

Показать

Запуск внешнего генератора

Включить внешний генератор можно с помощью следующего кода. В Custom Properties проекта требуется указать TTGen.

```
import java.lang.ProcessBuilder
import java.util.concurrent.TimeUnit

def process = testRunner.testCase.testSuite.project.getPropertyValue( "TTGen" ).execute()

process.waitForProcessOutput()

def exitValue = process.exitValue()
```

Немного интерактива с выбором среды запуска

В тесты можно добавить немного интерактива (всплывающие информационные окна). Их имеет смысл использовать только в специфических тестах или в тестах для ПМИ (программа и методика испытаний, где проверяющим надо показать красивый результат, а не просто все зелененькие шаги). В обычных тест-кейсах интерактив не нужен, так как в идеале эти тесты должны запускаться с командной строки, отчет в файл - минимум вмешательства QA.
Также можно создать для тест-кейса или тест-суита шаг со скриптом, в котором можно выбрать среду запуска теста, и, в зависимости от выбора, установить значения переменных.

Покажу небольшой интерактив на примере теста для ПМИ для ПУД. Тест-кейс "Управление пользователями". Тест-кейс делает полную цепочку: 1) генерит данные и файлы base64 2)создает, удаляет, редактирует и проверяет, что все корректно 3) удаляет все за собой. Метод sendCommand, который используется для управления пользователями, в основном получает и передает данные в base64, что неудобно для наглядного сравнения и подтверждения результатов.

Чтобы было все очевидно и проверяющим выводились информационные окошки используем класс UISupport:

```
import com.eviware.soapui.support.UISupport

UISupport.showInfoMessage("Это маленькой информационное сообщение в двойных кавычках!")

UISupport.showInfoMessage("""А это большое сообщение
из нескольких строк!

Внутри можно использовать пробелы, пустые строки и переменные ${var}!
""")
```

Класс работает в шаге Groovy Script и в Script Assertion. Пример использования: в ПУД мы создаем пользователя, отправив сообщение с base64, потом читаем созданного пользователя и сравниваем отправленное с полученным. Проверяющему выводим сообщение, в котором наглядно показываем: вот что отправили, вот что получили.

Показать

С помощью UISupport также можно задавать юзеру вопросы и записывать ответ:

```
String answer = UISupport.prompt("Вопрос","Заголовок","дефолтное значение")
```

Пример выбора среды: Выбор среды для теста

Разработка библиотеки

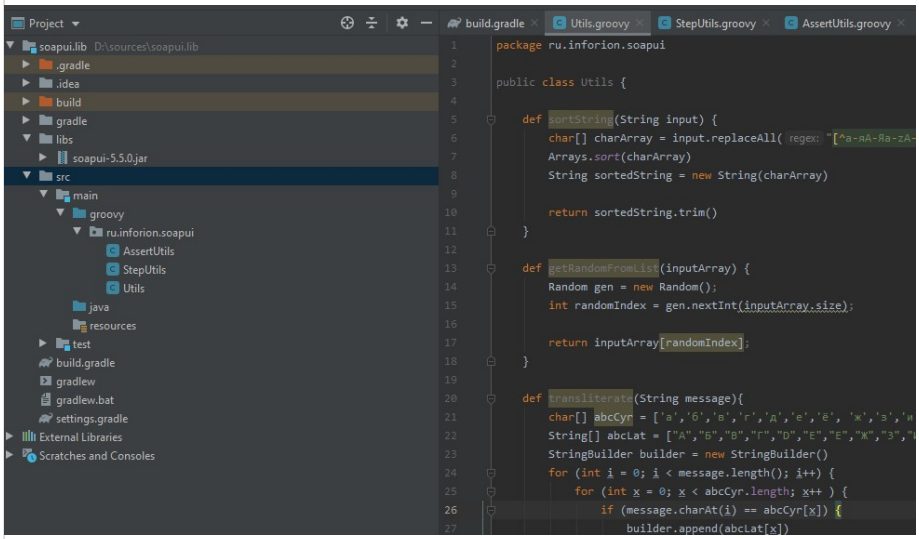
Скрипты это очень хорошо, но что делать, если какие-то конструкции вы используете вот прям постоянно и вас достало писать одно и то же? Можно вынести часть постоянно используемых скриптов в библиотеку и вызывать нужные методы, когда они понадобятся. SoapUI Pro поддерживает такую фичу из коробки, прямо из интерфейса пользователя, но пользователям SoapUI Opensource придется идти путем просветления: написать скрипты и скомпилировать их в jar, а затем скопировать в папку <SoapUI instal dir>/bin/ext.

