Wiki »

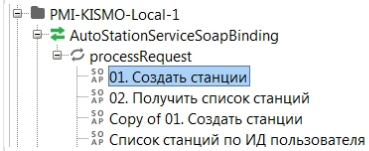
Тестирование веб-сервисов How-To Guide от простого к сложному

Содержание

Статья по **автоматизации тестирования**, тест-кейсы и тест-суиты тут
SoapUi (Open Source) можно скачать тут: https://www.soapui.org/downloads/latest-release.html (выбрать 64bit или 32bit)

Термины принятые тут

- "интерфейс" - узел с зелеными стрелочками в SoapUI;
- "проект" - папка проекта, в котором много интерфейсов с зелеными стрелочками;
- "запрос" - один единственный запрос
- "метод" - указание в запросе веб-сервису, что делать с переданными данными. Например, метод "PUT" (1) - создать новую запись, метод "GET" (4) - получить данные из БД, метод "DELETE" - пометить данные, как удаленные, метод "UPDATE" - обновить существующую запись.



- PMI-Kismo-Local - проект;
- AutoStationServiceSoapBinging - интерфейс;
- 01 Создать станции - запрос

Список веб-сервисов, назначение, адреса

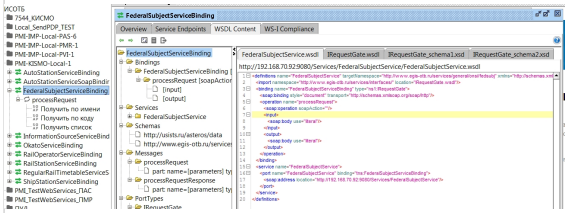
Из чего сделан адрес веб-сервиса обработчиков

Адрес веб-сервиса составлен из нескольких частей. Например для сервисов на WildFly:
http://192.168.65.60:8330/Services/AutoStationService?wsdl ,
где:

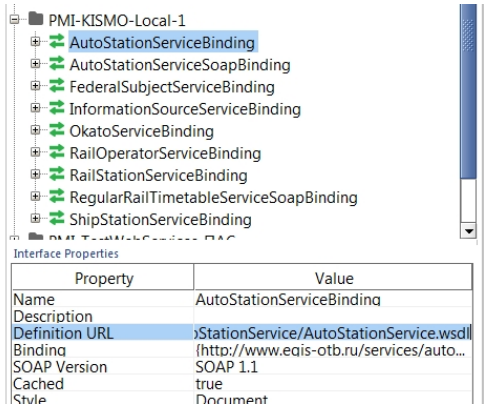
- http://192.168.65.60:8330 - адрес сервера, на котором развернуты обработчики (уточнять у разработчиков, по таблице IP-адресации, опытным путем :));
- /Services/ - общая часть для всех сервисов
- AutoStationService?wsdl - название файла с описанием сервиса.

В SoapUI адрес сервиса указывается в нескольких местах:

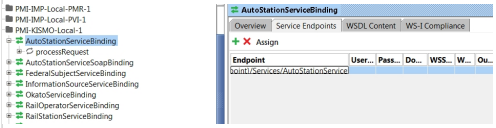
- в строке адреса запроса - это основной адрес. В большинстве случаев, достаточно поменять только этот адрес (например, просто поменять порт или IP). В этом поле адрес указывается без окончания ?wsdl (например: http://192.168.65.60:8330/Services/AutoStationService)



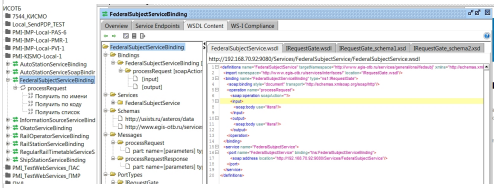
- в строке Definition URL свойств интерфейса (отображается в нижней части левой панели при выделенном интерфейсе). В этой строке указывается адрес с окончанием ?wsdl (или .wsdl для WAS)



- в окне редактирования интерфейса на вкладке "Service Endpoint". Эта строка обновляет все адреса запросов. Здесь указывается адрес без окончания ?wsdl:



Из WSDL загружается описание сервиса (структура запроса, вспомогательные схемы). SoapUI загружает описание один раз и сохраняет его в файле проекта. Это можно увидеть, нажав два раза по "интерфейсу" (узел с зелеными стрелочками) и переключившись на вкладку WSDL-контент.



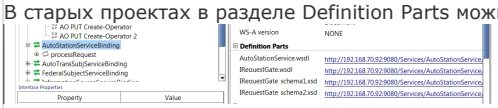
Для сервисов на WAS адрес сервиса может быть таким (такой формат можно видеть в старых проектах SoapUI:
http://192.168.70.92:9080/Services/AutoStationService/AutoStationService.wsdl
,где

- http://192.168.70.92:9080 - адрес обработчиков на WAS
- /Services/ - общая для всех сервисов часть;
- AutoStationService/ - название "папки" в которой лежит файл. Для каждого сервиса своя
- AutoStationService.wsdl - название файла с описанием сервиса.

Веб-сервисы также есть у ПУДа (для записи ПДП в обход обработчиков) и у шлюзов.

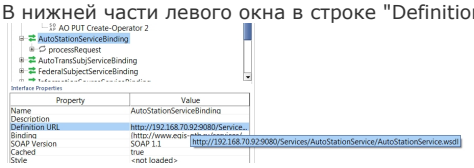
Как заменить адрес старого сервиса и обновить Definitions Part

В старых проектах в разделе Definition Parts можно видеть старые адреса.



Их можно обновить без пересоздания интерфейса. Для этого:

- Выбрать интерфейс, который нужно обновить.
- В нижней части левого окна в строке "Definition URL" посмотреть адрес. Для старых проектов это может быть так:



- Это адрес к старым сервисам. Нужно обновить его на новый адрес, например, вставим туда новый адрес тестовой зоны:
  http://192.168.65.60:8330/Services/AutoStationService?wsdl (в старых проектах может быть лишняя папка в пути и .wsdl вместо ?wsdl)
- После этого этого кликнуть правой кнопкой по интерфейсу и нажать Update Definitions (или кнопку F5).
- В появившемся окне нажать "Ок", ничего не менять (там будет стоять только одна первая галочка).
- Появится окошко с восклицательным знаком "Missing matching binding". Просто нажать Ок.
- Появится окошко с запросом на обновление интерфейса ("Update existing requests with new endpoints?"). Нажать "Ок".
- Интерфейс обновится, обновятся Definition Parts.

