GeekBrains

Тестирование Backend на Java

Повторение основ Web. Ручное тестирование SOAP API-сервисов с использованием SoapUI

На этом уроке

1. Узнаем, что такое API.
2. Рассмотрим типы API.
3. Научимся тестировать SOAP API.

Оглавление

Введение

В курсе «Тестирование веб-приложений» уже была тема, относящаяся к тестированию SOAP API с использованием SoapUI. Вернёмся к ней, чтобы автоматизировать ручные проверки. Для повторения базовой теории обратимся к методичке курса по тестированию веб-приложений. В этом пособии мы детальнее рассмотрим аспекты тестирования SOAP API и познакомимся с инструментами автотестера.

SOAP API

SOAP расшифровывается как Simple Object Access Protocol — простой протокол для доступа к объектам.

Основные особенности:

1. Прикладной протокол, часто использующий HTTP для передачи сообщений.
2. Все сообщения упаковываются в т. н. Envelope (конверт) и имеют строго определённую структуру.
3. Структура сообщений определена в XSD-файле и представляет собой XML.
4. Используется в больших enterprise-системах, где важна безопасность и целостность данных.
5. Если используется HTTP, то любой SOAP-запрос — это всегда POST-запрос.

Что нужно тестировать в SOAP API?

К функциональным проверкам относится следующее:

1. Соответствие WSDL- и XSD-требованиям.
2. Консистентность запросов и ответов, особенно при негативных проверках. Проще говоря, это проверки на пустые поля и некорректное заполнение.
3. Функциональная бизнес-логика приложения:
    - условно-обязательные поля, которые требуются только при определённых условиях;
    - интеграционные тесты из нескольких запросов;
    - прочее.

WSDL и XSD

Хорошая новость: даже если нет никакой документации, у SOAP API она есть всегда. Её имя — WSDL. В WSDL содержатся не только описания сервисов, но и XSD (XML Schema Definition) — файл описания типов данных для запросов и ответов сервисов. XSD также иногда включается в отдельный документ формата xsd.

WSDL — это xml-документ.

Это корневой тег WSDL:

```
wsdl:definitions
```

В нём объявляются пространства имён. В нашем примере это:

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://tempuri.org/">
```

Если XSD входит в WSDL, она содержит в тегах `<wsdl:types>`. В этих тегах может быть несколько XSD-схем. Сами схемы идут под тегами `<s:schema>`.

Для начала разберём вторую, меньшую, схему:

```
<s:schema elementFormDefault="qualified"
targetNamespace="http://microsoft.com/wsdl/types/">
<s:simpleType name="guid">
<s:restriction base="s:string">
<s:pattern
value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{1
2}"/>
</s:restriction>
</s:simpleType>
</s:schema>
```

Сам тег `<s:schema>` имеет два атрибута: `elementFormDefault` и `targetNamespace`. Они считаются необязательными и если указываются, то обычно вместе. elementFormDefault имеет два значения:

- `qualified` означает, что у всех элементов внутри есть namespace, он и указывается в `targetNamespace`;
- `unqualified` означает, что namespace у элементов схемы отсутствует.

Внутри схемы находятся объекты разных типов — простые (`s:simpleType`) и комплексные (`s:complexType`), которые состоят из одного или нескольких простых и/или комплексных объектов. Все пользовательские типы прописываются в тегах схемы, иначе WSDL не пройдёт валидацию. Простые типы имеют атрибут `name` (имя), по которому к ним обращаются другие объекты.

Самое замечательное для тестировщика находится внутри самого типа — это содержимое тегов restriction (ограничения). Они задаются разными способами, например, максимальным или минимальным значением, регулярным выражением, длиной выражения и так далее. Полное описание — здесь.

В нашем примере указывается шаблон для значения с использованием регулярного выражения для типа UUID. Это означает, что для полей с этим типом значения вроде «123» не пройдут валидацию, и запрос не отправится на сервер. У тега `restriction` также есть атрибут, указывающий, что это строка: `base="s:string"`.

Иногда сложный элемент состоит из одного простого. Такой элемент называется wrapper или обёрткой. Часто это требуется, чтобы дать имя простому элементу, например:

```
<s:complexType>
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="ProductID" type="s:int"/>
</s:sequence>
</s:complexType>
```

Здесь мы видим, что элемент `ProductID` — просто целое число, но ему дано пользовательское имя для улучшения читаемости кода. Тег `complexType` содержит указание, что это «последовательность» (sequence) элементов — самый распространённый вид дочернего тега, означающий обязательность присутствия элементов.

Другая возможность — тег `choice`. Он указывает, что его дочерние варианты не могут присутствовать в финальном документе одновременно.

Есть ещё два интересных атрибута, ограничивающие количество повторений элемента — `minOccurs="1" maxOccurs="1"` — не более и не менее одного, то есть ровно один раз.

Комплексные типы часто используют элементы других комплексных типов, например:

```
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="UpdateProductResult"
type="tns:ResponseResult"/>
</s:sequence>
</s:complexType>
```

Здесь тип `UpdateProductResult` содержит тип ResponseResult.

После описания всех схем тег `<wsdl:types>` закрывается, и начинается описание сервисов.

В тегах `wsdl:message` описываются форматы запроса и ответа для каждого из сервисов:

```
<wsdl:message name="GetProductSoapIn">
<wsdl:part name="parameters" element="tns:GetProduct"/>
</wsdl:message>
<wsdl:message name="GetProductSoapOut">
<wsdl:part name="parameters" element="tns:GetProductResponse"/>
</wsdl:message>
```

Все они используют типы, указанные в схеме выше.

В тегах `wsdl:portType` описываются:

- операции, которые выполняются с сервисами, именно их мы видим при импорте WSDL в новый проект SoapUI;
- и ожидаемые форматы запроса и ответа.

Например,

```
<wsdl:operation name="GetProduct">
<wsdl:input message="tns:GetProductSoapIn"/>
<wsdl:output message="tns:GetProductSoapOut"/>
</wsdl:operation>
```

В тегах `wsdl:binding` описывается способ доставки сообщений. Внутри снова появится теги `operation`, но теперь уже будут указаны эндпоинты для доставки сообщений и формат сообщения (например, document).

```
<wsdl:operation name="GetProduct">
<soap:operation soapAction="http://tempuri.org/GetProduct" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
```

И последнее, теги — это описание сервиса в целом, как правило, включая один или несколько элементов `<wsdl:port>` с информацией доступа для элементов `<wsdl:binding>`:

```
<wsdl:service name="SoapApi">
<wsdl:port name="SoapApiSoap" binding="tns:SoapApiSoap">
<soap:address location="http://soapapi.webservicespros.com/soapapi.asmx"/>
```

