

SoapUI Автоматизация тестирования

SoapUI обладает большими возможностями по автоматизации тестов. Здесь будет много скриптов и примеров. Начальные навыки программирования приветствуются, но не обязательны.

Полезные примеры всяких скриптов и решений:

- [Learning SoapUI](#)
- [Сборник примеров скриптов](#)
- [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 JDBC 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)

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 выполняются 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:

```
1 create user testadmin identified by oracle;
2
3 grant dba to testadmin;
```

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

Пример тест-кейса

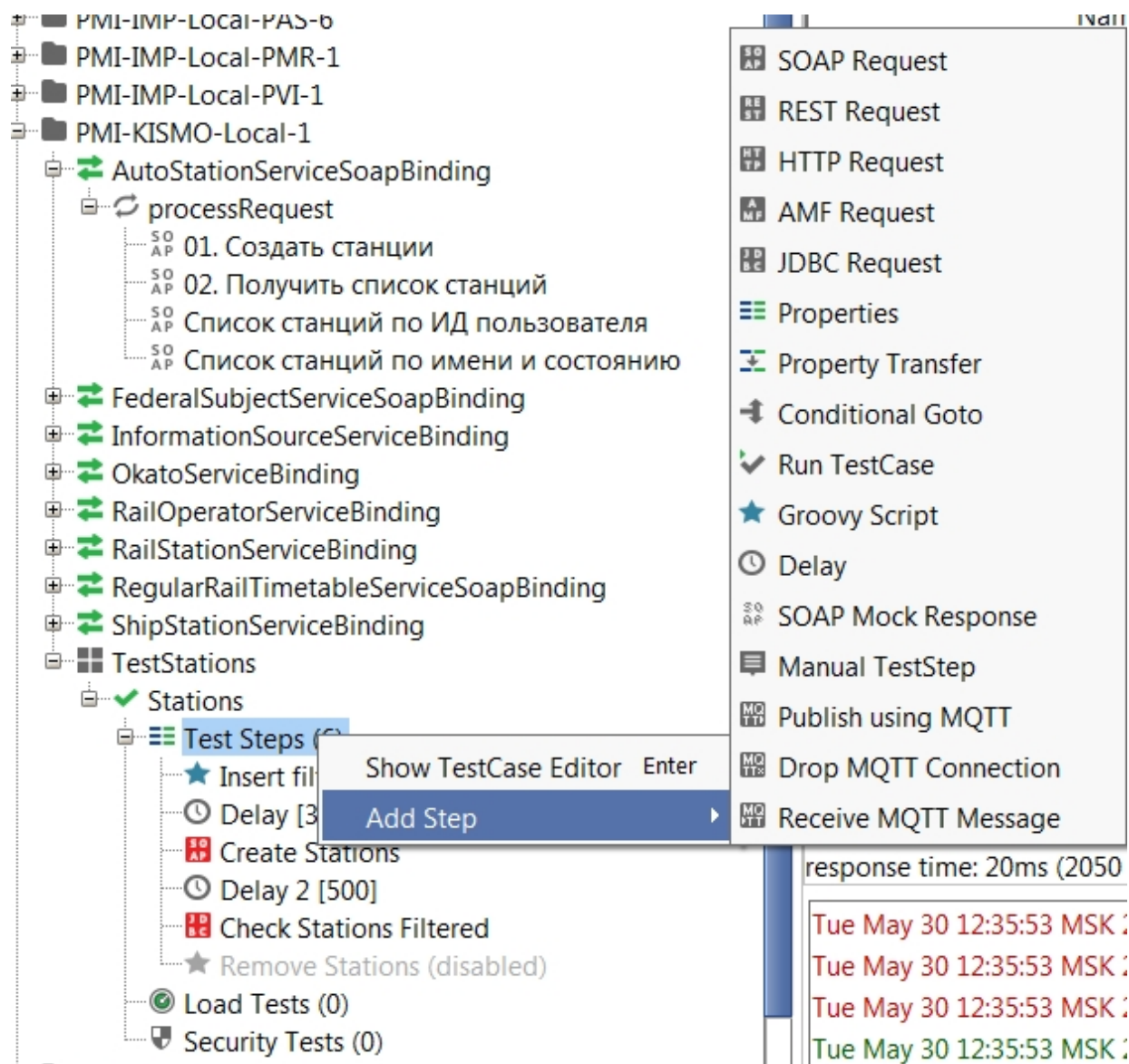
Ниже описан простой тест-кейс, который состоит из шагов:

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 параметризованы, в свойствах проекта расставлены переменные (ip адрес, creatorId и т.д.). Все это описано в основной статье, 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 TestSuite, 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.

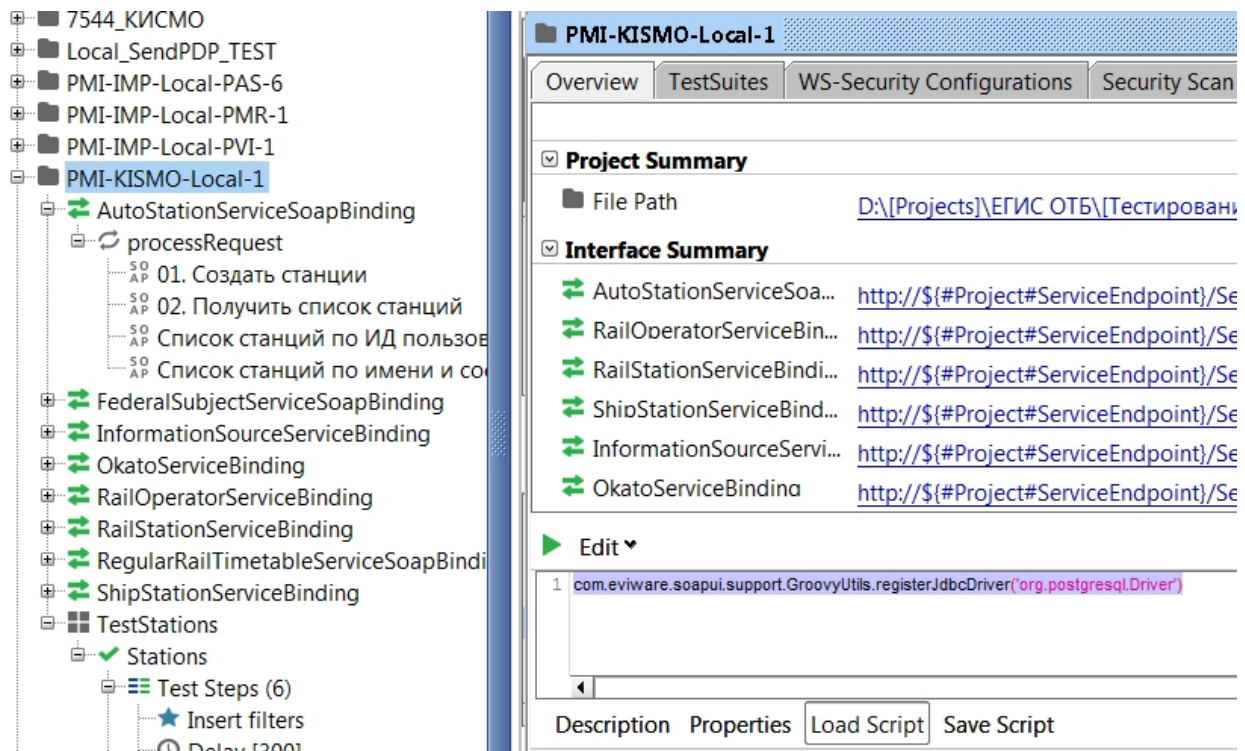
Первым шагом теста будет создание фильтров. Для этого мы будем использовать Groovy-скрипт. Groovy - это язык программирования (настройка для Java, [вики наше все](#)) . Нам пока будут нужны только самые простые конструкции, которые можно нагуглить или спросить у коллеги-программиста.

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:

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



- теперь в нашем тест-кейсе (Project/TestSuite/TestCase) создадим шаг Groovy Script (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:

```
1 /*регистрируем jdbc драйвер, файл драйвера нужно скачать
   https://jdbc.postgresql.org/download.html здесь, версия 4.2
2 и положить в папки C:\Program Files\SmartBear\SoapUI-5.3.0\lib и C:\Program
   Files\SmartBear\SoapUI-5.3.0\bin\ext*/
3 com.eviware.soapui.support.GroovyUtils.registerJdbcDriver('org.postgresql.Driver')
4 import groovy.sql.Sql
5 import java.sql.Driver
6
7 /*Подключение к БД*/
8 def sql =
   Sql.newInstance("jdbc:postgresql://192.168.70.91:5432/ORA2PG_T", "postgres", "pos
   tgres", "org.postgresql.Driver")
9
10 /*Выполняем скрипты между ''' (тройными одинарными кавычками)*/
11 sql.execute '''
12 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('231', '2017-04-03 00:00:00',
   '2018-04-03 00:00:00', '0', '1', '^КИСМО АВТОСТАНЦИЯ ЗСС 01$', '2');
13 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('232', '2017-04-03 00:00:00',
   '2018-04-03 00:00:00', '0', '2', '^КАЗСС03$', '2');
14 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('233', '2017-04-03 00:00:00',
   '2018-04-03 00:00:00', '0', '3', '^KADSS05$', '2');
15 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('234', '2017-04-03 00:00:00',
   '2018-04-03 00:00:00', '0', '4', '^ВЯЗЬМА$', '2');
```

```

16 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('235', '2017-04-03 00:00:00',
17 '2018-04-03 00:00:00', '0', '5', '^24403000000$', '2');
18 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('236', '2017-04-03 00:00:00',
19 '2018-04-03 00:00:00', '0', '6', '^РЕСПУБЛИКА КОМИ$', '2');
20 '''