Список адресов Прод, СГК, Тестовая

Актуальная версия тут: https://rm.inforion.ru/projects/egis/wiki/Prodloc#ВЗО-Обработчики

Показать

Программы и утилиты

- SoapUI - основная утилита для отправки запросов к веб-сервисам
- Notepad++ - вспомогательная утилита, блокнот для редактирования
- Расширение для браузера: Wizdler (Chrome, Firefox). Позволяет строить примеры запросов из wsdl/xsd прямо в браузере.

В Notepad++ понадобятся следующие базовые команды:

- для того чтобы в XML-файле подсвечивался синтаксис: в меню Синтаксисы > XML;
- чтобы сериализовать (конвертировать) часть текста в base64 (это нужно для передачи сервису файлов ПДП) - нужно выбрать текст, кликнуть правой кнопкой мыши и выбрать "Plugin Commands" > "Base64 Encode". Например:

обычный текст: мама мыла раму
тот же текст в base64: 0LzQsNC80LAg0LzRi9C70LAg0YDQsNC80YM=

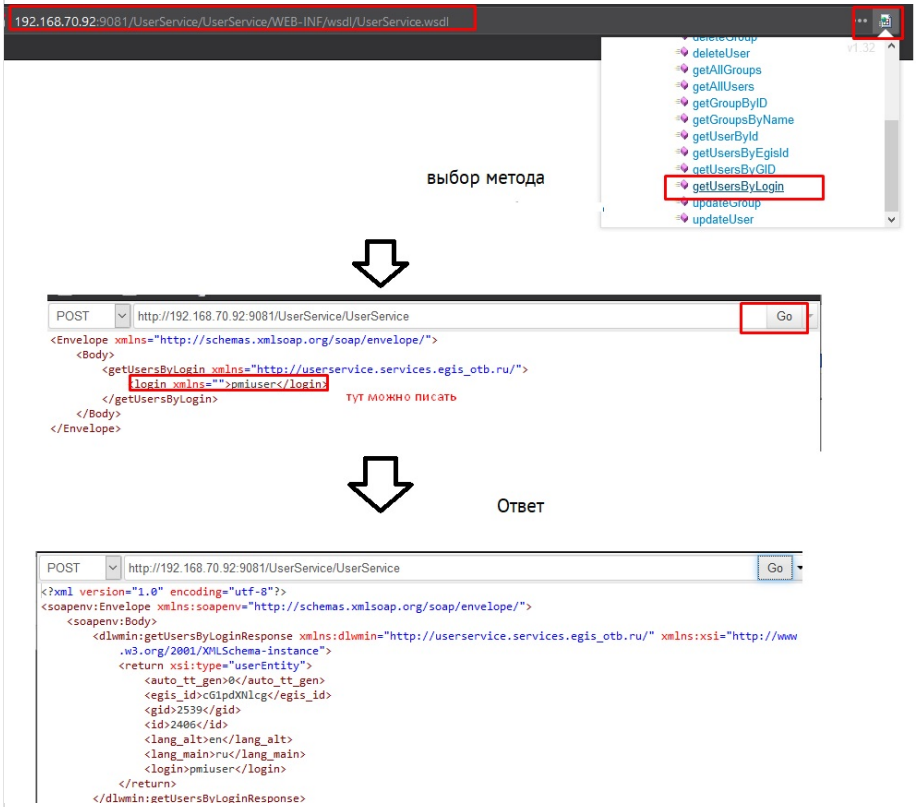
- чтобы "расшифровать" текст, нужно выделить его, кликнуть правой кнопкой мыши и выбрать "Plugin Commands" > "Base64 Decode"

Плагины к Notepad++ - для установки плагинов перейти в меню "Плагины" > "Plugin Manager" и установить:

XML Tools - плагин для работы с XML-файлами. Полезные команды:

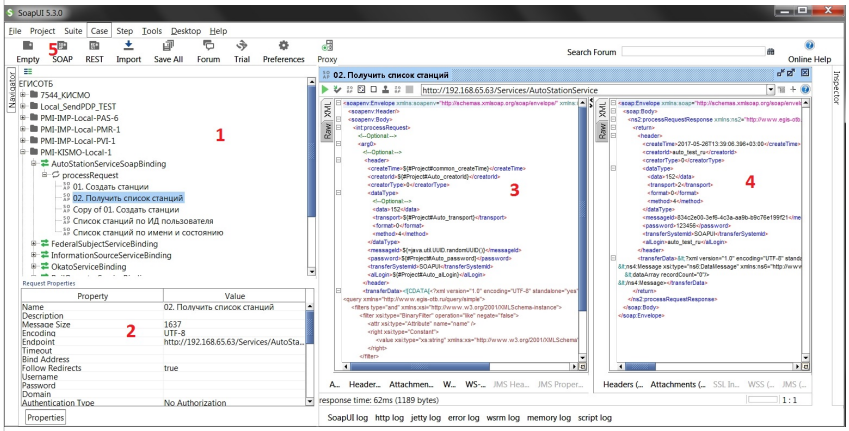
- Check XML Syntax - проверяет базовый синтаксис XML, может найти ошибки вроде незакрытого тэга или лишнего символа;
- Pretty Print (XML Only) - делает XML-ку красивой и читаемой расставляя в нужных местах отступы и табуляции;
- Convert selection XML to text - экранирует XML-символы (вместо символов < > ставит везде < и >). Применяется к выделенному тексту.
- Convert selection text to XML - операция, обратная предыдущей
- Comment selection / uncomment selection - закомментировать выделенный текст (например, если требуется временно "деактивировать" часть файла, чтобы она не обрабатывалась, но совсем удалять не хочется)
- Validate - проверяет структуру файла по указанному XSD.

Пример работы Wizdler'а на сервисе UserService.wsdl:



SoapUI: основные элементы интерфейса

1. Дерево проектов. Тут перечислены все открытые проекты.
2. Свойства выделенного объекта (проекта, интерфейса, запроса)
3. Основное поле запроса. Открывает двойным нажатие по запросу в панели слева. Это то, что мы отправляем сервису
4. Здесь отображается ответ, который приходит от сервиса
5. Основные панели меню



Также сразу рекомендую **настроить функцию автосохранения** (по умолчанию отключена). Для этого перейти в File > Preferences > выбрать "UI Settings" > в поле "AutoSave Interval:" поставить нужное значение в мин. Если открыто много проектов, то автосохранение каждые n минут может подтормаживать (10-15 проектов переваривает незаметно для работы).