```
</wsdl:port>
</wsdl:service>
```

WSDL — это обычно первое, что создаётся аналитиками или разработчиками, а, значит, найдя ошибки и неточности там, можно сократить затраты на исправление.

Автоматизация с использованием SoapUI

Для повторения дальнейших шагов потребуется:

1. Создать новый проект в SoapUI.
2. Импортировать указанную WSDL.
3. Сформировать тест-сьют с одним тест-кейсом и со всеми возможными операциями. Об этом — ранее в курсе о тестировании веб-приложений.

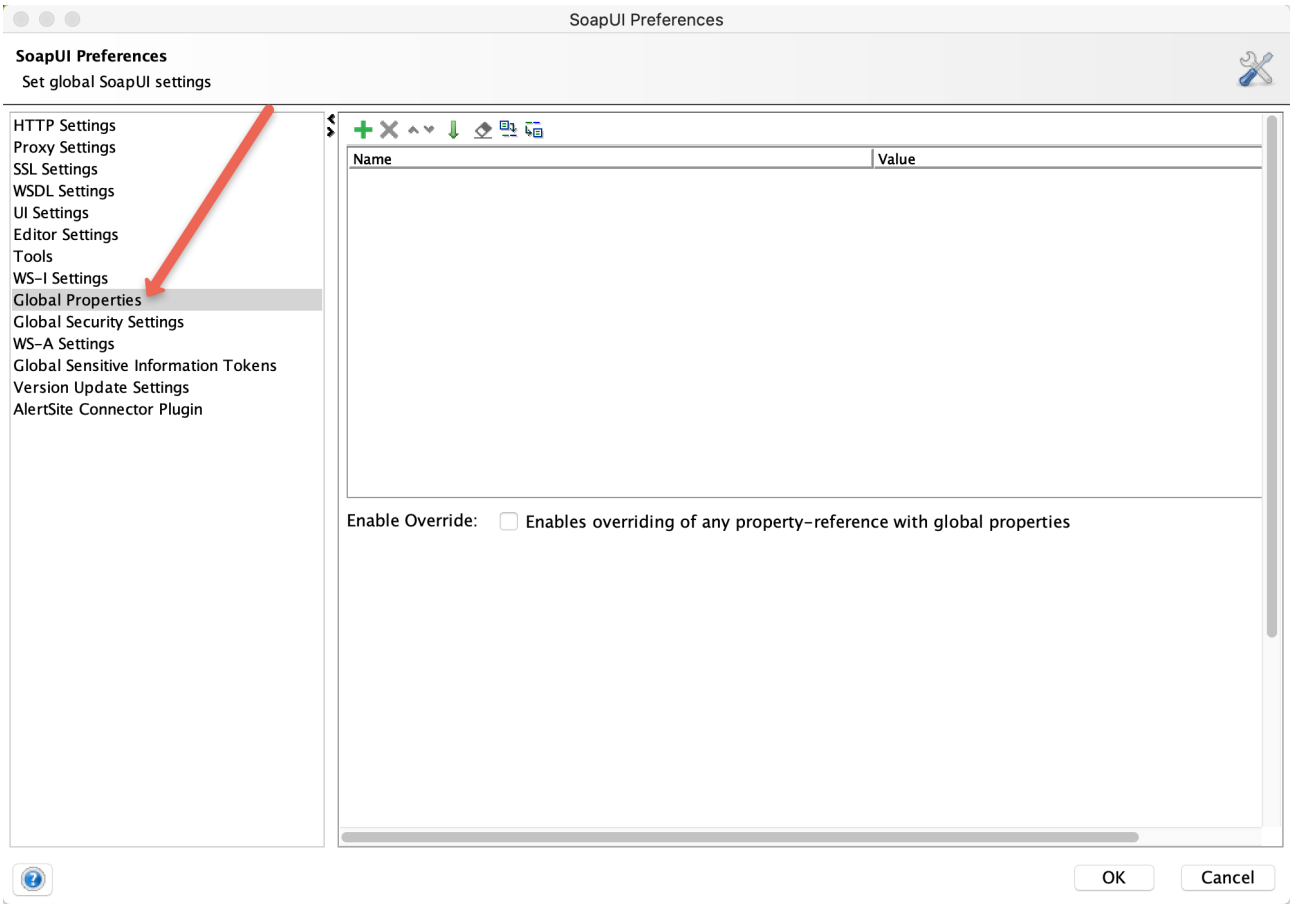
Первое, что смущает при просмотре запросы — хардкод данных, которыми нельзя управлять. Требуется вынести их в переменные и переиспользовать. Но как?

Переменные в SoapUI

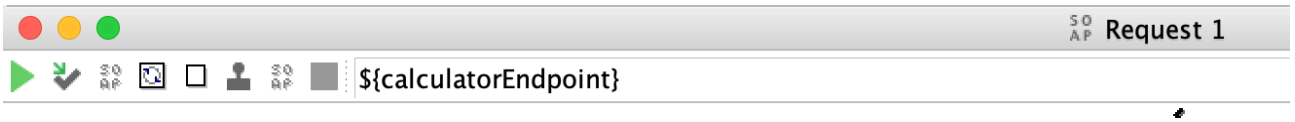
SoapUI часто использует пользовательские свойства для хранения пользовательских значений в проекте. Свойство (Property) — это строковое значение, так как в настоящее время все свойства обрабатываются как строки. Доступ к этому значению можно получить из скрипта, передачи свойств или ссылок на расширение свойств. SoapUI позволяет определять свойства на нескольких уровнях иерархии проекта.

Глобальные переменные

Устанавливаются в настройках SoapUI: File → Preferences → Global Properties.



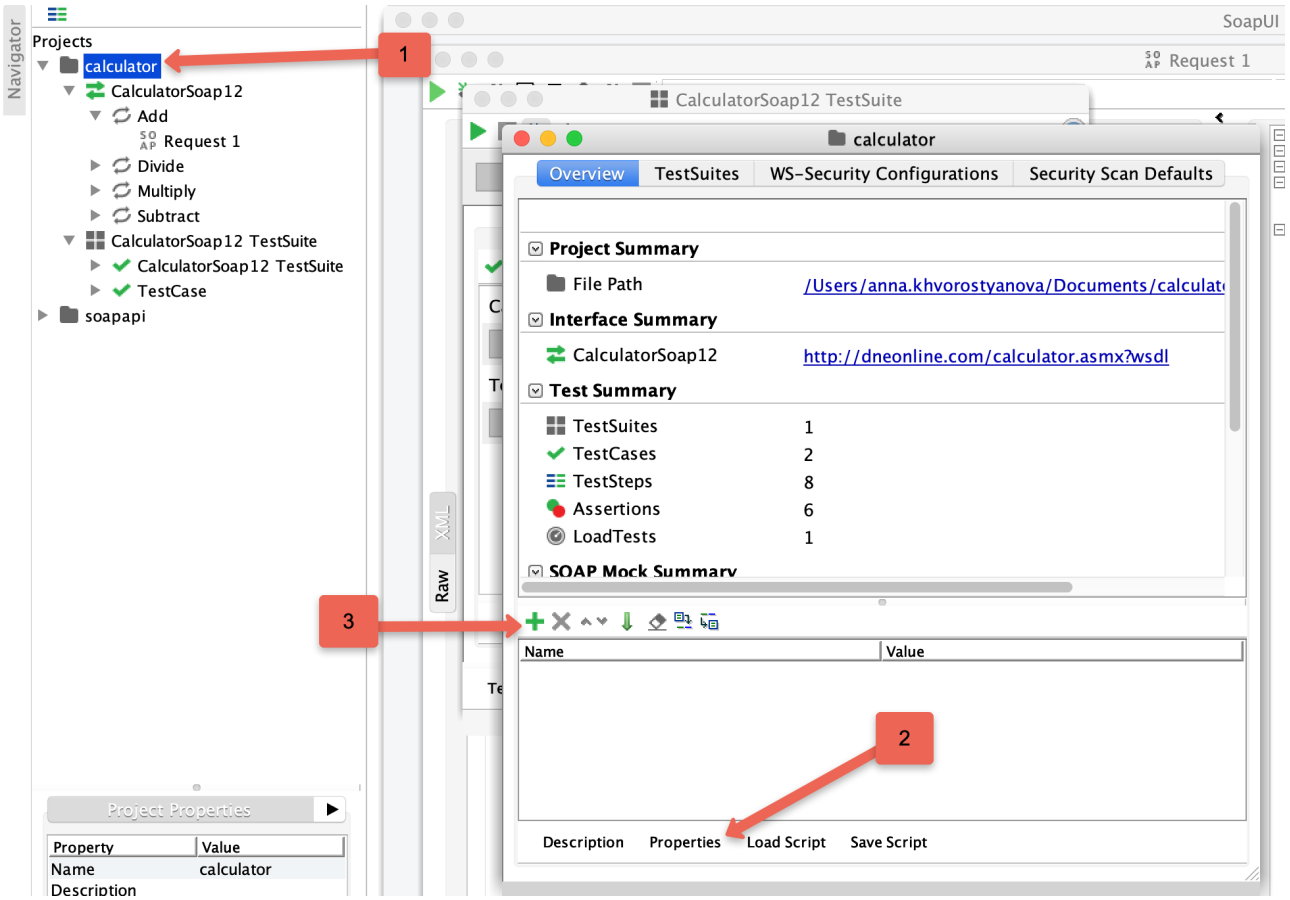
Установив переменную здесь, она станет вызываться как ${<название переменной>}, например:



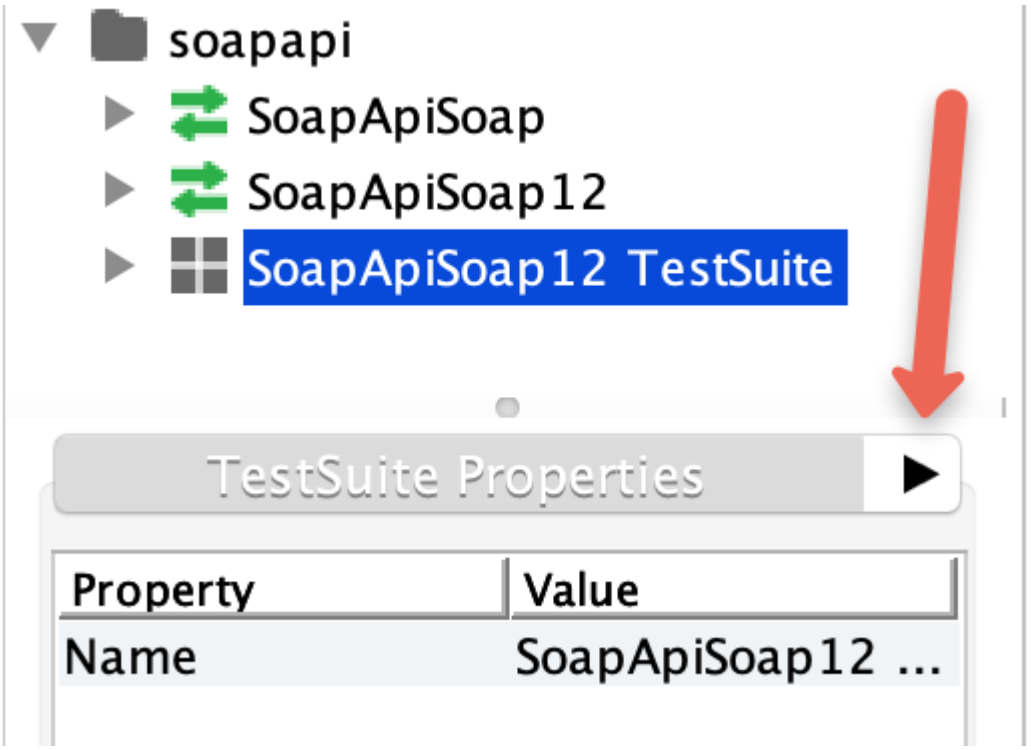
Они используются, например, для эндпоинтов разных стендов, а также для других значений, которые сложно запомнить и приходится часто использовать.

Переменные проекта, сьюта, тест-кейса

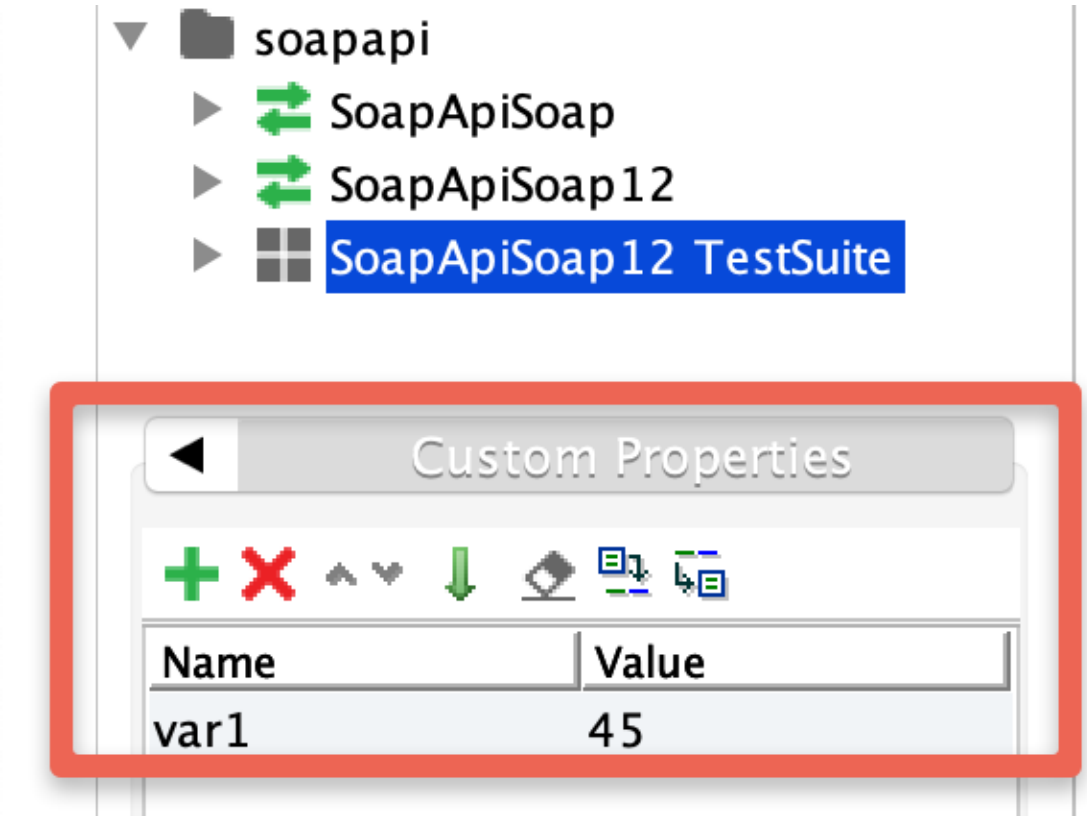
Как мы видим из названия, и проект, и тест-сьют, и тест-кейс в SoapUI имеют свои наборы свойств. Они доступны, если дважды щёлкнуть на сущность, например, проект, а затем на пункт Properties:



Список свойств также можно увидеть, если щёлкнуть единожды на сущность, а затем — на стрелочку рядом с Project/TestSuite/TestCase Properties, чтобы перейти к пользовательским свойствам (Custom Properties):



Результат:



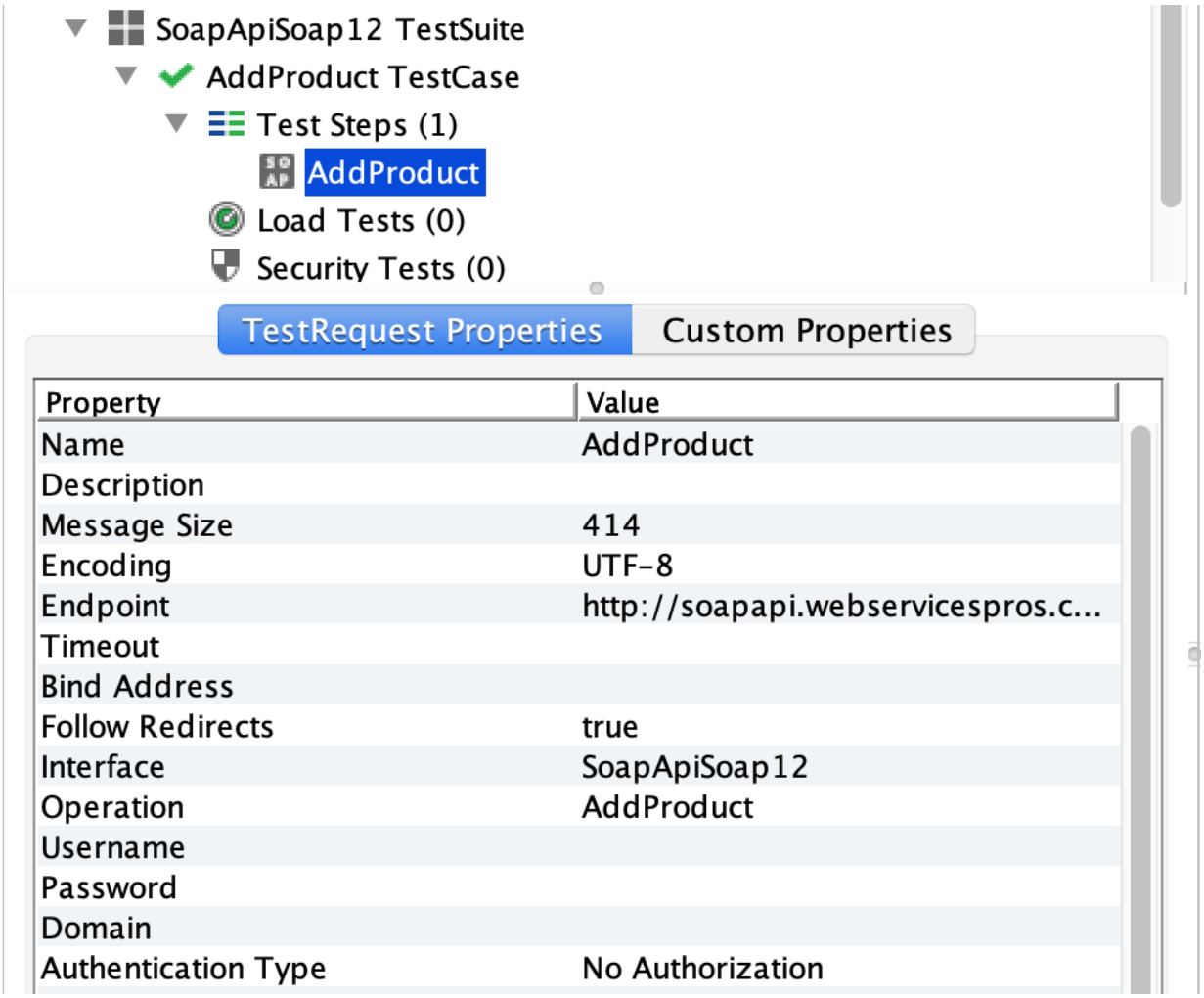
Обращаемся к таким пропертям из запроса, используя:

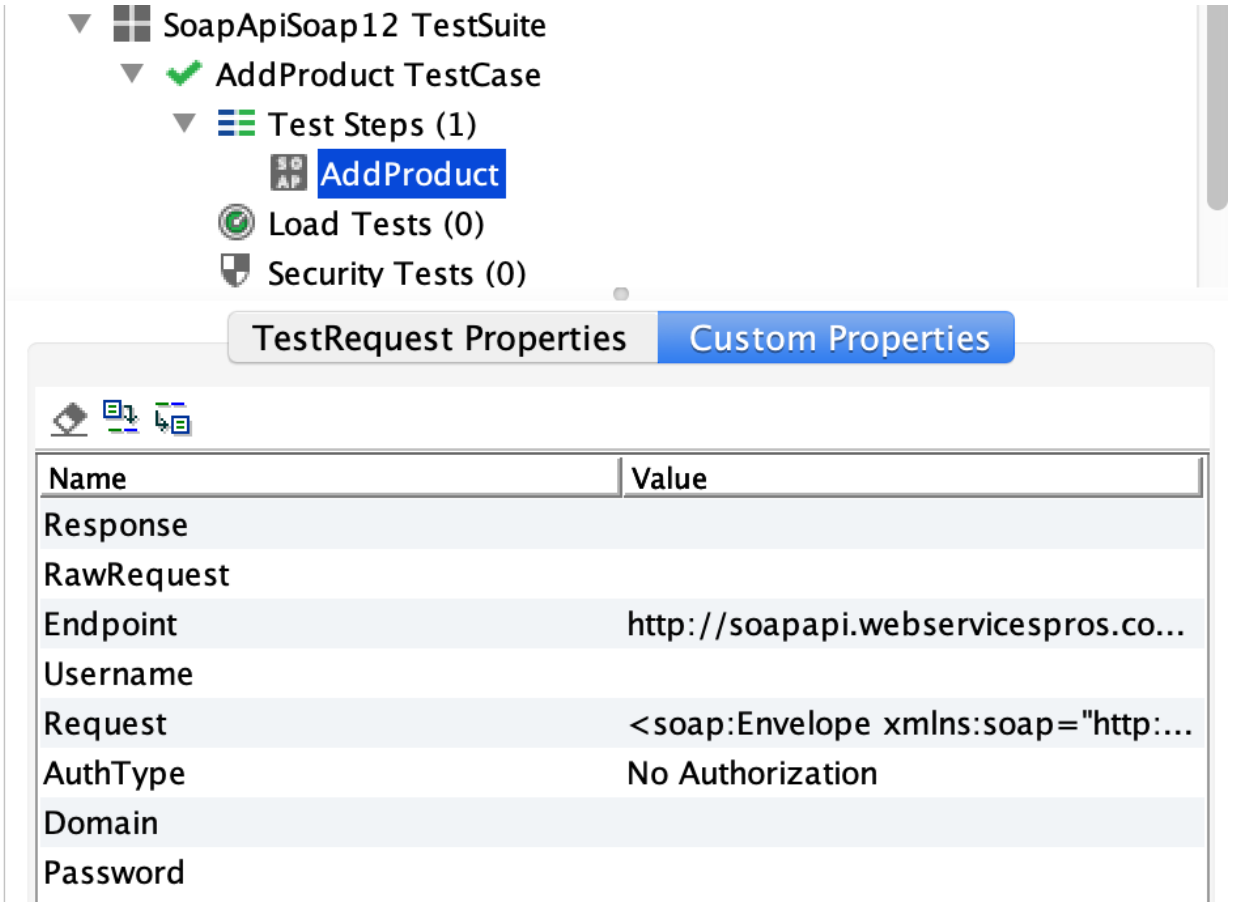
1. ${#Project#название_переменной} — для проектных переменных.
2. ${#TestSuite#название_переменной} — для переменных тест-сьюта.
3. ${#TestCase#название_переменной} — для переменных тест-кейса.

Таким же образом создаём переменные MockService и обращаемся к ним через ${#MockService#название_переменной}.

Переменные шагов теста

К переменным также можно обращаться в тестовых шагах кейса, они предзаданы и изменяются автоматически при запуске теста:


SoapApiSoap12 TestSuite
AddProduct TestCase
Test Steps (1)
AddProduct
Load Tests (0)
Security Tests (0)
TestRequest Properties
Custom Properties
Property
Value
Name
AddProduct
Description
Message Size
414
Encoding
UTF-8
Endpoint
http://soapapi.webservicespros.c...
Timeout
Bind Address
Follow Redirects
true
Interface
SoapApiSoap12
Operation
AddProduct
Username
Password
Domain
Authentication Type
No Authorization



Эти шаги выделяются в отдельную категорию, так как обращение к ним осуществляется через ${#НазваниеШага#Название переменной}. Например, на изображении выше для получения значения переменной Response требуется писать ${#AddProduct#Response}.

Очевидно, что у таких переменных ограниченная зона видимости, и за пределами тест-кейса они будут недоступны.

В SoapUI есть и динамические переменные.

Динамические переменные

В теле запроса используются вставки для рандомизации значений:

```
${=(int)(Math.random()*1000)}
```

Этот скрипт подставляет новое значение от 0 до 999 для каждого запуска.

Чтобы предоставить текущий форматированный timestamp в виде даты, прописываем следующее:

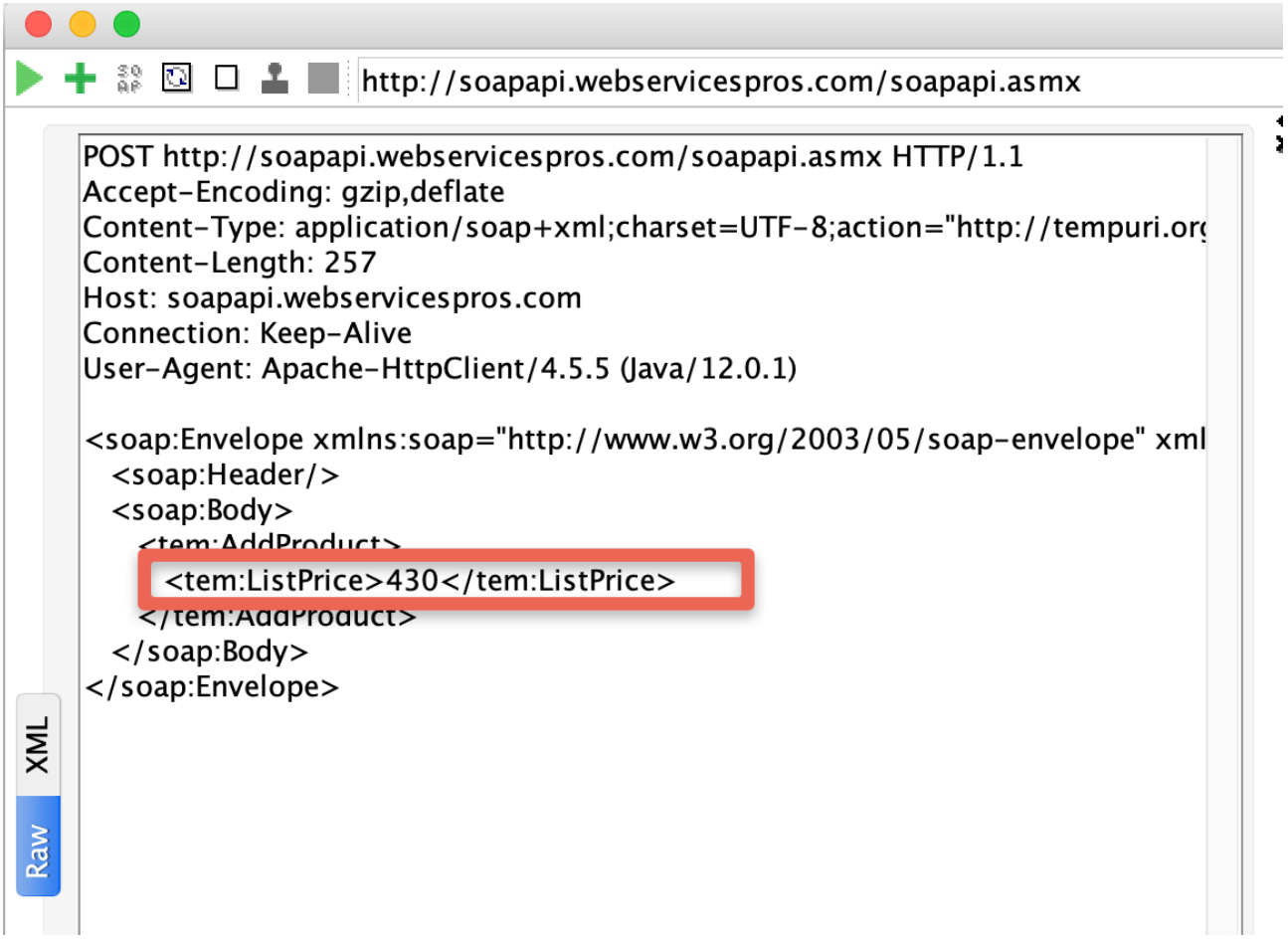
```
${=import java.text.SimpleDateFormat ; new
SimpleDateFormat("YYYY-MM-DDT00:00:00").format(new Date())}
```

Новый UUID генерируется таким образом:

```
${=java.util.UUID.randomUUID()}
```

И так далее.

Чтобы посмотреть подставленное выражение, открываем вкладку Raw после выполнения запроса:



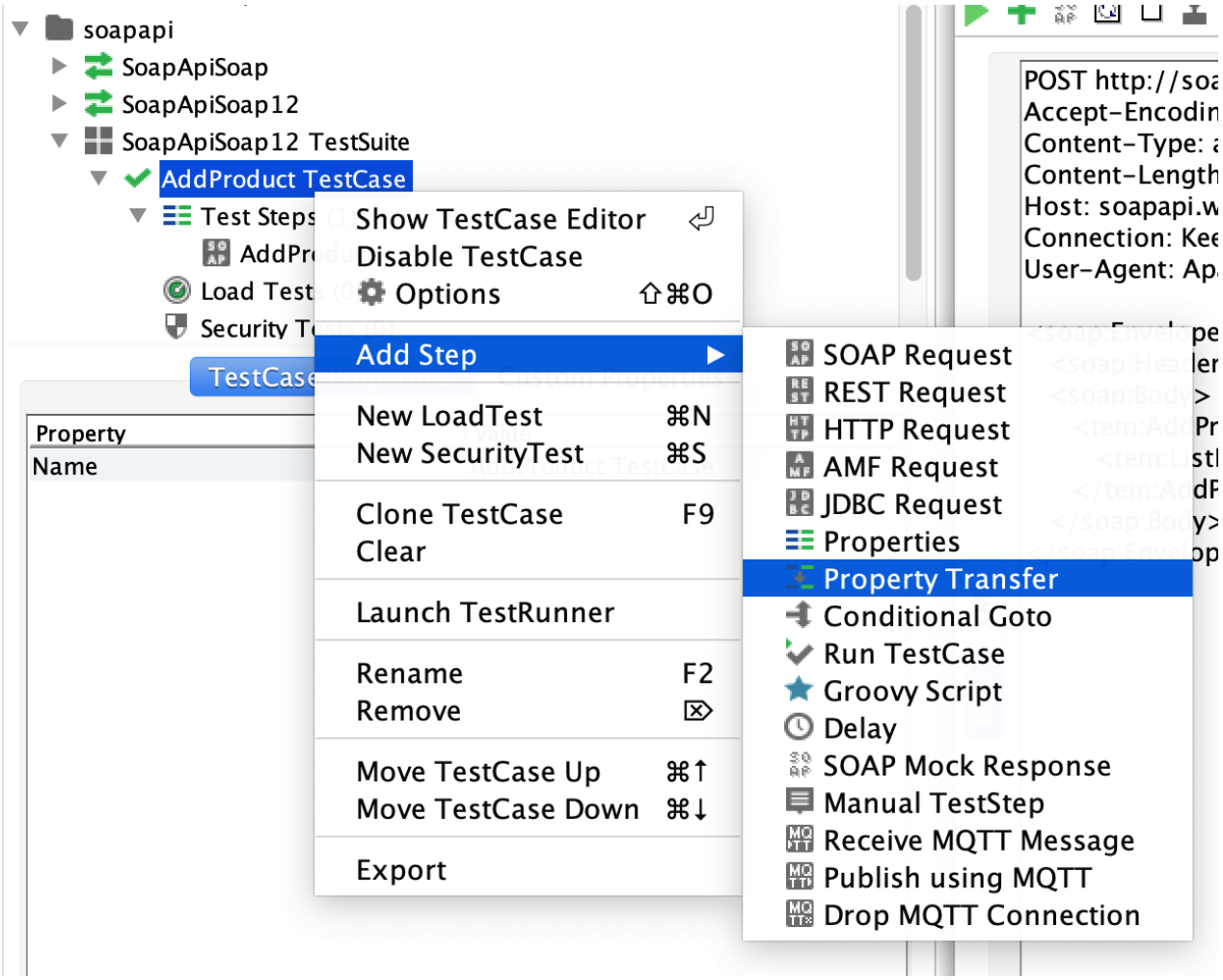
Эти и другие примеры — [в официальной документации](#) (на английском).

Property Transfer

Было бы здорово, если переменные из ответов одного шага передавались в другой, например:

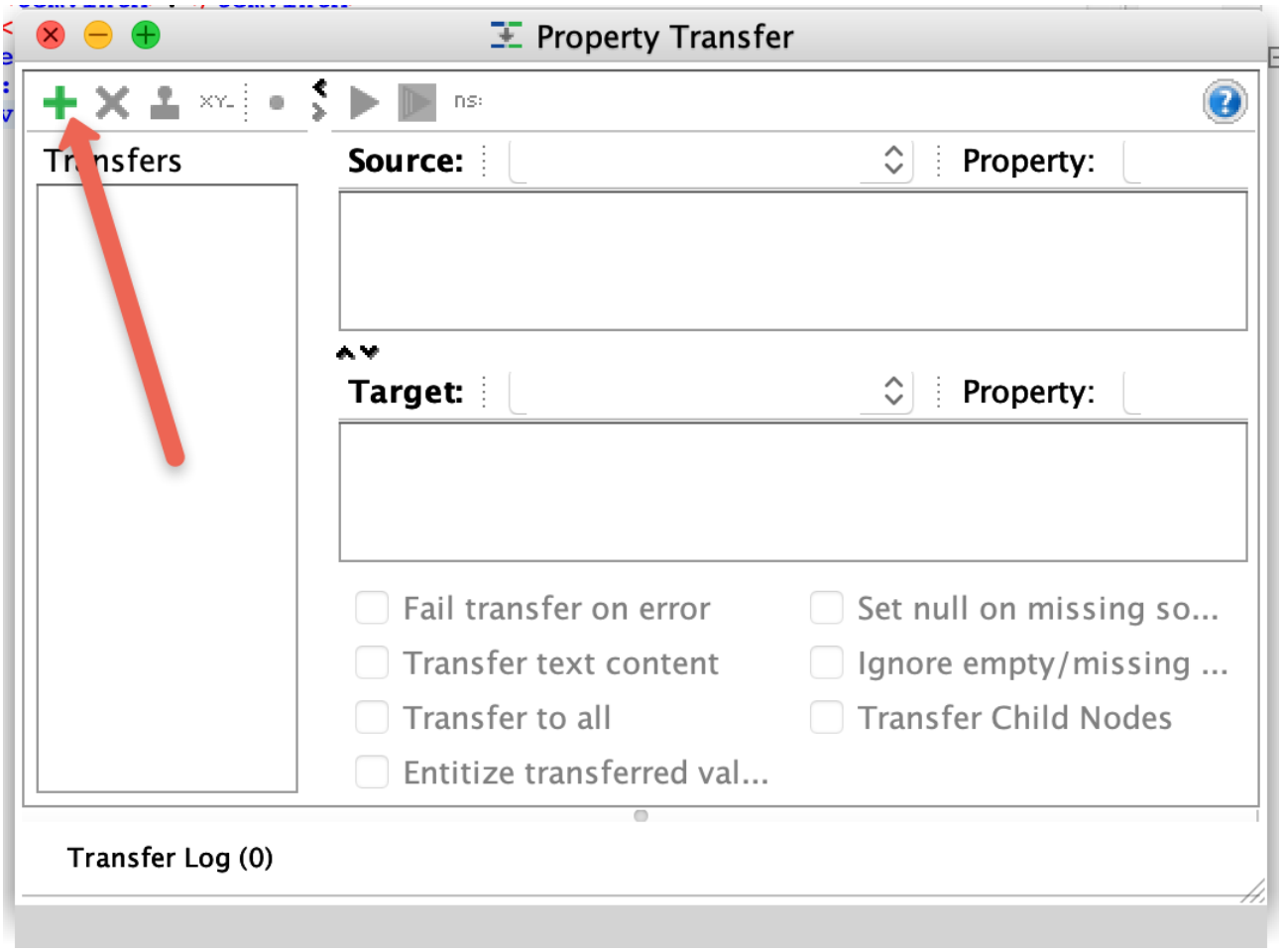
- создать продукт;
- получить его id;
- передать в запрос на изменение или удаление.

И такая возможность есть! Для этого используется специальный шаг по «транспорту» свойств — Property Transfer. Он создаётся в любом тест-кейсе:

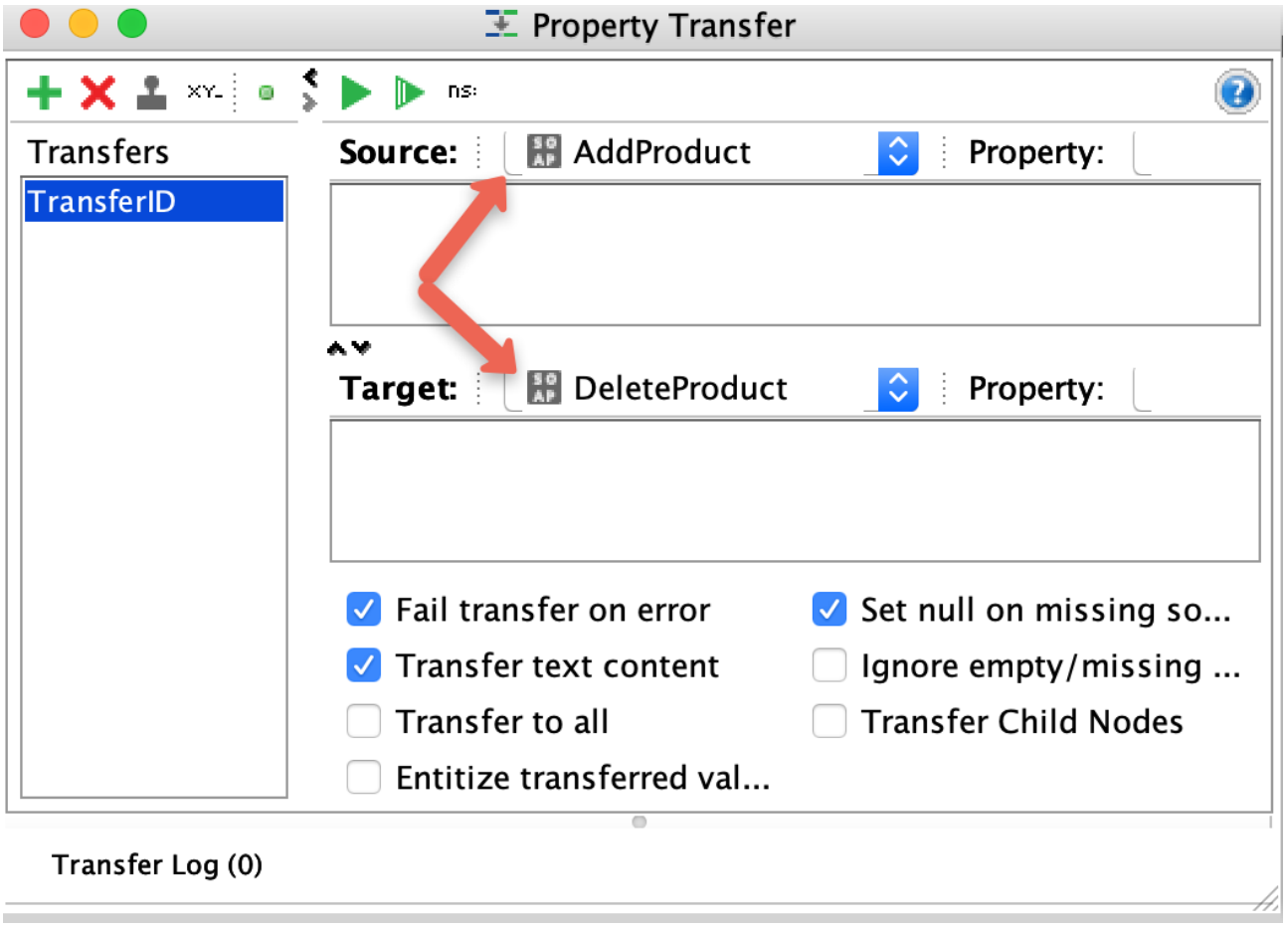


Допустим, что у нас есть тест-кейс, состоящий из двух операций: создания продукта и его последующего удаления. Добавим шаг Property Transfer между ними:

Откроем шаг Property Transfer и нажмём на кнопку «+»:

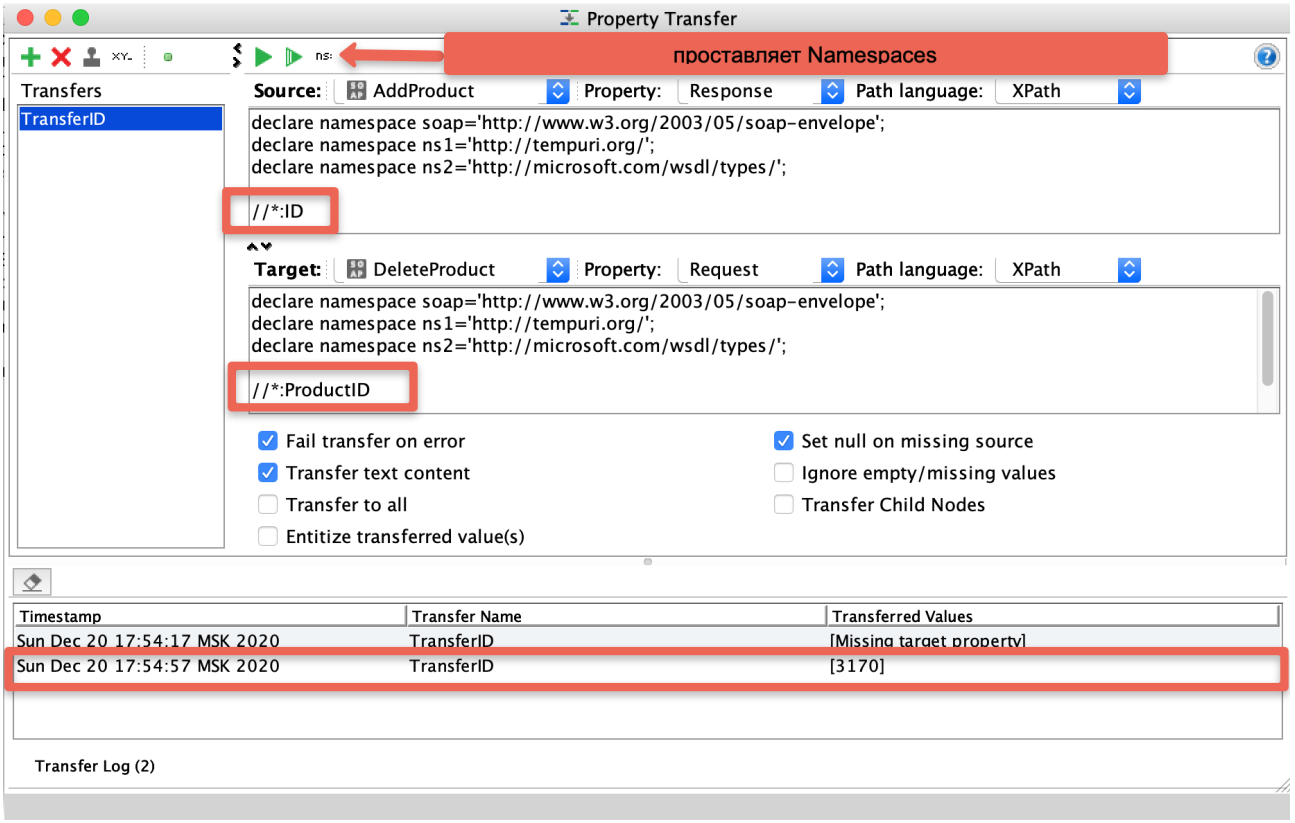


После ввода названия требуется выбрать источник свойства и его назначения. В нашем случае они подставляются автоматически благодаря заранее выбранному местоположению шага:



Далее для обоих полей, источника и целевого запроса, надо выбрать конкретные поля. И здесь нам пригодится XPath. Подробнее — в методичке к курсу о тестировании веб-приложений.