```

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 обзвала их:

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

Скрипт Groovy немного изменим, добавив вызов этих переменных:

```

1  /*Регистрируем jdbc драйвер, файл драйвера нужно скачать
2     https://jdbc.postgresql.org/download.html здесь, версия 4.2
3     и положить в папки C:\Program Files\SmartBear\SoapUI-5.3.0\lib и C:\Program
4     Files\SmartBear\SoapUI-5.3.0\bin\ext*/
5  com.eviware.soapui.support.GroovyUtils.registerJdbcDriver('org.postgresql.Driver')
6
7  import groovy.sql.Sql
8  import java.sql.Driver
9
10 /*Здесь мы объявляем переменные, значения которых достаем из свойств проекта (у
11    меня переменные и в скрипте,
12    и в свойствах проекта названы одинаково*/
13 def pgCon = testRunner.testCase.testSuite.project.getPropertyValue( "pgCon" )
14 def pgUser = testRunner.testCase.testSuite.project.getPropertyValue( "pgUser" )
15 def pgPass = testRunner.testCase.testSuite.project.getPropertyValue( "pgPass" )
16
17 /*Здесь мы просто указываем объявленные ранее переменные*/
18 def sql = Sql.newInstance(pgCon,pgUser,pgPass,"org.postgresql.Driver")
19
20 sql.execute '''
21 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('231', '2017-04-03 00:00:00',
22 '2018-04-03 00:00:00', '0', '1', '^КИСМО АВТОСТАНЦИЯ ЗСС 01$', '2');
23 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('232', '2017-04-03 00:00:00',
24 '2018-04-03 00:00:00', '0', '2', '^КАЗСС03$', '2');

```

```

19 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('233', '2017-04-03 00:00:00',
    '2018-04-03 00:00:00', '0', '3', '^KADSS05$', '2');
20 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('234', '2017-04-03 00:00:00',
    '2018-04-03 00:00:00', '0', '4', '^ВЯЗЬМА$', '2');
21 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('235', '2017-04-03 00:00:00',
    '2018-04-03 00:00:00', '0', '5', '^24403000000$', '2');
22 INSERT INTO "GENERAL_ONSI"."FILTER" VALUES ('236', '2017-04-03 00:00:00',
    '2018-04-03 00:00:00', '0', '6', '^РЕСПУБЛИКА КОМИ$', '2');
23 ''

```

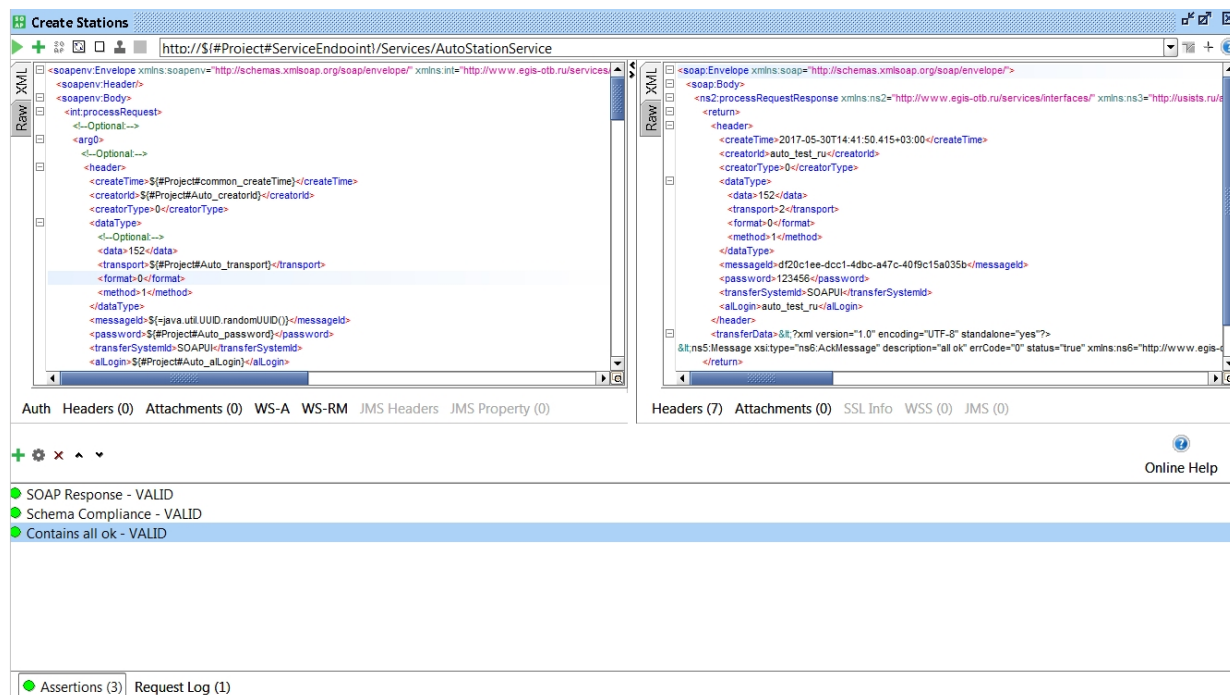
Так как вообще-то этот шаг по сути не входит в тест, правильнее было бы его записать в Setup Script (два раза кликнуть по Test Steps, в нижней панели будет вкладка Setup Script - туда можно записывать все скрипты, которые нужно выполнить перед началом теста - например загрузить тестовые данные). Но для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

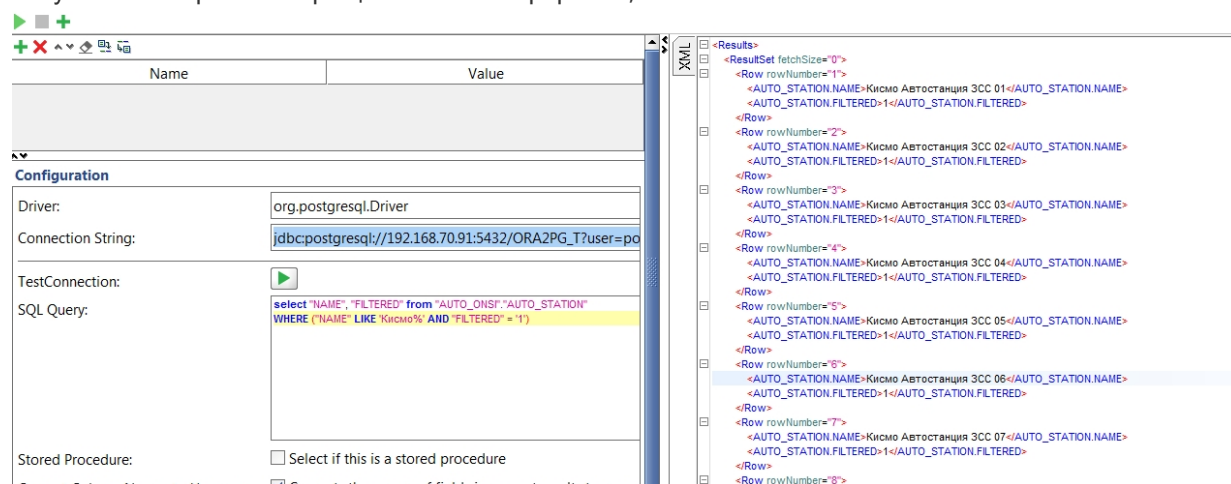
В моем случае есть несколько индикаторов успешности выполнения (это зависит от теста, данных, которые записываются, и ожидаемого результата), например:

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.Driver
- **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:

```
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, используя 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.

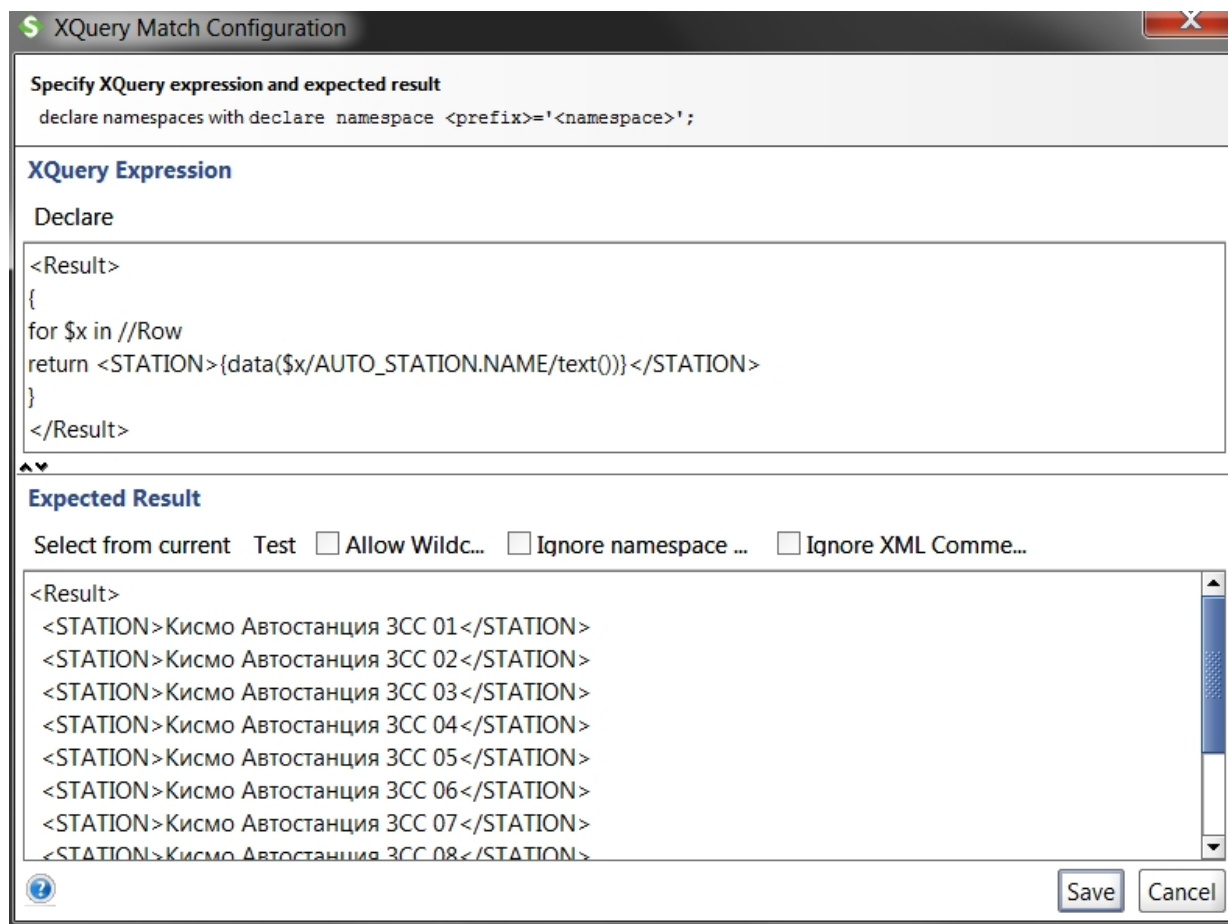
Следующая проверка будет скриптом (Script > Script Assertion). Эту же проверку так же можно разбить на 6 простых проверок (Property Content > Contains) с именем каждой станции, но я хочу все сразу. Скрипт содержит все имена станций, который должны присутствовать в результатах:

```
1 responseData = messageExchange.getResponseContent()
2 log.info(responseData)
3 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 01</AUTO_STATION.NAME>")
4 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 03</AUTO_STATION.NAME>")
5 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 05</AUTO_STATION.NAME>")
6 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 07</AUTO_STATION.NAME>")
7 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 09</AUTO_STATION.NAME>")
8 assert responseData.contains("<AUTO_STATION.NAME>Кисмо Автостанция ЗСС 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:

```
1 <Result>
2 {
3   for $x in //Row
4   return <STATION>{data($x/AUTO_STATION.NAME/text())}</STATION>
5 }
6 </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):

```

1 <Result>
2   <STATION>Кисмо Автостанция ЗСС 01</STATION>
3   <STATION>Кисмо Автостанция ЗСС 03</STATION>
4   <STATION>Кисмо Автостанция ЗСС 05</STATION>
5   <STATION>Кисмо Автостанция ЗСС 07</STATION>
6   <STATION>Кисмо Автостанция ЗСС 09</STATION>
7   <STATION>Кисмо Автостанция ЗСС 11</STATION>
8 </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:

The screenshot shows a web service testing interface. On the left, the 'Configuration' tab is active, showing the following settings:

- Driver: org.postgresql.Driver
- Connection String: jdbc:postgresql://192.168.70.91:5432
- TestConnection: [button]
- SQL Query: select "NAME", "FILTERED" from "AUTO_ON" WHERE ("NAME" LIKE "Кисмо%" AND "FILTERED" = 1)

On the right, the 'XML' tab shows the response structure:

```

<Results>
  <ResultSet fetchSize="10">
    <Row rowNumber="1">
      <AUTO_STATION.NAME>Кисмо Автостанция ЗСС 01</AUTO_STATION.NAME>
      <AUTO_STATION.FILTERED>1</AUTO_STATION.FILTERED>
    </Row>
    <Row rowNumber="2">
      <AUTO_STATION.NAME>Кисмо Автостанция ЗСС 03</AUTO_STATION.NAME>
      <AUTO_STATION.FILTERED>1</AUTO_STATION.FILTERED>
    </Row>
    <Row rowNumber="3">
      <AUTO_STATION.NAME>Кисмо Автостанция ЗСС 05</AUTO_STATION.NAME>
      <AUTO_STATION.FILTERED>1</AUTO_STATION.FILTERED>
    </Row>
    <Row rowNumber="4">
      <AUTO_STATION.NAME>Кисмо Автостанция ЗСС 07</AUTO_STATION.NAME>
      <AUTO_STATION.FILTERED>1</AUTO_STATION.FILTERED>
    </Row>
    <Row rowNumber="5">
      <AUTO_STATION.NAME>Кисмо Автостанция ЗСС 09</AUTO_STATION.NAME>
      <AUTO_STATION.FILTERED>1</AUTO_STATION.FILTERED>
    </Row>
  </ResultSet>
</Results>

```

At the bottom, the 'Test Results' tab shows the following status:

- JDBC Status - VALID
- There are some results - VALID
- At least 6 results - FAILED
- > Missing token [Row rowNumber="6"] in Response
- Expected stations filtered - VALID
- Not contain unfiltered stations, contains all filtered - FAILED
- > XQuery Match Assertion failed for path [<Result>{for \$x in //Row return <STATION>{data(\$x/AUTO_STATION.NAME/text())}</STATION>}</Result>} : Exception: c

Below the test results, there are links for 'Assertions (11)' and 'Request Log (2)'.

Примеры выполнения конкретных операций

Разбор XML-файла ответа веб-сервиса

Tip: извлечение xml из transferData в одну строку

```

1 transferData = new XmlHolder((new
  XmlHolder(messageExchange.getResponseContentAsXml()))
  .getNodeValue("//*:transferData"))
2 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:

```
1  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2    <soap:Body>
3      <ns2:processRequestResponse xmlns:ns2="http://www.egis-
4      otb.ru/services/interfaces/" xmlns:ns3="http://usists.ru/asteros/data">
5        <return>
6          <header>
7            <createTime>2017-06-01T11:54:31.543+03:00</createTime>
8            <creatorId>auto_test_ru</creatorId>
9            <creatorType>0</creatorType>
10           <dataType>
11             <data>151</data>
12             <transport>2</transport>
13             <format>0</format>
14             <method>4</method>
15           </dataType>
16           <messageId>d4269757-0f32-4034-9eb4-4385eeff14d4</messageId>
17           <password>123456</password>
18           <transferSystemId>SOAPUI</transferSystemId>
19           <aLogin/>
20         </header>
21         <transferData><![CDATA[<?xml version="1.0" encoding="UTF-8"
22         standalone="yes"?>
23         <ns4:Message xsi:type="ns5:DataMessage" xmlns:ns6="http://www.egis-
24         otb.ru/data/onsi/operators/" xmlns:ns5="http://www.egis-otb.ru/messaging/"
25         xmlns:ns2="http://www.egis-otb.ru/data/timetable/delta/"
26         xmlns:ns4="http://www.egis-otb.ru/datatypes/" xmlns:ns3="http://www.egis-
27         otb.ru/data/onsi/rail/countries/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
28         instance">
29           <dataArray recordCount="3">
```

```

23      <data xsi:type="ns6:Operator" egisId="22071" contactDepartment="Отдел
транспортной безопасности" contactEmail="otb1@transport.ru"
contactFax="84955086666" contactLastName="Ларионов" contactName="Леонид"
contactPerson="Заместитель директора" contactPhone="84951236666"
contactPost="143456, г. Саратов, ул. Ленина, д. 2, вл. 5"
contactSurname="Васильевич" egrul="5077746887312" latName="Operator 1"
lawAddress="143456, г. Саратов, ул. Ленина, д. 2, вл. 5" name="Перевозчик 1"
shortName="ПЕРЕВ1" state="active" isgid="1281" isuid="1883"
sourceId="testPmiPasA06201" id="25112">
24      <actualPeriod xsi:type="ns4:DateTimePeriod" from="2017-05-
25T10:34:00.000Z" to="2018-06-01T10:34:00.000Z"/>
25      </data>
26      <data xsi:type="ns6:Operator" egisId="22072" contactDepartment="Отдел
транспортной безопасности" contactEmail="otb2@transport.ru"
contactFax="84955086666" contactLastName="Сидоров" contactName="Петр"
contactPerson="Заместитель директора" contactPhone="84951236666"
contactPost="143456, г. Москва, ул. Ленина, д. 2, вл. 5"
contactSurname="Васильевич" egrul="5077746887312" latName="Operator 2"
lawAddress="143456, г. Москва, ул. Ленина, д. 2, вл. 5" name="Перевозчик 2"
shortName="ПЕРЕВ2" state="active" isgid="1281" isuid="1883"
sourceId="testPmiPasA06202" id="25113">
27      <actualPeriod xsi:type="ns4:DateTimePeriod" from="2017-05-
25T10:34:00.000Z" to="2018-06-01T10:34:00.000Z"/>
28      </data>
29      <data xsi:type="ns6:Operator" egisId="22073" contactDepartment="Отдел
транспортной безопасности" contactEmail="otb3@transport.ru"
contactFax="84955086666" contactLastName="Иванов" contactName="Леонид"
contactPerson="Заместитель директора" contactPhone="84951236666"
contactPost="143456, г. Самара, ул. Ленина, д. 2, вл. 5"
contactSurname="Васильевич" egrul="5077746887312" latName="Operator 3"
lawAddress="143456, г. Самара, ул. Ленина, д. 2, вл. 5" name="Перевозчик 3"
shortName="ПЕРЕВ3" state="active" isgid="1281" isuid="1883"
sourceId="testPmiPasA06203" id="25114">
30      <actualPeriod xsi:type="ns4:DateTimePeriod" from="2017-05-
25T10:34:00.000Z" to="2018-06-01T10:34:00.000Z"/>
31      </data>
32      </dataArray>
33      </ns4:Message>]]></transferData>
34      </return>
35      </ns2:processRequestResponse>
36      </soap:Body>
37      </soap:Envelope>

```

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 `//*[@transferData]` и `@shortName`) описано ниже.

Код для извлечения значений атрибутов из XML:

```
1  /*Импортируем две нужные библиотеки для работы с XML*/
2  import com.eviware.soapui.support.XmlHolder
3  import com.eviware.soapui.support.GroovyUtils
4
5  /* Создаем переменную responseXmlHolder, указываем,
6  что содержимое - XML. В переменную записываем возвращенный от сервиса ответ */
7  responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
8
9  /*Из предыдущей переменной с ответом от сервиса извлекаем только данные между
transferData
10 и записываем их в переменную cdataXml.
11 log.info в консоли покажет извлеченные данные */
12 cdataXml = responseXmlHolder.getNodeValue("//*[@transferData]")
13 log.info (" Извлеченный CDATA: " + cdataXml)
14
15 /* Еще одна переменная, которая содержит XML данные, на этот раз туда
записываем
16 извлеченные из transferData данные */
17 cdataXmlHolder = new XmlHolder(cdataXml)
18
19 /* Здесь мы объявляем переменную recordCount, в которую записываем значение,
20 извлеченное из поля <dataArray recordCount="3">. Команда assert == 3 проверяет,
21 что это значение равно 3 (должно быть возвращено 3 записи).*/
22 recordCount = cdataXmlHolder.getNodeValue("//*[@dataArray/@recordCount]")
23 assert recordCount == "3"
24 log.info (" Количество записей: " + recordCount)
25
26 /* Узнаем имя перевозчика (указанное в атрибуте shortName поля data.
27 Эта операция выведет имя перевозчика первого узла, а у нас их recordCount=3*/
28 operatorName1 = cdataXmlHolder.getNodeValue("//*[@dataArray/data/@shortName]")
29 log.info ( "Имя перевозчика 1: " + operatorName1)
30
31 /* А тут мы добавляем указание на конкретное поле data (то, что
32 содержит указанный sourceId, и извлекаем его краткое имя shortName*/
33 operatorName2 =
34   cdataXmlHolder.getNodeValue("//*[@dataArray/data[@sourceId='testPmiPasA06202']/@
shortName")
34 assert operatorName2 == "ПЕРЕВ2"
35 log.info ( "Имя перевозчика 2: " + operatorName2)
```

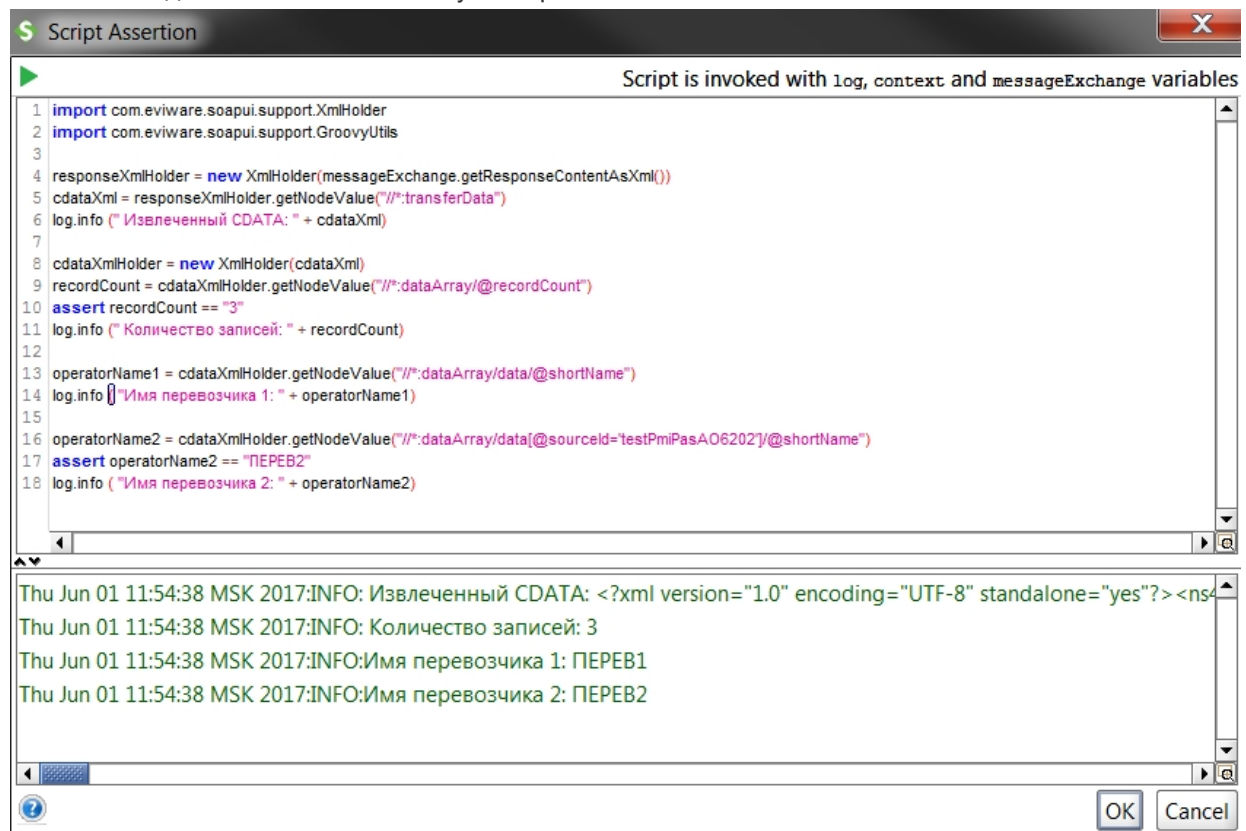
Код для получения `id` из XML:

```

1 import com.eviware.soapui.support.XmlHolder
2 import com.eviware.soapui.support.GroovyUtils
3
4 responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
5 cdataXml = responseXmlHolder.getNodeValue("//*:transferData")
6
7 cdataXmlHolder = new XmlHolder(cdataXml)
8
9 tsId = cdataXmlHolder.getNodeValue("//*:dataArray/data/@id")
10 def testCaseProperty =
    messageExchange.modelItem.testStep.testCase.testSuite.setPropertyValue(
        "OperAutoId", tsId )

```

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

-----внутри которой папки

и .

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

```
1 /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:

```
1 | /**: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:

```
1 | /**: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:

```
1 | /**:dataArray/data[@sourceId='testPmiPasA06202']/@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:

```
1 | <data xsi:type="ns7:AutoPDP" status="1" .....>
2 |     ...
3 |     <surname value="КисмоАвтоГ"/>
4 |     ...
5 | </data>
```

Вот так:

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

Автоматическая сериализация в base64 в запросе

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.

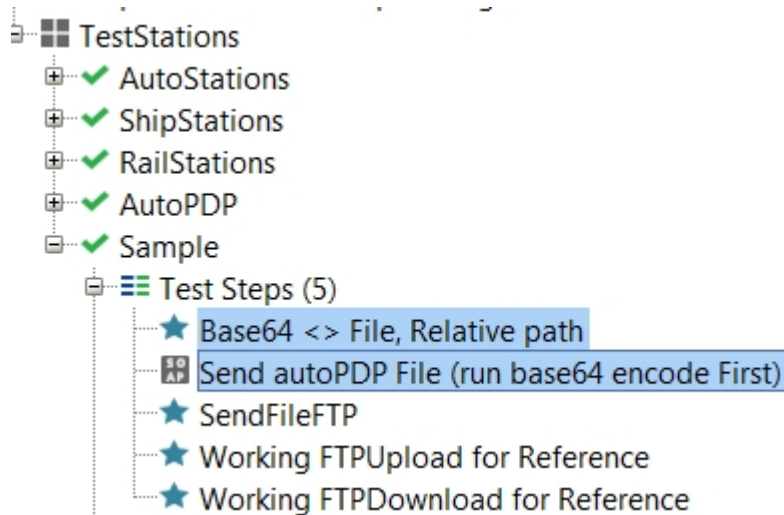
Для начала поймем, что нужно сделать:

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 закодированную 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:

```

1  --папка SoapUI <<< тут лежит файл проекта
2  ----testFiles <<< это общая папка для всех тестовых файлов
3  -----kismoPMI <<< папка для определенной задачи
4  -----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`

```

1  import com.eviware.soapui.support.GroovyUtils
2
3  /*Прочитаем значение переменной testFilesFolder в Custom Properties проекта */
4
5  def testFilesFolder = testRunner.testCase.testSuite.project.getPropertyValue(
6    "testFilesFolder" )
7
8  log.info(" Путь к папке с тестовыми файлами относительно файла проекта: " +
9    testFilesFolder)
10
11 /*Эти две строчки прочитают путь до папки с проектом и запишут его в переменную
12 projectPath*/
13 def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
14 def projectPath = groovyUtils.projectPath
15 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.csv`). В результате скрипт прочитает файл, расположенный на компьютере в папке `/<путь к папке проекта>/testFiles/kismoPMI/auto/pdp.csv`.

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

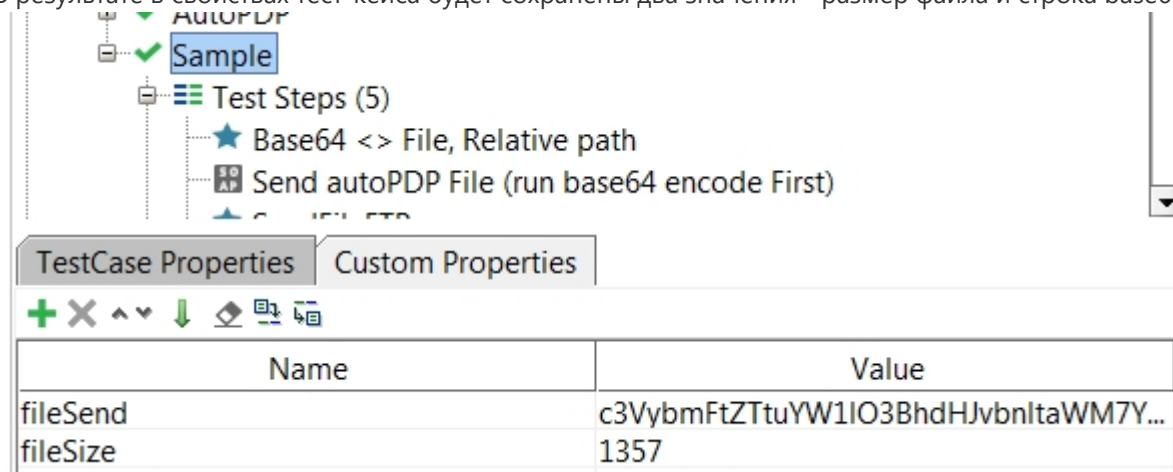
В запросе SOAP, который отправляет файл, также указывается размер файла. Там может быть случайное число, можно его не менять. Но можно получить размер нашего файла и записать его в переменную `fileSize` (переменная в **Custom Properties** тест-кейса):

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

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

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

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



Name	Value
fileSend	c3VybmFtZTtuYW1lO3BhdHJvbnltaWM7Y...
fileSize	1357

Целиком скрипт:

```
1 import com.eviware.soapui.support.GroovyUtils
2
3 /*
4  * Получаем относительный путь к папке с тестовыми файлами, testFilesFolder -
5  * свойство проекта, где указан путь
6  * к папке относительно файла проекта
7  */
```

```

7  def testFilesFolder = testRunner.testCase.testSuite.project.getPropertyValue(
   "testFilesFolder" )
8  log.info(" Путь к папке с тестовыми файлами относительно файла проекта: " +
   testFilesFolder)
9  def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
10 def projectPath = groovyUtils.projectPath
11 log.info(" Путь к папке проекта на диске: "+ projectPath)
12
13 /*
14  * Читаем файл, расположенный относительно файла проекта в папке, путь к
   которой
15  * указан в свойствах проекта (в данном случае /testFiles/auto/pdp.csv
16  */
17 def pdpFile = new File(projectPath + testFilesFolder, "/pdp.csv"
   ).getText("UTF-8");
18 log.info(" Содержимое файла: " + pdpFile)
19
20 /* Определяем размер файла, записываем его в свойста тест-кейса fileSize*/
21
22 def fileSize = pdpFile.length().toString()
23 testRunner.testCase.setPropertyValue( "fileSize", fileSize )
24
25 /*Берем содержимое файла, пишем в кодированную строку,
26 и записываем ее в Properties этого тест-кейса (Sample)*/
27 /*def pdpFileEncoded = pdpFile.bytes.encodeBase64().toString()* <<
   кракозябрики*/
28
29 String pdpFileEncoded = pdpFile.getBytes( 'UTF-8' ).encodeBase64()
30 testRunner.testCase.setPropertyValue( "fileSend", pdpFileEncoded )

```

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

```

1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:pman="http://www.egis-otb.ru/general/pmanager">
2    <soapenv:Header/>
3    <soapenv:Body>
4      <pman:parseFile>
5        <!--Optional:-->
6        <arg0>
7          <!--Optional:-->
8          <header>
9            <createTime>${#Project#CreateTime}</createTime>
10           <creatorId>auto_test_ru</creatorId>
11           <creatorType>0</creatorType>
12           <dataType>
13             <!--Optional:-->
14             <data>1</data>
15             <transport>${#Project#autoTransport}</transport>
16             <format>0</format>
17             <method>1</method>
18           </dataType>
19           <messageId>${=java.util.UUID.randomUUID()}</messageId>

```

```

20         <password>123456</password>
21         <transferSystemId>SOAPUI</transferSystemId>
22         <aLogin>auto_test_ru</aLogin>
23     </header>
24     <transferData><![CDATA[<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
25 <ns10:Message xsi:type="ns6:FileMessage" source="soapUI"
xmlns:ns5="http://www.egis-otb.ru/data/onsi/rail/countries/"
xmlns:ns6="http://www.egis-otb.ru/messaging/" xmlns:ns7="http://www.egis-
otb.ru/data/pdp/" xmlns:ns8="http://www.egis-otb.ru/query/"
xmlns:ns10="http://www.egis-otb.ru/datatypes/" xmlns:ns9="http://www.egis-
otb.ru/messaging/acs/commands/" xmlns:ns11="http://www.egis-otb.ru/logs/"
xmlns:ns12="http://www.egis-otb.ru/acs/errors/" xmlns:ns2="http://www.egis-
otb.ru/data/timetable/delta/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:ns3="http://www.egis-otb.ru/requests/"
xmlns:ns4="http://usists.ru/asteros/data">
26     <fileData>${#TestCase#fileSend}</fileData>
27     <fileInfo xsi:type="ns10:FileInfoExtended" rewrite="true" ackSrc="5"
size="${#TestCase#fileSize}" format="plain:auto-csv:1" fileName="autoPDP.csv"
createdAt="${=import java.text.SimpleDateFormat; Calendar cal =
Calendar.getInstance(); cal.add(Calendar.HOUR, -2); new SimpleDateFormat("yyyy-
MM-dd'T'HH:mm'Z'").format(cal.getTime());}"
archiveId="${=java.util.UUID.randomUUID()}"/>
28 </ns10:Message>]]></transferData>
29     </arg0>
30 </pman:parseFile>
31 </soapenv:Body>
32 </soapenv:Envelope>

```

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

- `<fileData>${#TestCase#fileSend}</fileData>` - здесь мы достаём нашу сериализованную скриптом строку из переменной `fileSend` и вставляем в `fileData`
- `size="${#TestCase#fileSize}"` - берём посчитанный скриптом размер файла из переменной `fileSize`
- `fileName =` - так как обработчику неважно имя, можно не менять. А можно тоже сгенерировать автоматически;
- `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):

```
1 [user_test_ru]
2 ReadWrongRecPBD = 1
3 ReadAccessPVB = 1
4 ReadLogPBD = 1
5 WriteAccessPVB = 1
6 EmergencyStartPVB = 1
7 WriteWrongRecPBD = 1
8 ReadLogPVB = 1
9 ReadPDPAccessPBD = 1
10 ControlACS = 1
11 CheckRec = 1
12 SearchAccessPVB = 1
13 WriteAccessPBD = 1
14 IntermediaryAccess = 1
15 Password = 123456
```

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 запишем эти данные в читаемом виде и закодируем их скриптом в переменную. Вот так:

```
1 import com.eviware.soapui.*
2
3 // Строка как она есть построчно пишется в тройных двойных кавычках"
4 def userFile = """"[user_test_ru]
5 ReadWrongRecPBD = 1
6 ReadAccessPVB = 1
7 ReadLogPBD = 1
8 WriteAccessPVB = 1
9 EmergencyStartPVB = 1
10 WriteWrongRecPBD = 1
11 ReadLogPVB = 1
12 ReadPDPAccessPBD = 1
13 ControlACS = 1
14 CheckRec = 1
15 SearchAccessPVB = 1
16 WriteAccessPBD = 1
17 IntermediaryAccess = 1
18 Password = 123456""""
19
20 String userFileEncoded = userFile.getBytes( 'UTF-8' ).encodeBase64()
21 testRunner.testCase.setPropertyValue( "userFile", userFileEncoded )
```

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

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



```

2  /*      Сериализация в строку base64 и сохранение в переменную тест-кейса
    nameOfProperty
3      Контекст: teststep. Для использования в assertion заменить
4      testRunner.testCase.setPropertyValue на
5      messageExchange.modelItem.testStep.testCase.setPropertyValue*/
6
7  def base64This(String prop, String input) {
8      String encoded = input.getBytes( 'UTF-8' ).encodeBase64()
9      testRunner.testCase.setPropertyValue( "${prop}", encoded )
10 }
11
12 log.info ("Результат: " + base64This("nameOfProperty",
13 """"строка или
14 строки для кодирования""""))
15 log.info ("Результат: " + base64This("nameOfProperty","А это просто одна
    строчка для кодирования"))

```

В запросе просто забираем 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):

```

1  import com.eviware.soapui.support.XmlHolder
2  import com.eviware.soapui.support.GroovyUtils
3
4  //Извлекаем ответ из transferData
5  responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
6  dataXml = responseXmlHolder.getNodeValue("//*:transferData")
7  log.info ("Извлеченные данные: " + dataXml)
8
9  String decoded = new String(dataXml.decodeBase64(), "UTF-8")
10 log.info ("Декодировано: " + decoded)

```

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 x не найден".