SoapUI: как создать, сохранить и открыть проект

Чтобы **Создать новый проект**, нужно просто нажать кнопку **"Empty"** в панели меню или выбрать в меню **"File"** пункт **"Create Empty Project"**. Пустой проект появится в списке слева.

Чтобы **переименовать** проект, нужно кликнуть по нему правой кнопкой мыши и выбрать в меню **"Rename"** (или нажать кнопку f2 на клавиатуре).

Чтобы **добавить в проект** новый интерфейс нужно:
1. нажать правой кнопкой мыши по проекту и выбрать пункт **"Add WSDL"**;
2. в открывшемся окне ввести адрес веб-сервиса (путь к файлу wsdl), например:
   http://192.168.65.60:8330/Services/AutoStationService?wsdl и нажать OK;
3. новый интерфейс появится в проекте. Если раскрыть узел созданного интерфейса, будет видно узел processRequest (для обработчиков), внутри которого создан шаблон запроса. Его можно открыть двойным кликом - справа откроется окно (слева - запрос, справа - ответ

Чтобы **сохранить проект**, нужно нажать по нему правой кнопкой мыши и выбрать пункт **"Save Project"**. Указать путь, куда сохранить проект. Проект сохраняется в формате XML. Все проекты можно сохранить, выбрав в меню **File** пункт **"Save All Projects"** (или нажать на клавиатуре Ctrl+Alt+S.

Чтобы **открыть проект** их XML файла, нужно нажать кнопку **Import** в панели меню и открыть файл проекта на локальном диске. Проект появится в списке проектов слева (его можно переименовать и пересохранить - Save Project As).

Если нужно **скопировать интерфейс** из одного проекта в другой, то:
нажать правой кнопкой мыши по интерфейсу, который нужно скопировать;
1. выбрать пункт **Clone Interface**;
2. в появившемся окне выбрать в выпадающем списке проект, куда нужно скопировать интерфейс и нажать OK;
3. скопируется интерфейс и все его запросы.

Если в ходе отмены изменений (так много раз нажал **Ctrl + Z**, что ОМГ **все пропало**), то нужно всего лишь нажать **Ctrl + Y** и оно вернется (сочетание клавиш для Redo).

Формат запроса

Запрос состоит из заголовка и основной части. Рассмотрим на примере. Это запрос (его можно целиком вставить в SoapUI) создает одну станцию

Показать

Заголовок запроса

Заголовок запроса размещается между тэгами <header> </header>, данные, которые необходимо передать обработчику - в поле </transferData>.
Рассмотрим заголовок:

Показать

Способ кодирования информации в полях creatorType, transport, format, method описывается классами пакета ru.usists.dispatcher артефакта xsd-classes (ссылка). Открываете класс и смотрите перечень значений.

Поле	Описание	Возможные значения	Пример
createTime	Дата запроса	Указывается дата запроса в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС.МММ+03:00 При передаче запроса генерировать автоматически, `${=new java.text.SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.ms+03:00").format(new Date())}`	
creatorId	Логин пользователя	Этот логин можно посмотреть по таблице GENERAL_ONSI.INFORMATION_SOURCES	auto_test_ru
creatorType	-	1 = User, Пользователь 2 - INTERNAL_SYSTEM, Внутренняя система 3 - EXTERNAL_SYSTEM, Внешняя система любое другое число - не обрабатывается	0
data	Тип передаваемых данных	1 – ПДП 2 – Расписания 152 – Станции 151 – Перевозчики 106 - Автовокзал 9 – Фактические расписания 3 – Ответ подсистемы 5 – Закрытый сервер квитанций 6 – Открытый сервер квитанций 8 – Проверка полноты данных 100 – ОКАТО 113 - Авиационный транспорт 117 - ТАП 107 - ТКП Типы файлов, передаваемых в архив подлинников: 114 – Информационное сообщение 108 - Квитанция с выходного шлюза 115 - Квитанция с входного шлюза 116 - Запрос на повторное размещение данных 109 - Терминальные таблицы 110 - Список стран 118 - Типы городов 119 - Неправильный файл с ШИ 120 - Консолидированный файл с выходного шлюза	152

		200 - Тип документа 201 - Тип маршрута 202 - Статус резервирования 203 - Страна 204 - Субъект федерации 300 - Отчет системы 305 - Список оригиналов файлов	
transport	Сегмент транспорта	0 - Смешанный 1 - АВИА 2 - АВТО 3 - ЖД 4 - МОРЕ	2
format	Формат данных	0 INTERNAL_XML – внутренний XML 1 CSV 2 EDIFACT 3 EXPRESS3 4 XML 256 ZIP_CSV 258 ZIP_EXPRESS3 259 ZIP_XML 260 INTERNAL_XML_ZIP	0 для soapui
method	Метод, операция в передаваемом запросе (создать, удалить, обновить)	1 - Создать 2 - Обновить 3 - Удалить 4 - Найти (Получить)	1
messageId	Идентификатор этого сообщения	Любое рандомное значение (лат буквы, цифры), может быть одинаковое для всех запросов. Но лучше все таки либо давать вменяемые идентификаторы, чтобы проще было искать в логах	ThisIsMessageId
password	Пароль пользователя, указанного в creatorId/ Пароль можно посмотреть в базе ПУДа	-	123456
transferSystemId	Идентификатор системы, которая передает сообщение	Может быть любое, я обычно пишу SOAPUI	SOAPUI
alLogin	Альтернативный логин, обычно - не используется	Совпадает с creatorId	auto_test_ru

Тело запроса. Пример создания станции

Тело запроса - собственно, данные, которые нужно передать обработчику.
Данные помещаются в поле <transferData> и в дополнительную обертку <![CDATA[... тут данные ....]]>.
_PSA: Ирина заметила, что формат данных похож на файл .proc с САТа, так что можно попробовать прямо его и запихать. _

Рассмотрим запрос с созданием одной станции:

Создание станции

В первой части запроса приведены **переменные пространств имен**. Нам здесь важно обратить внимания на переменные:

- **ns7**="http://www.egis-otb.ru/gtimport/" - эта переменная будет во всех запросах на создание (станций, операторов, раписаний;
- **ns4**="http://www.egis-otb.ru/data/onsi/stations/" - это только для станций
- **ns6**="http://www.egis-otb.ru/datatypes/" - для всех запросов.