Нам понадобятся:

- среда разработки (например, Idea Intellij, хватит free версии);
- Groovy (распаковать, добавить в переменные окружения);
- gradle, который нам поможет все собрать все в jar файл.

Установите Idea, установите Gradle (загрузить и распаковать) и добавьте gradle в переменные окружения (вот тут написано как). В Idea в Settings > Build, Execution, Deployment > Build Tools > ... убедиться, что путь к gradle подцепился.

Создать в idea новый проект, выбрать Gradle и указать artifact id например soapui.lib (groupid можно оставить пустым). Gradle создаст структуру папок и выполнит начальную настройку. Вот такая структура папок, например, вышла у меня:



Чтобы использовать некоторые классы soapui (например для установки свойств тест-кейса) нужно импортировать библиотеку soapui. Ее можно скопировать прямо из установленного SoapUi/bin/файл soapui-5.5.0.jar , скопировать в папку /libs в директории проекта (создать папку). См. на рисунке выше. Чтобы подключить библиотеку нужно в файл build.gradle отредактировать раздел dependencies, а также добавить информацию о jar

Показать

Библиотека xmlbeans нужна soapui для работы с XML.

Наши классы объединим в package ru.inforion.soapui. Это нужно для того, чтобы мы в SoapUI могли импортировать все классы вот так: `import ru.inforion.soapui.*`, ну и чтобы с другими пакетами, коих в SoapUI тысячи, не пересекались названия методов и классов.

В папке /main/groovy создадим package (правой кнопкой - package) и обозвать его ru.inforion.soapui. Кликнуть правой кнопкой по package и создать Groovy Class. Я разделила методы на три класса. Пусть первый будет Utils. Методы этого класса можно использовать и в Test step, и в Assertion и они не требуют никаких дополнительных библиотек.
Idea очень умная, так что она сразу создаст каркас для класса, осталось только вставить методы. Например:

```
package ru.inforion.soapui

public class Utils {

    def sortString(String input) {
        char[] charArray = input.replaceAll("[^a-яА-Яа-zA-Z0-9 ]+","").toCharArray()
        Arrays.sort(charArray)
        String sortedString = new String(charArray)

        return sortedString.trim()
    }
}
```

Этот класс можно сразу скомпилировать и протестировать. Для этого в окне Idea справа (такая маленькая вертикальная панелька) раскрыть панель Gradle, раскрыть Tasks/build и дважды кликнуть "jar". Проект скомпилируется, готовый .jar будет в папке проекта в директории /build/libs. Этот .jar необходимо скопировать в папку C:\Program Files (x86)\SmartBear\SoapUI-5.5.0\bin\ext и перезапустить SoapUI.

В Groovy Script вызывать класс и метод нужно вот так:

```
import ru.inforion.soapui.*

def myClass = new Utils()

log.info (myClass.sortString("Ура работает!"))
```

Для того чтобы использовать объекты SoapUI, например устанавливать свойства тест-кейса или работать с XML ответом от сервиса, нужно указать контекст. Во втором классе у меня будут методы для использования в Test step. В assertion контекст другой, поэтому его придется тоже запихать в отдельный класс.

Создадим маленький метод для сериализации строки в base64 и сохранения результата в свойство тест-кейса:

```
package ru.inforion.soapui

class StepUtils {
    def context
    def log
    def testRunner

    def base64This(String prop, String input) {
        String encoded = input.getBytes( 'UTF-8' ).encodeBase64()
        testRunner.testCase.setPropertyValue("${prop}",encoded)
    }

}
```

Вызвать этот класс и метод можно вот так:

```
import ru.inforion.soapui.*

def anotherClass = new StepUtils(context: context, log: log, testRunner: testRunner)
anotherClass.base64This("Property","Blabla")
```

Скрипты в Assertion не имеют доступа к testRunner, поэтому там нужно использовать messageExchange. Например ниже метод, который извлекает содержимое CData из ответа сервиса

```
package ru.inforion.soapui

import com.eviware.soapui.support.XmlHolder

class AssertUtils {
    def log
    def context
    def messageExchange

    def getResponseCdata() {
        XmlHolder responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
        String dataXml = responseXmlHolder.getNodeValue("//*:transferData")
        XmlHolder dataXmlHolder = new XmlHolder(dataXml)

        return dataXmlHolder
    }
}
```

Вызов в assertion. Проверяла в ПУД, на методе sendfileReadRequest.

```
import ru.inforion.soapui.*

def someClass = new AssertUtils(log: log, context: context, messageExchange: messageExchange)
log.info ("Результат: " + someClass.getResponseCdata().getNodeValue("//fileInfo/@fileName"))
```

Обновлено Анна Леонова почти 2 года назад · 76 изменени(я, ий)

Go to top