```

1  import com.eviware.soapui.support.XmlHolder
2  import com.eviware.soapui.support.GroovyUtils

```

```

3
4 //сравнение строк посимвольно (так как строки могут быть возвращены в разном
   порядке)
5 def SortString(String input) {
6     char[] charArray = input.replaceAll("[^a-zA-Z0-9
   ]+", "").toCharArray()
7     Arrays.sort(charArray)
8     String sortedString = new String(charArray)
9
10    return sortedString.trim()
11 }
12
13 responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
14 dataXml = responseXmlHolder.getNodeValue("//*:transferData")
15
16 if (!dataXml.contains("не найден")) {
17     String responseDecoded = new String(dataXml.decodeBase64(), "UTF-8")
18     //log.info ("Декодирован ответ: " + responseDecoded)
19     def userFile =
messageExchange.modelItem.testStep.testCase.getPropertyValue( "userFile" )
20     String userFileDecoded = new String(userFile.decodeBase64(), "UTF-8")
21     //log.info ("Декодировано что отправляли: " + userFileDecoded)
22
23     assert SortString(responseDecoded) == SortString(userFileDecoded)
24 }
25 else {
26     assert !dataXml.contains("не найден"), "Пользователь не найден"
27 }

```

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 библиотеке 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 ниже. 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 файлу в свойствах проекта и имя файла (его тоже можно в свойства вынести, чтобы вообще скрипт не менять).

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:

```
1  import org.apache.commons.net.ftp.FTP
2  import org.apache.commons.net.ftp.FTPClient
3  import java.time.LocalDateTime
4  import java.time.format.DateTimeFormatter
5  import java.util.concurrent.ThreadLocalRandom
6  import java.io.File
7
8  // Параметры подключения к FTP-серверу записаны в одной переменной в формате
   "ftpHost, ftpPort, ftpUser, ftpPassword"
9  // Например: 192.168.70.95,21021,auto_test_ru,123456
10 ftpConString = testRunner.testCase.getPropertyValue( "ftp" )
11
12 // Путь к папке с этим проектом
13 def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
14 def projectPath = groovyUtils.projectPath
15
16 /* Метод для загрузки файла на FTP
17  *
18  * @param ftpConString String Строка подключения к FTP (в формате "ftpHost,
   ftpPort, ftpUser, ftpPassword")
19  * @param folder String Папка, откуда берем файл
20  * @param fileName String Имя файла на ФС, который будем переименовывать и
   загружать
21  * @param ext String Разрешения файла ("xml","csv" и т.п.). Без точки
22  *
23  */
24 def uploadToFtp(String ftpConString, String folder, String fileName, String
   ext) {
25     def (ftpHost, ftpPort, ftpUser, ftpPassword) = ftpConString.split(",")
26     //генерируем имя файла за последние 12 ч
```

```

27     uploadFileName = "20000_" +
28         LocalDateTime.now()
29         .minusHours(new ThreadLocalRandom().nextInt(12))
30
31     .format(DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss_SSS")).toString() +
32     "." + ext
33     try {
34         // Читаем файл из папки и складываем в поток
35         File localFile = new File(folder + "/" + fileName)
36         InputStream inputStream = new FileInputStream(localFile)
37
38         FTPClient ftpClient = new FTPClient()
39
40         ftpClient.connect(ftpHost, Integer.parseInt(ftpPort))
41         ftpClient.login(ftpUser, ftpPassword)
42         ftpClient.enterLocalPassiveMode()
43         ftpClient.setFileType(FTP.BINARY_FILE_TYPE)
44
45         //Загрузить файл с именем uploadFileName
46         boolean done = ftpClient.storeFile(uploadFileName, inputStream)
47         inputStream.close()
48         if (done) {
49             log.info("Файл успешно загружен на сервер ${ftpHost}:${ftpPort}
50 с именем ${uploadFileName}")
51         }
52     } catch (IOException ex) {
53         log.error("Ошибка: " + ex.getMessage())
54         ex.printStackTrace()
55     } finally {
56         try {
57             if (ftpClient.isConnected()) {
58                 ftpClient.logout()
59                 ftpClient.disconnect()
60             }
61         } catch (IOException ex) {
62             ex.printStackTrace()
63         }
64     }
65
66     // Вызов метода. Указываем строку подключения как строку (или переменную), путь
67     к папке (или переменную) как строку
68     // имя файла, который возьмем с ФС и расширение
69     uploadToFtp(ftpConString, projectPath, "upload.csv", "csv")

```

Генерация случайных строчных данных на примере ФИО

Есть несколько способов генерации случайных данных, мы рассмотрим два: случайный набор букв/цифр/символов или случайный выбор из подобранного набора данных. Первый способ проще реализовать, но генерируется часто нечитаемый набор, типа "Гривалжсва". 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 заикливать, это описано ниже.

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 случайной 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:

```
1 // Генерация случайных строк
2
3 def generator = { String alphabet, int n ->
4     new Random().with {
5         (1..n).collect { alphabet[ nextInt( alphabet.length() ) ] }.join()
6     }
7 }
8
9 //Транслитерация TO DO: проверить на соответствия правилам ЕГИС
10 String transliterate(String message){
11     char[] abcCyr = ['а','б','в','г','д','е','ё',
12         'ж','з','и','й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ч',
13         'ш','щ','ъ','ы','ь','э','ю','я',
14         'А','Б','В','Г','Д','Е','Ё',
15         'Ж','З','И','Й','К','Л','М','Н','О','П','Р','С','Т','У','Ф','Х','Ц','Ч','Ш',
16         'Щ','Ъ','Ы','Ь','Э','Ю','Я',
17
18         'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t',
19         'u','v','w','x','y','z'
20
21         , 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T',
22         'U','V','W','X','Y','Z']
23     String[] abcLat =
24         ["A","Б","В","Г","Д","Е","Е","Ж","З","И","Й","К","Л","М","Н","О","П","Р","С","Т",
25         ",","У","Ф","Х","Ц","Ч","Ш","Щ","Ъ","Ы","Ь","Э","Ю","Я",
26
27         "A","Б","В","Г","Д","Е","Е","Ж","З","И","Й","К","Л","М","Н","О","П","Р","С","Т",
28         ",","У","Ф","Х","Ц","Ч","Ш","Щ","Ъ","Ы","Ь","Э","Ю","Я",
29
30         "A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T",
31         ",","U","V","W","X","Y","Z",
32
33         "A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T",
34         ",","U","V","W","X","Y","Z"]
```

```

19     StringBuilder builder = new StringBuilder()
20     for (int i = 0; i < message.length(); i++) {
21         for (int x = 0; x < abcCyr.length; x++ ) {
22             if (message.charAt(i) == abcCyr[x]) {
23                 builder.append(abcLat[x])
24             }
25         }
26     }
27     return builder.toString()
28 }
29
30 pdName = (generator((( 'a'..'и')+( 'к'..'щ')+( 'э'..'я'))).join(), 9
31 )).capitalize()
32 pdSurname = (generator((( 'a'..'и')+( 'к'..'щ')+( 'э'..'я'))).join(), 9
33 )).capitalize()
34 pdPatronymic = (generator((( 'a'..'и')+( 'к'..'щ')+( 'э'..'я'))).join(), 9
35 )).capitalize()
36
37 testRunner.testCase.setPropertyValue("Name",pdName)
38 testRunner.testCase.setPropertyValue("Surname",pdSurname)
39 testRunner.testCase.setPropertyValue("Patronymic",pdPatronymic)
40 testRunner.testCase.setPropertyValue("NameNormal",transliterate(pdSurname))
41 testRunner.testCase.setPropertyValue("SurnameNormal",transliterate(pdName))
42 testRunner.testCase.setPropertyValue("PatronymicNormal",transliterate(pdPatronymic))
43
44 log.info ("ФИО: " + pdSurname + " " + pdName + " " + pdPatronymic )

```

Пример сгенерированных ФИО: Цсщвлпрхп Тткимщщс Ютшудчжкс. Не очень созвучно, но уж что есть. Также при записи в переменную можно добавлять какой-нибудь префикс, например "Тест", чтобы позже находить данные 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 отчества;
- записываем в свойства проекта рандомно выбранные ФИО.

Выбор случайного элемента из списка:

```

1  import java.util.concurrent.ThreadLocalRandom
2
3  //Выбор случайного значения из массива на примере ФИО
4
5  names =
6  ["Бирабиджан", "Наколюк", "Миритафузий", "Аленгодок", "Мивираил", "Леронад", "Ихтигюк",
7  "Кпеорад", "Инирал", "Бекмамбет", "Рокуват", "Синарит", "Золофт", "Лакмита", "Сероквель",
8  "Кветиапин", "Пиносол", "Маример",
9  "Аквамарис", "Граммиддин", "Септолет", "Спазмалгон"]
10
11 surnames =
12 ["Диранов", "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
13 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
14 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
15 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
16 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
17 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
18 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
19 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
20 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
21 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
22 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
23 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
24 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
25 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
26 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
27 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
28 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
29 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
30 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
31 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
32 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
33 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
34 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
35 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
36 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
37 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
38 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
39 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
40 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
41 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
42 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
43 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
44 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
45 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
46 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
47 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
48 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
49 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
50 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
51 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
52 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
53 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
54 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
55 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
56 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
57 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
58 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
59 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
60 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
61 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
62 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
63 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
64 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
65 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
66 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
67 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
68 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
69 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
70 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
71 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
72 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
73 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
74 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
75 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
76 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
77 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
78 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
79 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
80 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
81 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
82 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
83 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
84 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
85 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
86 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
87 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
88 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
89 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
90 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
91 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
92 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
93 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
94 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
95 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
96 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
97 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
98 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
99 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",
100 "Кемералюк", "Вирифанов", "Понураев", "Белетореев", "Сиктавкаров",

```