Переменные указанные в первой части, должны совпадать по всему запросу. Из примера выше видно:

- указан тип данных ns7:ImportedEntry;
- указан тип записи (это автостанции) - ns4:AutoStation"

```
<data sourceId="testKismoSAS01" xsi:type="ns7:ImportedEntry">
            <data xsi:type="ns4:AutoStation"
```

Если какие-то пространства будут не совпадать, то запрос вернется с ошибкой типа "_Ошибка при обработке запроса: javax.xml.bind.UnmarshalException: Unable to create an instance of ru.egis_otb.datatypes.DataRecord_"

Далее идет блок с **количеством записей**, которые передаются в запросе: <dataArray **recordCount**="1">. Тут просто, сколько здесь указано записей, столько и будет обработано. То есть, если вы передаете 5 станций, а в recordCount указано "2", то обработано и записано в базу будет только первые две станции.

После этого идет обычный блок <data> с **данными станции**. Отличия от шлюзов и тестов на САТ:

- это необязательно, но все же лучше указывать в **sourceId** что-нибудь читаемое, уникально для теста, например - testKismo456,testKismo457. В этом случае вам просто будет найти ваши данные в базе, просто из удалить и при переносе тестов на СГК не возникнет ошибки с дублированием записей (ну потому что sourceId="001" уже везде есть;
- **isuid** и **isgid** - это идентификатор и идентификатор группы пользователя (того, что указан в поле creatorId). Как их узнать написано в разделе ниже тут.
- **countryCode, federalSubject, okato** - здесь должно быть указано как значение (значение при этом может быть любым не пустым, но лучше нормальные все таки писать), так и идентификатор.
  Например, чтобы записать страну:

```
<countryCode value="Российская Федерация" id="185" />
```

Окато:

```
<okato value="86000000000" id="1300033"/>
```

Идентификатор ОКАТО можно посмотреть по базе данных в таблице GENERAL_ONSI.CLASS_OKATO (поле CODE - 11 цифр кода окато, ID - идентификатор, который должен быть указан в запросе).
Также идентификатор можно запросить у сервиса OkatoService по имени или коду. Пример такого запроса указан ниже (поиск по коду 01204859002):

Показать

Как узнать creatorId и пароль по egisId или логину

Если прислали письмо, что проблема с какими-то данными (не находят расписания) на портале для пользователя с egisId=20000.
Сначала ищем логин этого пользователя в базе обработчиков. На СГК база данных - PostgreSQL, на рабочей - Oracle

```
Select * from "GENERAL_ONSI"."INFORMATION_SOURCE" where "EGIS_ID"='20000';
```

В поле Login логин этого пользователя.

По логину:

```
Select * from "GENERAL_ONSI"."INFORMATION_SOURCE" where "LOGIN"='auto_test_ru';
```

Также посмотреть:

- поле GID (группа источников информации);
- поле ID (идентификатор).

Эти данные нужны для работы с запросами.

Далее делаем запрос к базе данных ПУДа. Oracle на СГК и на рабочей (ВЗХ база).

```
select * from ACS1_1.USER_ACCOUNT where LOGIN='AbramovJuL';
```

AbramovJuL – это логин, который получили ранее. В поле Password пароль этого пользователя. С этими данными можем зайти на портал ПЗП-А и посмотреть информацию. Также эти данные можно использовать для работы запросами.

Формат запросов поиска (GET 4)

Помимо создания (метод PUT - 1) сервисы также поддерживают метод GET (получить), с помощью которого можно искать записанные данные.
Рассмотрим пример с поиском станции по имени.

Показать

Как видно, заголовок почти такой же как для метода PUT (1) - см. описание заголовка в разделе выше . Отличие: в поле <method> указано не 1, а 4. В поле transferData передается запрос на поиск (или "фильтр"). Все возможные значения в этом поле описаны в XSD-схеме

Этот запрос делает две вещи: ищет все станции, начинающиеся с КИСМО и выводит только 6 результатов (maxResults). Рассмотри фильтр по имени:

```
<filters type="and" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <filter xsi:type="BinaryFilter" operation="like" negate="false">
            <attr xsi:type="Attribute" name="name" />
            <right xsi:type="Constant">
                <value xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">КИСМО%</value>
            </right>
        </filter>
</filters>
```

- тэг <filters> </filters> - поле для перечисления фильтров. Внутри может быть перечислено несколько фильтров. **Логическая операция** над фильтрами указывается в атрибуте **type=**. В данном случае это <filters type="and" - то есть все перечисленные фильтры в поле <filters> (у нас он один) должны быть выполнены. Возможные значения: and (И), or (ИЛИ);
- <filter xsi:type="BinaryFilter" operation="like" - здесь указан тип фильтра (бинарный BinaryFilter). Все типы фильтра указаны в файле, для поиска чаще всего используется бинарный. В некоторых сервисах удобнее использовать QueryFilter. В operation="like" указана операция, которая должна быть выполнена. Для бинарного типа возможные значения: eq ("="), ne (не равно), like (похож), lt (меньше), gt (больше), le (меньше или равно), ge (больше или равно). В данном случае указана операция like, чтобы найти все станции, которые начинаются на КИСМО. Если нужно найти по точному значению - то тут указывается оператор "eq" (equal - равно)
- <attr xsi:type="Attribute" name="name" /> - здесь в атрибуте name= указывается собственно поле, по которому нужно выполнить поиск. В данном случае это имя станции (name). Может быть указано любое другое поле, например sourceId, или nearestTown и т.д.
- <value xsi:type="xs:string" - здесь указан тип данных. Для имени (name) - это string (строковое значение). Для каких-то других полей это может быть другой тип, например xs:short, xs:long,xs:integer. Я в основном выясняю опытным путем, но вообще все типы полей описаны в соответствующих XSD схемах (например, для поля статус (state) тип данных xs:short);
- КИСМО%</value> - собственно само значение, по которому выполнить поиск. В данном случае - начальные буквы станции с %, который заменяет все остальные возможные символы.

<maxResults>6</maxResults> - это простой запрос, который выводит только 6 результатов. Также можно еще использовать например order.

Ниже приведен пример запроса расписания, который выводит 1 расписание с указанным именем и в состоянии active.

Поиск актуального расписания по имени

Примеры запросов

Отправка ПДП в сервис обработчика

ПДП файл можно отправить сервису ParserManagerBinding (http://IP:PORT/ParserManagerRouter/ParserManager?wsdl). ПДП файл передается сервису в виде сериализованой в base64 строки и дополнительных данных (имя файла, тип данных и.т.д.).

Рассмотрим пример. В примере ниже отправляется файл из 4 ПДП записей: Отправка ПДП

Как видно, в заголовке запроса (<header></header>) указан тип данных ПДП (<data>1</data>), остальное аналогично описанному формату в разделах выше.

В <transferData> передаются сами данные. Данные разделены на две основные части: fileData и fileInfo. В fileData записан сам файл в строке base64. Для того чтобы убедиться в этом, необходимо скопировать кракозябры между тегами fileData, вставить в Notepad++, выделить все и нажать по выделенному тексту правой кнопкой мыши. В контекстном меню выбрать Plugin Commands > Base64 Decode. Будет видно, что передается обычное содержимое обычного файла ПДП (csv формата, с заголовками). Соответственно также можно сериализовать любые данные обратно в base64 (Plugin Commands > Base64 Encode)

Показать

В части <fileinfo> указывается информация о файле:
TODO: Уточнить значения, сделать список всех форматов.

Параметр	Описание	Возможные значения	Пример
rewrite	?	true false	обычно true
ackSrc	?	1 2 3 4 5	5
size	Размер файла в байтах (?)	Любое целое	1723
format	Формат передаваемых данных	plain:auto-csv:1 plain:ship-csv:1 plain:rail-csv:1	plain:auto-csv:1
fileName	Имя файла	Может быть указано любое	20000_2017_05_02_04_00_00_019.csv
createdAt	Дата создания файла		2017-05-16T13:30:22.629+03:00
archiveId	Идентификатор записи в архиве	Должно быть уникальные для каждой передаваемой записи	

sendDataWriteRequest: Отправка ПДП в ПУД

Запись ПДП напрямую в БД (в обход обработчиков) в сервис AccessControl . В сервисе несколько методов, для отправки ПДП данных используется метод **sendDataWriteRequest** .

Пример запроса для отправки одной записи ПДП:

Запрос на запись ПДП

При записи не выполняется валидация данных, поэтому можно записывать всякую ерунду (например, очень много цифр в поле docNumber). При записи нужно менять arrayId и Id (или генерировать их автоматически), а также Имя (или другое поле, иначе будет ошибка об ограничении уникальности).

Расширенные свойства проектов, параметризация

Использование java классов в проекте: UUID

В SoapUi есть возможность вставлять в запросы код java в определенном формате. Для чего это нужно? Самый простой пример - автоматическая генерация messageId (поле messageId в заголовке файле).

Вставка кода или переменной вместо значения в запрос выполняется в следующем формате. Начало кода предваряет символ доллара ($), java код или переменная записываются в фигурные скобки {}.

```
${какая-то java фича}
```

Например, функция генерации случайного идентификатора (UUID):

```
=java.util.UUID.randomUUID()
```

Вставляем ее прямо в запрос в заголовок в поле <messageId>:

```
<messageId>${=java.util.UUID.randomUUID()}</messageId>
```

Теперь при каждой отправке сообщения в этом поле будет генерироваться рандомный идентификатор, и ничего не нужно менять руками.

Использование java классов в проекте: Текущая дата

Методы java Date, Calendar и SimpleDateFormat, конечно, были хороши (нет), но сильно устарели. Отныне и впредь правильно использовать для всего, связанного со временем, класс `java.time`. Примеры и описание на русском можно почитать тут . Старые примеры ниже под катом, они будут работать. Но поглядите, какие новые красивые:

В заголовке запроса есть поле <createTime>. В нем проставляется какая-то дата, ее можно проставлять и менять руками, а можно генерировать автоматически. Для этого вставляем в поле формулу:

```
${=java.time.LocalDateTime.now().toString()}
```

Результат: `<createTime>2020-03-23T12:26:44.2644+03:00</createTime>`

Дата сразу сгенерируется в нужном формате.

Если нужно отнимать и добавлять от текущей даты дни и часы, то теперь это проще простого:

```
# отнимаем часы
${=java.time.LocalDateTime.now().minusHours(3).toString()}
# добавляем дни
${=java.time.LocalDateTime.now().plusDays(1).toString()}
# отнимаем дни
${=java.time.LocalDateTime.now().minusDays(14).toString()}
```

Ну красота же? Форматирование даты под нужный нам вид, правда, все равно развесисто, но это не так часто требуется:

```
${=java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPattern("yyyy_MM_dd")).toString()}
```

А расчеты периодов теперь вообще просто сказка, но об этом в статье про автоматизацию. Целиком можно почитать тут

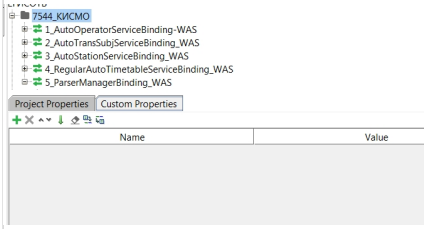
Старые дедовские методы

Параметризация данных

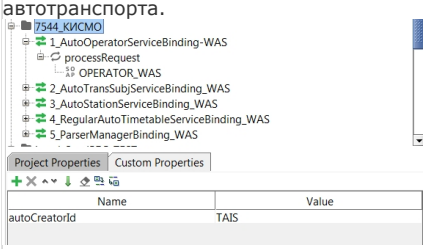
Предположим у нас есть проект, который нужно перенести на другую площадку. Или надо поменять пользователя во всех запросов. А еще бывает что в тесте нужно сквозное именование (например, один и тот же sourceId используется в нескольких запросах на обновление удаление и поиск). А еще когда переносишь проект на другую площадку, хочется просто поменять ip в одном месте, и чтобы везде сразу поменялось.