```

9  "Борокелев", "дамаранов", "Искандеров", "колотузинов", "Бомбушкин", "Кравчук",
   "Желторотиков",
10 "Кавтноров",
   "Берамидтинов", "канареейков", "желопызин", "Цикролюк", "Вирифтеев", "Туеноров", "Хро
   комаров", "Жерониев"]
11 patronymics = ["Бирабиджанов", "Наколюкович", "Аленгодокович", "Леронадович",
   "Мирифтеевич",
12 "Кнеопардович",
   "Ерифтеевич", "Инимавович", "Беронорович", "Жеротдинович", "Кемерович", "Инитуарович
   ", "Берогорович", "Илиносивич", "Геромаронович"]
13
14 //ThreadLocalRandom вместо Random для производительности
15 pdName = names[ThreadLocalRandom.current().nextInt(names.size)]
16 pdSurname = surnames[ThreadLocalRandom.current().nextInt(surnames.size)]
17 pdPatronymic =
   patronymics[ThreadLocalRandom.current().nextInt(patronymics.size)]
18
19 log.info ("ФИО: " + pdSurname + " " + pdName + " " + pdPatronymic)
20
21 testRunner.testCase.setPropertyValue("Name", pdName)
22 testRunner.testCase.setPropertyValue("Surname", pdSurname)
23 testRunner.testCase.setPropertyValue("Patronymic", pdPatronymic)

```

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 случайных 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 случайного числа между указанными значениями (`min ... max`) выполняется по формулам:

```

1  # с java 1.7
2  import java.util.concurrent.ThreadLocalRandom
3
4  ThreadLocalRandom.current().nextInt(min, max + 1)
5
6  # до Java 1.7
7  Math.random()*(max - min)+min

```

Например, случайное число между 5 и 10:

```

1 # с java 1.7
2 ThreadLocalRandom.current().nextInt(5, 10 + 1)
3
4 # до Java 1.7
5 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:

```

1 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:

```

1 // с java 1.7
2 ${=import java.util.concurrent.ThreadLocalRandom; (int)
  (ThreadLocalRandom.current().nextInt(1000, 1900+1))}
3
4 // до Java 1.7
5 ${=(int)(Math.round((Math.random()*(1140 - 1001))+1001))}

```

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

```

1 <filter xsi:type="BinaryFilter" operation="eq" negate="false">
2     <attr xsi:type="Attribute" name="gid" />
3     <right xsi:type="Constant">
4         <value xsi:type="xs:long"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">${=(int)(Math.round((Math.random()*
  (1140 - 1001))+1001))}</value>
5     </right>
6 </filter>

```

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

```

1 // с java 1.7
2 import java.util.concurrent.ThreadLocalRandom
3
4 randomNumberNew = ThreadLocalRandom.current().nextInt(1000, 1900+ 1)
5 log.info ("Случайное число: " + randomNumberNew.toString())
6
7 // до Java 1.7
8 randomNumber = Math.round(Math.random()*(59 - 1) + 1)

```

Также может пригодиться 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:

```
1  import java.time.LocalDateTime
2  import java.time.Period
3  import java.time.format.DateTimeFormatter
4  import java.time.temporal.ChronoUnit
5  import java.util.concurrent.ThreadLocalRandom
6
7  // В скобках (nextInt(14)) пишем сколько дней отнять от текущей даты,
   формируем вид даты
8  // Вместо старого рандома используем ThreadLocalRandom (работает также, но
   лучше, особенно,
9  // если надо в скрипте использовать много рандомов)
10
11 date = LocalDateTime.now().minus(Period.ofDays((new
   ThreadLocalRandom().nextInt(14))))
12 DateTimeFormatter formatter =
   DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss_SSS") //здесь задаем формат
   даты
13
14 //formatter - переменная из строки выше, .format - метод, не меняется, date -
   наша переменная с датой на 2 строки выше
15 log.info (formatter.format(date).toString())
16
17 // Кстати, длинные строки можно переносить. Вот так:
18 uploadFileName = "20000_" +
19     LocalDateTime.now()
20     .minusHours(new ThreadLocalRandom().nextInt(12))
21     .format(DateTimeFormatter.ofPattern("yyyy_MM_dd_HH_mm_ss_SSS")).toString() +
   "." + ".csv"
22 log.info (uploadFileName)
```

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 шаг теста для проверки результатов). У этих скриптов 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:

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

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

Установка и получение свойств на всех уровнях:

```
1 | // Установка свойств на уровне Проекта  
2 | testRunner.testCase.testSuite.project.setPropertyValue( "projectProperty"  
    , "Значение свойства")  
3 | // Установка свойств на уровне Тест-суита  
4 | testRunner.testCase.testSuite.setPropertyValue( "testSuiteProperty" , "Значение  
    свойства")  
5 | // Установка свойств на уровне Тест-кейса  
6 | testRunner.testCase.setPropertyValue( "testCaseProperty" , "Значение свойства")  
7 |  
8 | //Получение свойства на уровне Проекта  
9 | def someVar1 = testRunner.testCase.testSuite.project.getPropertyValue(  
    "projectProperty" )  
10 | //Получение свойства на уровне Тест-суита  
11 | def someVar2 = testRunner.testCase.testSuite.getPropertyValue(  
    "testSuiteProperty" )  
12 | //Получение свойства на уровне Тест-кейса  
13 | def someVar3 = testRunner.testCase.getPropertyValue( "testCaseProperty" )
```

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

```
1 | messageExchange.modelItem.testStep.testCase.setPropertyValue( "propertyName",  
    "Привет, Мир!" )
```

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

Установка и получение свойств из script assertion:

```

1 // Установка свойств на уровне Проекта
2 messageExchange.modelItem.testStep.testCase.testSuite.project.setPropertyValue(
  "projectProperty" ,"Значение свойства")
3 // Установка свойств на уровне Тест-суита
4 messageExchange.modelItem.testStep.testCase.testSuite.setPropertyValue(
  "testSuiteProperty" ,"Значение свойства")
5 // Установка свойств на уровне Тест-кейса
6 messageExchange.modelItem.testStep.testCase.setPropertyValue(
  "testCaseProperty" ,"Значение свойства")
7
8 //Получение свойства на уровне Проекта
9 def someVar4 =
  messageExchange.modelItem.testStep.testCase.testSuite.project.getPropertyValue(
    "projectProperty" )
10 //Получение свойства на уровне Тест-суита
11 def someVar5 =
  messageExchange.modelItem.testStep.testCase.testSuite.getPropertyValue(
    "testSuiteProperty" )
12 //Получение свойства на уровне Тест-кейса
13 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:

```

1 def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
2 def projectPath = groovyUtils.projectPath
3 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 сохраняться 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:

```

1 def testFilesFolder = ( '\\files\\pmi\\' )
2 def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
3 def projectPath = groovyUtils.projectPath
4 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:

```

1 def testDir = new File(projectPath + testFilesFolder)
2 if( !testDir.exists() ) {
3     testDir.mkdirs()
4 }

```

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.

```

1 def pdpFile = new File(projectPath + testFilesFolder +
2     "kismoAutoPdp_generated.csv")
3 if (pdpFile.exists()) {
4     pdpFile.delete();
5 }

```

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)

```

1 def pdpHeader =
2     "surname;name;patronymic;birthday;docType;docNumber;documentAdditionalInfo;departPlace;arrivePlace;routeType;departDate;departDateFact;citizenship;gender;recType;rank;operationType;operatorId;placeId;route;places;buyDate;termNumOrSurname;arriveDate;arriveDateFact;grz;model;registerTimeIS;operatorVersion"
3 def pdpLine1 = "КисмоАвтоА;Алефтин;Тестовый;1971-11-15;0;8602328040;;Кисмо
4     Автостанция ЗПФС 01;Кисмо Автостанция ЗПФС
5     02;0;${ttDatePdp}T12:00Z;;РОССИЯ;М;1;;1;20998;22998;Кисмо Авторасписание ЗПФР
6     01;4;${ttDatePdpBuy}T01:00Z;345;${ttDatePdp}T14:00Z;${ttDatePdp}T14:00Z;;${ttDatePdpRegister}T13:00Z;1"
7 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:

```

1 import com.eviware.soapui.support.GroovyUtils
2 import java.io.File
3 import java.util.concurrent.ThreadLocalRandom
4 import java.time.LocalDate
5 import java.time.format.DateTimeFormatter
6 import java.io.BufferedWriter
7

```