Для всего в SoapUi можно использовать свойства проекта. В свойствах проекта можно указывать в виде переменных какие-то значения, которые используются в запросах постоянно, а в самих запросах просто делать ссылку на эту переменную. Например, можно создать переменную, в которой указан логин пользователя, который используется для отправки запроса. В самих запросах указывать переменную. А если нужно перенести запрос на другую площадку, изменить пользователя - просто изменить значение переменной, и везде во всех запросах логин изменится автоматически.

Свойства проекте можно посмотреть в нижней левой панели при выделенном проекте на вкладке Custom Properties. Должен быть выделен именно сам проект (папка проекта), а не, например, интерфейс или запрос:



Сейчас в проекте нет никаких переменных. Чтобы создать переменную, нужно нажать пиктограмму с зелеными плюсиком. В поле Name указать какое-то название на латинице, в поле значение - указать значение. Например, создадим переменную для поля creatorId для автотранспорта.



Чтобы использовать переменную в запросе, нужно сослаться на нее в формате:

```
${#Project#ИМЯПЕРЕМЕННОЙ}
```

где ИМЯПЕРЕМЕННОЙ - наша переменная, на картинке выше - это autoCreatorId. А #Project - указание, что ищем переменную в текущем проекте, обычно не меняется (можно ссылаться и на другой проект).

Теперь в самом запросе в поле creatorId заголовка вставим нашу переменную:

```
<creatorId>${#Project#autoCreatorId}</creatorId>
```

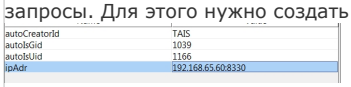
Теперь, если нужно поменять creatorId для всех запросов, достаточно просто поменять значение в Custom Properties проекта.
Но подождите, ведь с creatorId также связаны isUid и isGid (как узнать их описано тут), они тоже должны поменяться, если указан другой creatorId. Поэтому, нужно создать еще две переменные, в одной указать isUid, а в другой - isGid:

А в запросе (прямо в <transferData>) указать ссылку на эти переменные в соответствующих полях:

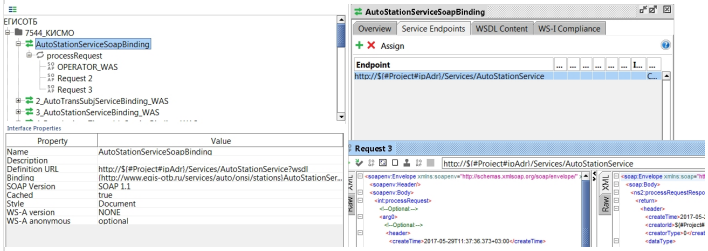
```
<data xsi:type="ns8:Operator"
        <тут какие-то данные
```

```
        name="АВТОКОЛОННА 0000"
        egisId="20100"
        isuid="${#Project#autoIsUid}"
        isgid="${#Project#autoIsGid}">
                <actualPeriod xsi:type="ns5:DateTimePeriod" to="2025-09-01T00:00Z" from="2015-09-01T00:00Z"/>
<countryCode value="Российская Федерация" />
```

Таким образом можно параметризовать любые значения. Например, я люблю записывать в параметры IP-адрес, куда направлять запросы. Для этого нужно создать переменную, например ipAdr=192.168.65.60:8330:



и указать эту переменную в поле Definition Url в свойствах интерфейса и в поле на вкладке Service Endpoints в Interface Viewer (двойной клик по интерфейсу). Для существующих запросов также можно поменять адрес запроса (как на картинке). Теперь, если мне нужно проверить запрос на другом IP-адресе, я просто меняют адрес и порт в переменной.



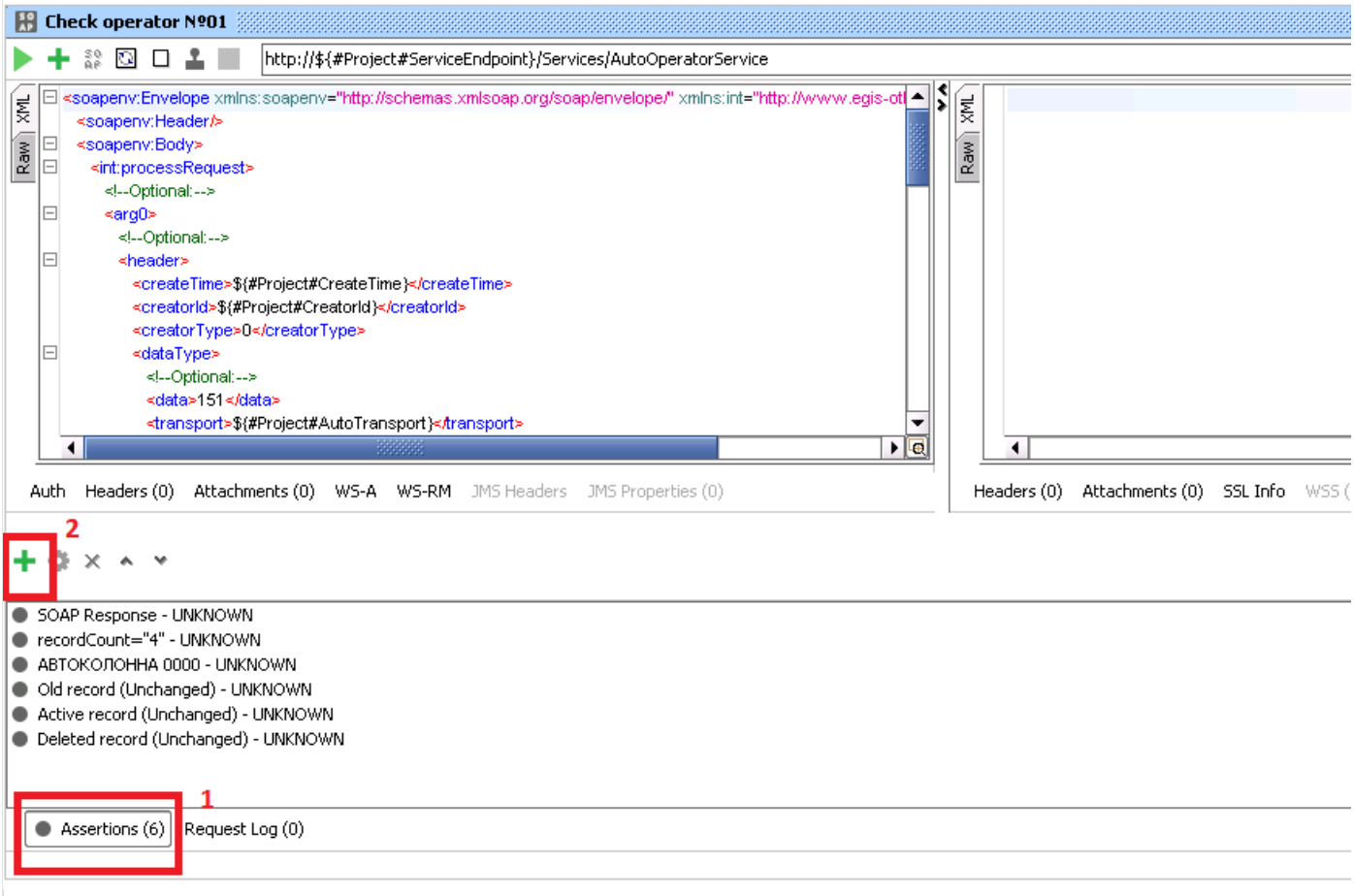
Все переменные из проекта можно сохранить (экспортировать) в файл. Для этого на вкладке Custom Properties проекта нужно нажать последнюю кнопку (там где зеленый плюсик). Все переменные будут сохранены в файл. Предпоследняя кнопка позволяет загрузить переменные из файла.

Использование регулярных выражений в проверках тест-кейсов

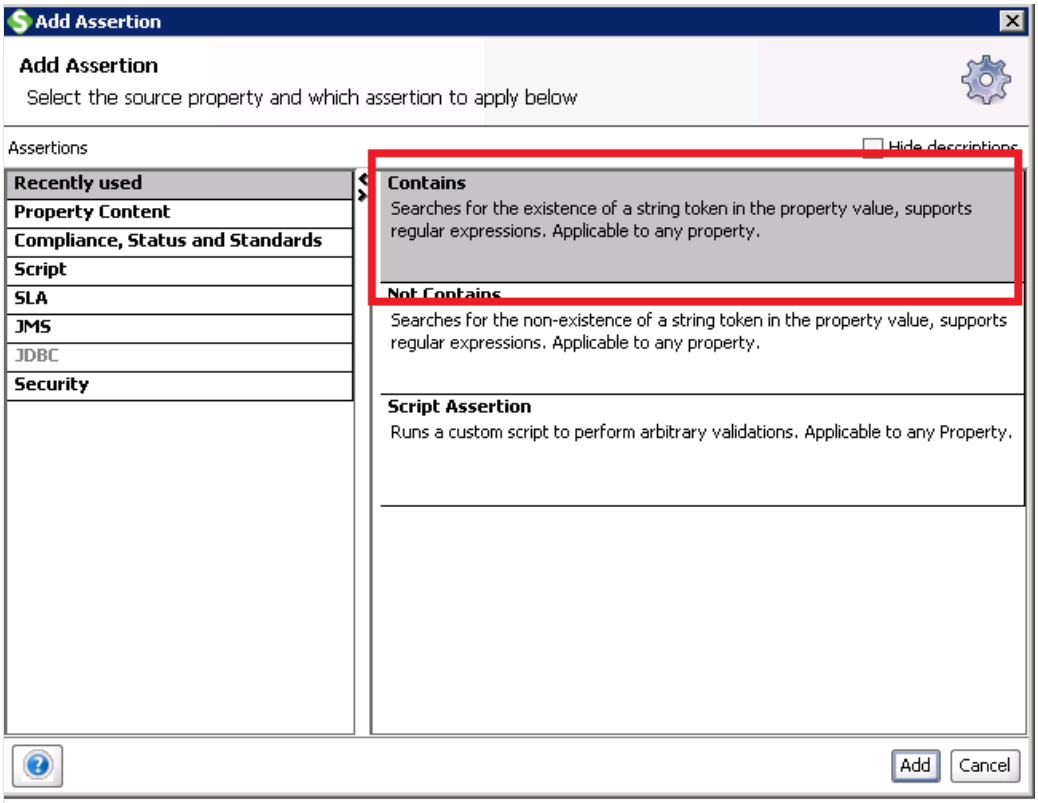
Сайты проверки и составления регулярок:

- https://www.regextester.com
- https://regex101.com
- https://regexr.com
- https://jex.im/regulex
- https://www.phpliveregex.com

В тест-кейсах можно делать проверки с помощью регулярных выражений. Для это надо зайти в запрос в тест-кейсе и зайти в Assertion (1)



Выбрать зеленый плюс (2). Затем выбрать "Contains".



В открытом окне нужно поставить галочку в чекбокс "Use taken as Regular Expression". Затем в поле "Content" уже написать регулярное выражение.
Несколько примеров регулярных выражений:

(?s).*sourceId="Soap7276_0001" isuid="[0-9]+" isgid="[0-9]+" state="active" id="[0-9]+"(?s).* - будет соответствовать sourceId="Soap7276_0001" isuid="1883" isgid="1281" state="active" id="31819"

[0-9]{4}[-]{1}[0-9]{2}[-]{1}[0-9]{2}[T]{1}[0-9]{2}[:]{1}[0-9]{2}[:]{1}[0-9]{2}[.]{1}[0-9]{3}[Z]{1} - будет пропускать любую дату 2025-09-01T00:00:00.000Z