```

8  date = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-
   dd")).toString()
9
10 generatedFile =
   """"surname;name;patronymic;birthday;docType;docNumber;documentAdditionalInfo;de
   partPlace;arrivePlace;routeType;departDate;departDateFact;citizenship;gender;re
   cType;rank;operationType;operatorId;placeId;route;places;buyDate;termNumOrSurna
   me;arriveDate;arriveDateFact;grz;model;registerTimeIS;operatorVersion
11  Воробьёв;Андрей;Сергеевич;1971-11-
   15;0;8602328040;;Горицы;кижи;0;${date}T23:30Z;;РОССИЯ;М;1;;1;20100;22000;M197;4
   ;${date}T01:00Z;345;${date}T10:30Z;${date}T10:30Z;;;${date}T13:00Z;1""""
12
13 def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
14 def projectPath = groovyUtils.projectPath
15
16 def writeToFile(String folder, String fileName, String inputString) {
17     OutputStream outputStream = new FileOutputStream(folder + "/" + fileName)
18     OutputStreamWriter outputStreamWriter = new
   OutputStreamWriter(outputStream, "UTF-8")
19     outputStreamWriter.write(inputString)
20     outputStreamWriter.flush()
21     outputStreamWriter.close()
22 }
23
24 writeToFile(projectPath,"upload.csv",generatedFile)
25
26 log.info(" Создан файл: " + projectPath + "\\\" + "upload.csv")

```

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

```

1  import com.eviware.soapui.support.GroovyUtils
2  import java.io.File
3
4  def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
5  String encodedFileFromFS = (new File(groovyUtils.projectPath + "/" +
   "upload.xml" )
6
   .getText("UTF-8"))
7
   .getBytes( 'UTF-8' ).encodeBase64()
8
9  testRunner.testCase.setPropertyValue( "encodedFileFromFS" , encodedFileFromFS)

```

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:

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

Для деактивации:

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

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

Например вот так:

```
1  /*Получаем список всех шагов теста */
2  def oNameList = testRunner.testCase.getTestStepList().name
3
4  def createOperator= testRunner.testCase.getPropertyValue( "createOperator" )
5
6  if (createOperator == "true") {
7      log.info(' Нету оператора с таким egis_id, создаем')
8      testRunner.testCase.getTestStepByName("Create Operator").setDisabled(false)
9      testRunner.testCase.getTestStepByName("Check Operator").setDisabled(false)
10     testRunner.testCase.getTestStepByName("Remove Operator
11     DB").setDisabled(false)
12 }
13 else if (createOperator == "false") {
14     log.info(' оператор с таким egis_id уже есть, используем его, деактивируем
15     ненужные шаги')
16     testRunner.testCase.getTestStepByName("Create Operator").setDisabled(true)
17     testRunner.testCase.getTestStepByName("Check Operator").setDisabled(true)
18     testRunner.testCase.getTestStepByName("Remove Operator
19     DB").setDisabled(true)
20 }
21 else {
22     log.info(' Ошибка чтения свойства или я уже не знаю, что там у вас
23     случилось')
24 }
25 }
```

Операции с БД в скрипте Groovy

При подключении 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):

```

1 def sql =
  sql.newInstance("jdbc:postgresql://192.168.70.91:5432/ORA2PG_T", "postgres", "post
gres", "org.postgresql.Driver")
2 def stations = sql.rows '''
3 SELECT "ASID" FROM "AUTO_ONSI"."AUTO_STATION" WHERE "FOREIGN_ID" LIKE
  'testKismoPdpAs%' ORDER BY "FOREIGN_ID" ASC;
4 '''
5 def stationId1 = stations[0][0]
6 def stationId2 = stations[1][0]
7 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:

```

1 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:

```

1 def timetable = sql.rows '''
2 SELECT "TTID" FROM "AUTO_TIMETABLE_NEW"."TIMETABLE" WHERE "NAME" = 'Кисмо
  Авторасписание ЗПФР 01';
3 '''
4 for(i=0; i<timetable.size; i++) {
5   def ttid = timetable[i][0]
6   sql.execute "DELETE FROM \"AUTO_TIMETABLE_NEW\".\"AUTO_ROUTE_POINT\" WHERE
  \"TTID\" = '${ttid}'; DELETE FROM
  \"AUTO_TIMETABLE_NEW\".\"ACTUAL_TIMETABLE_SHORT\" WHERE \"TTID\" = '${ttid}';
  DELETE FROM \"AUTO_TIMETABLE_NEW\".\"ATT_GEN_INFO\" WHERE \"TTID\" = '${ttid}';
  DELETE FROM \"AUTO_TIMETABLE_NEW\".\"TIMETABLE\" WHERE \"TTID\" = '${ttid}';"
7 }

```

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:

```

1 import groovy.sql.Sql
2 import java.sql.Driver
3

```

```

4  def (conString, conUser, conPass, conDriver) =
    testRunner.testCase.testSuite.getPropertyValue( "dbConnectionVZO" ).split(",")
5  def sql = Sql.newInstance(conString,conUser,conPass,conDriver)
6
7  def userId = testRunner.testCase.getPropertyValue( "userId" )
8  def groupId = testRunner.testCase.getPropertyValue( "groupId" )
9
10 sql.withStatement {
11     stmt -> stmt.queryTimeout = 30
12 }
13 log.info ("Connection established")
14
15 def clearIs(String uid, String gid, Sql sql) {
16
17     sql.execute """
18         DELETE FROM GENERAL_ONSI.INFORMATION_SOURCE WHERE ID = ${uid}
19     """
20     sql.execute """
21         DELETE FROM GENERAL_ONSI.INFORMATION_SOURCE_GROUP WHERE GID = ${gid}
22     """
23 }
24
25 clearIs(userId, groupId, sql)
26 sql.close()

```

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. 2...n Все наши шаги теста, которые надо повторять
3. 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:

```

1  IP:192.168.70.95:9080,192.168.70.95:9081,192.168.70.95:9082
2  counter:0

```

В первом скрипте запишем:

```

1  /*Получаем список IP-шников из переменной*/
2  def s = testRunner.testCase.getPropertyValue( "IP")
3  /*конвертируем строку с ip-шникам в список*/
4  def list = s.split(',')

```

```

5  /*Проверяем, какое значение счетчика в переменной counter*/
6  def i = Integer.parseInt(testRunner.testCase.getPropertyValue( "counter"))
7  /*Меняем все Endpoint (адреса сервиса) IP на IP из списка. */
8  def teststeps = testRunner.testCase.getTestStepList()
9      teststeps.each { teststep ->
10          teststep.setPropertyValue('endpoint','http://' + list[i] +
11              '/dataReader/services/DataReaderService?wsdl')
12      }
13  /* Прибавляем к счетчику (counter) 1 */
14  i++
15  testRunner.testCase.setPropertyValue( "counter", i.toString())

```

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.

```

1  def s = testRunner.testCase.getPropertyValue( "IP")
2  def list = s.split(',')
3
4  def i = Integer.parseInt(testRunner.testCase.getPropertyValue( "counter"))
5  log.info(list.size())
6
7  if (i < list.size()) {
8      testRunner.gotoStepByName ('Setup Script [Testovaya]')
9  } else {
10     testRunner.testCase.setPropertyValue( "counter", new Integer(0).toString())
11 }

```

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`:

```

1 import com.eviware.soapui.support.UISupport
2
3 UISupport.showInfoMessage("Это маленькой информационное сообщение в двойных
  кавычках!")
4
5 UISupport.showInfoMessage("""А это большое сообщение
6 из нескольких строк!
7
8 внутри можно использовать пробелы, пустые строки и переменные ${var}!
9 """)

```

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.

```

1 import com.eviware.soapui.support.XmlHolder
2 import com.eviware.soapui.support.GroovyUtils
3 import com.eviware.soapui.support.UISupport
4
5 def SortString(String input) {
6     char[] charArray = input.replaceAll("[^a-zA-Z0-9 ]+", "").toCharArray()
7     Arrays.sort(charArray)
8     String sortedString = new String(charArray)
9
10    return sortedString.trim()
11 }
12
13 responseXmlHolder = new XmlHolder(messageExchange.getResponseContentAsXml())
14 dataXml = responseXmlHolder.getNodeValue("//*:transferData")
15
16 if (!dataXml.contains("не найден")) {
17     String responseDecoded = new String(dataXml.decodeBase64(), "UTF-8")
18     def userFile =
19     messageExchange.modelItem.testStep.testCase.getPropertyValue( "userFile" )
20     String userFileDecoded = new String(userFile.decodeBase64(), "UTF-8")
21
22     assert SortString(responseDecoded) == SortString(userFileDecoded),
23     "Отправленные и прочитанные данные не совпадают"
24     UISupport.showInfoMessage("""Отправленные:
25 ${userFileDecoded}
26
27 Полученные данные:
28 ${responseDecoded}
29 """)
30 }
31 else {
32     assert !dataXml.contains("не найден"), "Пользователь не найден"
33     UISupport.showErrorMessage("Пользователь не найден")
34 }

```

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

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

Пример выбора среды:

```
1 | // Выбор среды запуска для тест-кейса (можно также написать для тест-суита,  
   | проект)  
2 | def result = com.eviware.soapui.support.UISupport.prompt("Выберете среду  
   | запуска", "Среда запуска", ['Тестовая', 'СГК', 'САТ', 'Рабочая'])  
3 | switch(result) {  
4 |     case "Тестовая":  
5 |         testRunner.testCase.setPropertyValue( "ServiceEndpoint"  
   | , "192.168.70.92:9080")  
6 |         break  
7 |     case "СГК":  
8 |         testRunner.testCase.setPropertyValue( "ServiceEndpoint"  
   | , "10.10.34.31:9080")  
9 |         break  
10 |    case "САТ":  
11 |        testRunner.testCase.setPropertyValue( "ServiceEndpoint" , "какое-то  
   | значение")  
12 |        break  
13 |    case "Рабочая":  
14 |        testRunner.testCase.setPropertyValue( "ServiceEndpoint" , "какое-то  
   | рабочее значение")  
15 |        break  
16 |    default:  
17 |        log.info("Неизвестное значение")  
18 |        break  
19 | }
```