(?s).*[E,e]rror(?s).*
(?s).*[E,e]xception(?s).*

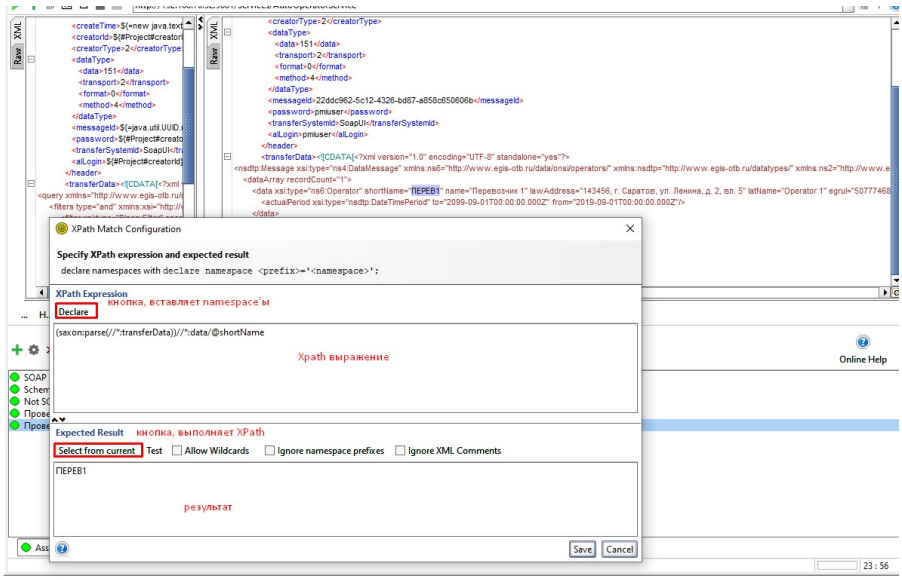
XPath для проверок и параметризации

XPath в проверках и просто в запросах/свойствах (Assertion > XPath Match) позволяет обращаться к значением полей и атрибутов в ответе запроса и сверять их с заданными.

XPath удобно использовать, если нужно, например, получить значение из ответа и использовать его в следующем запросе, проверить конкретное значение в ответе запроса. Для каких-то простых одиночных полей проще использовать XPath, однако нужно помнить, что свойства, полученные ответа - динамические, и работают только после выполнения запроса и получения ответа.

- Также для этого можно использовать скрипт Groovy (см. тут)
- Про пути XPath немножко можно прочитать тут
- Немножко примеров

В запросе в Assertions добавим XPath Match. В верхней части нужно ввести путь к значению, в нижней ожидаемый результат.
Сверху есть кнопка "Declare" (надпись), при нажатии которой в верхний запрос автоматически добавляются нужные неймспейсы. Я стараюсь составлять запрос так, чтобы они были не нужны.
В поле Expected Result тоже есть кнопка-надпись - Select from current, которая выполняет XPath выражение и отображает результат в нижнем поле. Для проверки в нижнем поле можно использовать регулярные выражения и wildcards (например, "*" - для всех символов, ? - для любого одного и т.д.



Для примера я выполнила запрос на чтение перевозчика. В ответе возвращен стандартный егисовский xml с transferData, внутри transferData - XML-ка с нужной информацией.

Получить любое значение поля ответа XML достаточно просто. Например, получим messageId из заголовка (text() - позволяет прочитать значение как строчку):

```
//*:header/messageId/text()
```

Или creatorId: `//*:header/creatorId/text()`

Чтобы читать значение внутри CDATA (в поле transferData) у SoapUI есть фича - saxon, - парсер XML, поторый можно использовать в выражении XML. В SoapUI используется версия 9.1 (вроде как).
Прочитаем значение shortName перевозчика в transferData:

```
(saxon:parse(//*:transferData))//*:data/@shortName
```

Все. Выражение в скобках сохраняет данные из transferData как XML, и дальше можно ходить по дереву XML как по папкам.

XPath также можно использовать для расширения свойств. Это достаточно удобная, но очень куцо документированная фича, которая позволяет обращаться к ответу или запросу прямо из ссылки на свойста. Например, в запросе мы можем обратиться к свойству тест-кейса из запроса вот так:
`${#TestCase#propertyName}`

Таким же образом можно обратится к шагу, а также, получить Response этого шага (Response - это такое же свойство). Вот так:
`${Название шага#Response}`

Название шага без кавычек, как есть. Далее мы можем расширить свойство и обратиться к значению поля XML-ответа:

```
${Прочитать#Response#//*:createTime/text()}
```

Получим текст из поля createTime ответа из шага теста "Прочитать". Добавим сюда saxon и прочитаем поле `name` из ответа:

```
${Прочитать#Response#//(saxon:parse(//*:transferData))//*:data/@name}
```

В месте, где вставлено свойство, будет вызвано значение поля ответа из указанного шага.
Это выражение также можно вставить в свойство тест-кейса, и обращаться к свойству - так тоже будет работать.

Автоматизируй это

Перенесла на отдельную страницу: SoapUI Автоматизация тестирования

Отлов запросов Wireshark'ом

Появился новый сервис, или старый запрос перестал работать, или непонятно в каком формате отправлять данные, и подсказать некому. Что делать? В этом случае может помочь Wireshark - утилита, которая записывает весь входящий и исходящий на целевую машину траффик, в том числе - все HTTP запросы к веб-сервисам. Прямо в Wireshark можно изучить запрос и посмотреть, в каком формате он был отправлен (можно прям его скопировать, и вставить в SoapUI с небольшими изменениями).

Инструкция по использованию Wireshark

Нюансы SoapUI (Шпаргалка/Правила хорошего тона)

1. При загрузки расписаний в базу данных в Soap запросе указываются id станций, операторов и вокзалов. Из-за этого может возникнуть ситуация, что расписание загружено, а станций (операторов или вокзалов) нет. Все это может вызвать ошибки при работе внешнего генератора. Поэтому если в базу было загружено такое расписание следует его удалить с помощью SQL запросов.

Решение некоторых ошибок

1. **Ожидаются элементы "{http://www.egis-otb.ru/query/simple}query&quot ...** (Возникла при создании расписания)
- Это означает что метод создания не поддерживается данным сервисом.

2. **Ошибка: prefix dt is not bound to a namespace** (Было: <countryCode value="Российская Федерация" xsi:type="dt:SimpleDictionaryValue" />)
- Нужно было <countryCode id="185" value="Российская Федерация" /> или <countryCode id="185"/>

3. **Ошибка: Fault occurred while processing** - Посмотреть блок выделенный: <![CDATA[< …. - После CDATA не должно быть пробелов, переносов и т.д.

4. **Источник информации не найден** - Проверить <creatorId>, <alLogin>, isuid и isgid, что существует такой перевозчик.

5. **Не удалось определить класс записи в коллекциях** - Возникает, если логин, указанный в заголовке soap запроса (тэг <creatorId>) не соотвествует isuid и isgid, которые указываются в запросе. ~~#8798~~

Обновлено Анна Леонова почти 2 года назад · 107 изменени(я, ий)

Go to top