1. Нажимаем на кнопку Declare Namespaces, она скрыта за аббревиатурой ns:.
2. В первом случае выбираем Response и пишем xpath, чтобы получить значение id.
3. Для целевого поля выбираем значение Request и пишем xpath, чтобы указать, куда надо вставить значение. Получится примерно так:



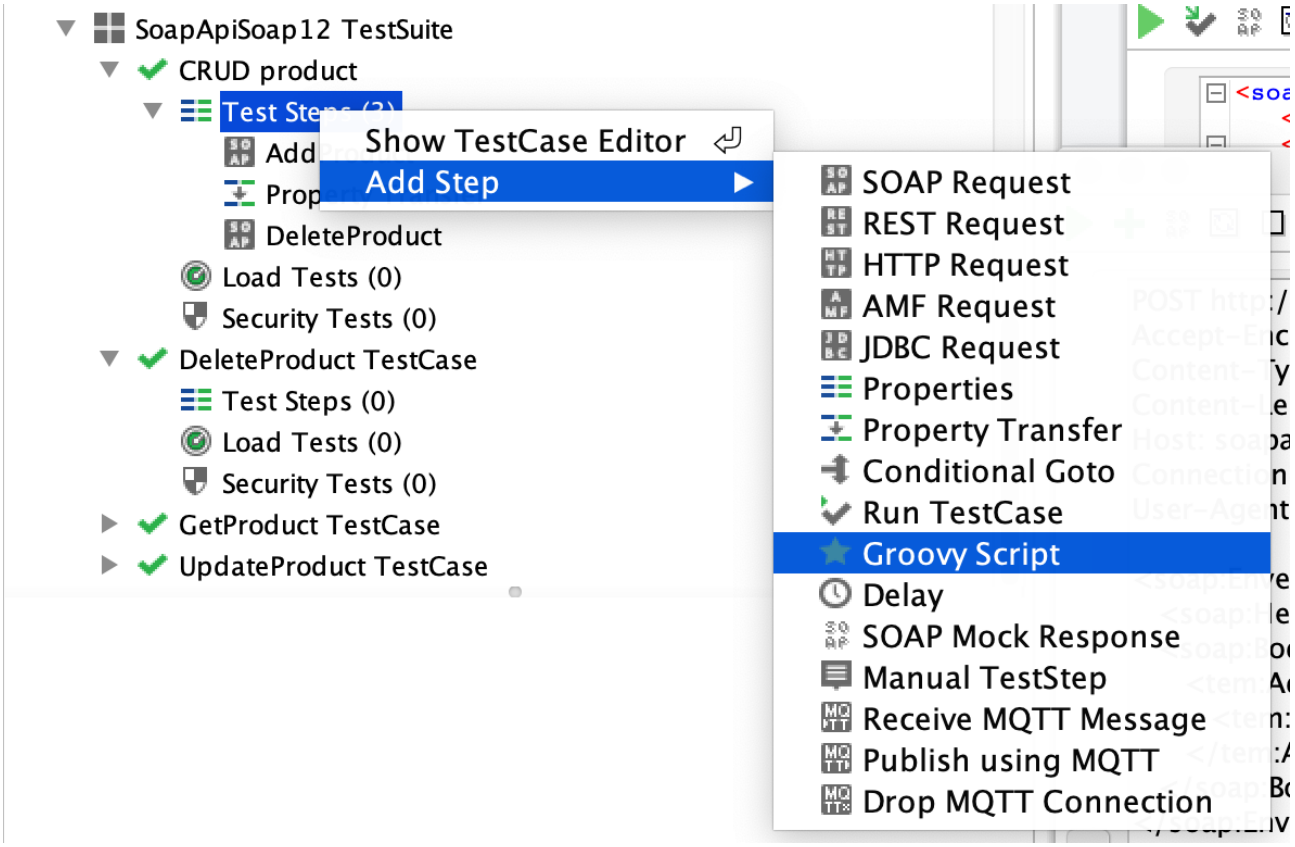
На месте namespace у каждого тега стоит *. Это означает, что мы их пропускаем, и нам подходит тег из любого пространства имён.

4. Запускаем трансфер и видим, что в логе появилось конкретное значение. Важно, чтобы исходный шаг теста хотя бы раз успешно запустился и имел Response.

5. Открываем DeleteProduct и видим, что значение подставилось:

Groovy Script

Создаётся так же, как и любой другой шаг, через контекстное меню тест-кейса или тестовых шагов:



«Груви» — это язык JVM, и, что важно, любой код на Java считается валидным кодом на groovy. Подробнее о groovy — здесь..

Через скрипты можно запускать шаги и тест-кейсы, получать значения свойств и создавать новые свойства любого уровня.

В основном мы будем использовать функцию управления свойствами.

Получение переменных

1. Получить переменную тест-кейса:

```
def testCaseProperty = testRunner.testCase.getPropertyValue("MyProp")
```

2. Получить переменную тест-сьюта:

```
def testSuiteProperty = testRunner.testCase.testSuite.getPropertyValue( "MyProp"
)
```

3. Получить переменную проекта:

```
def projectProperty = testRunner.testCase.testSuite.project.getPropertyValue(
"MyProp" )
```

4. Получить глобальную переменную:

```
def globalProperty =
com.eviware.soapui.SoapUI.globalProperties.getPropertyValue( "MyProp" )
```

Создание переменных и сеттинг значений

Важно! Типа основных переменных — String. Для других переменных требуется сделать приведение типов, например, через String.valueOf(var1).

1. Создать переменную тест-кейса со значением:

```
testRunner.testCase.setPropertyValue( "MyProp", someValue )
```

2. Создать переменную тест-сьюта со значением:

```
testRunner.testCase.testSuite.setPropertyValue( "MyProp", someValue )
```

3. Создать переменную проекта со значением:

```
testRunner.testCase.testSuite.project.setPropertyValue( "MyProp", someValue )
```

4. Создать глобальную переменную:

```
com.eviware.soapui.SoapUI.globalProperties.setPropertyValue( "MyProp", someValue
)
```

Сохранение данных в файл

Ещё один полезный трюк с использованием Groovy Steps — это сохранение данных (как правило, ответов) в файл.

Сначала получаем ответ сервера по названию шага теста AddProduct, а затем сохраняем в новый файл:

```
def response = context.expand( '${AddProduct#Response}' )
new File( "C:/Users/anna/" + new Random.nextInt(1,1000) + "_response.txt"
).write( response );
```

Чтобы не писать абсолютный путь, лучше воспользоваться специальной переменной projectDir при условии, что проект уже сохранился. Но для этого её надо импортировать:

```
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context)
def projectDir = groovyUtils.projectPath
```

Тогда сохранение в файл будет выглядеть так:

```
new File(projectDir, new Random.nextInt(1,1000) + "_response.txt" ).write(
response );
```

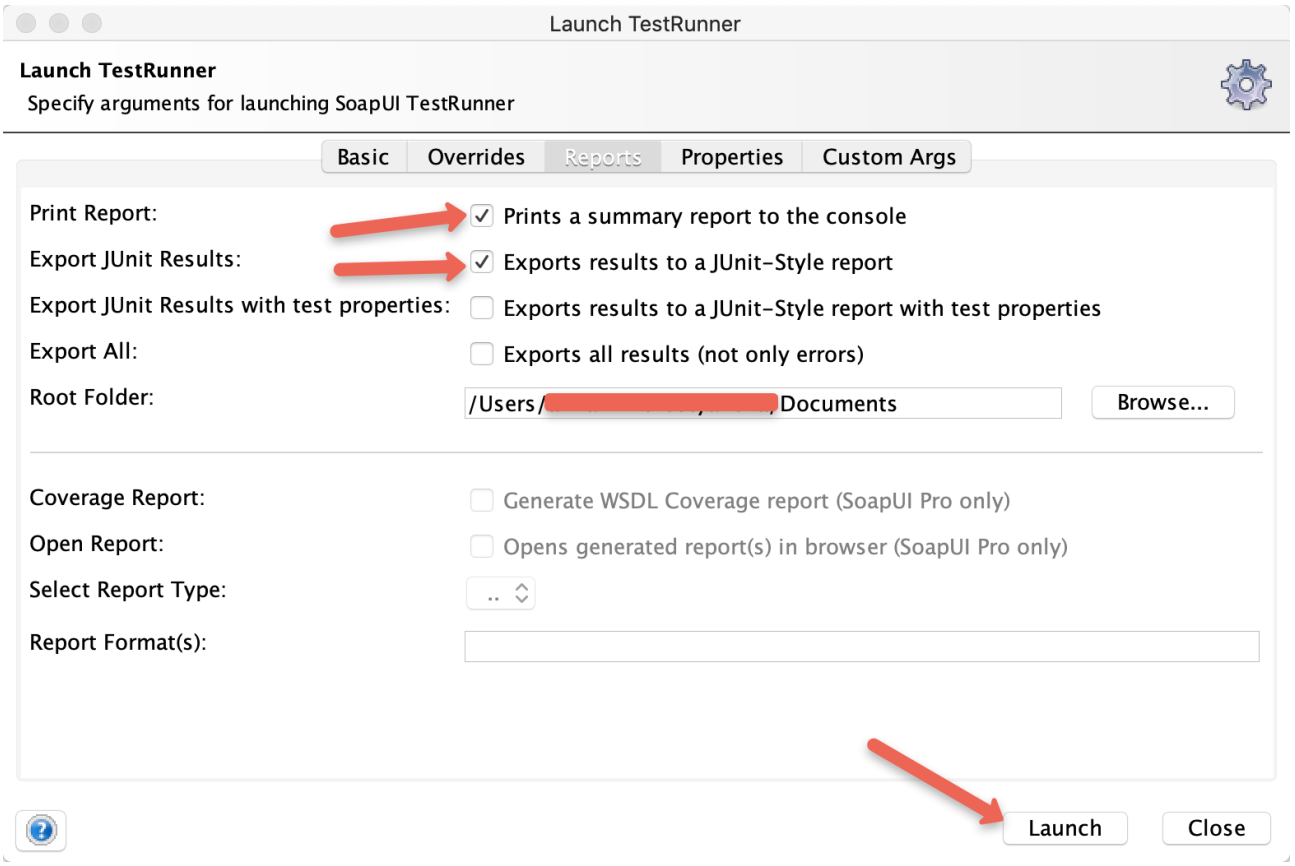
Больше информации — в документации.

Assertions

Assertions подробно разбирались в курсе по тестированию веб-приложений. Для более полной информации можно обратиться к официальной документации. Отметим, что наиболее надёжные — XPath (XQuery) assertions и Groovy Assertions. Они позволяют указывать конкретные поля для проверок. Для прогонов тестов каждый шаг с запросом должен содержать хотя бы один ассерт.

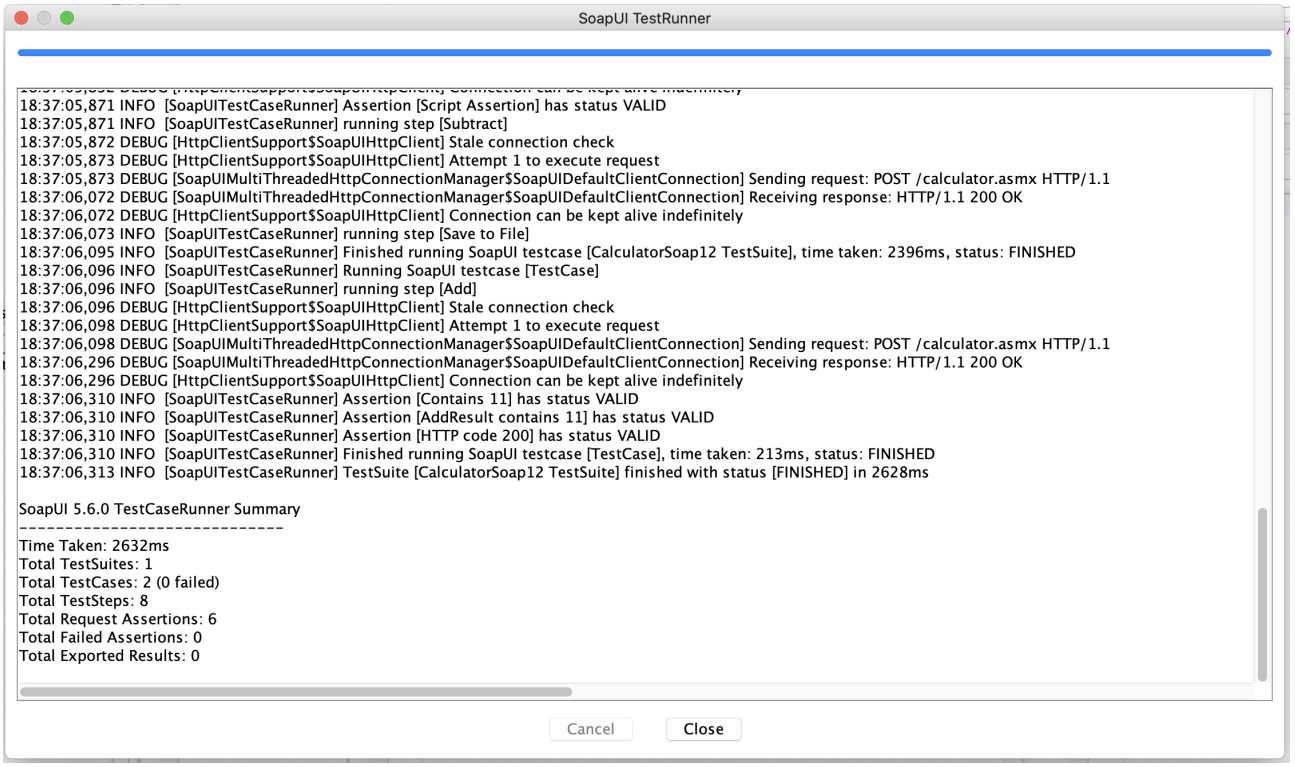
Отчётность

После составления тестового набора можно запустить его целиком. Для этого требуется нажать правой кнопкой мыши на сьют или тест-кейс и выбрать Launch TestRunner. Чтобы настроить отчётность, надо перейти на вкладку Reporting и поставить следующие галочки:

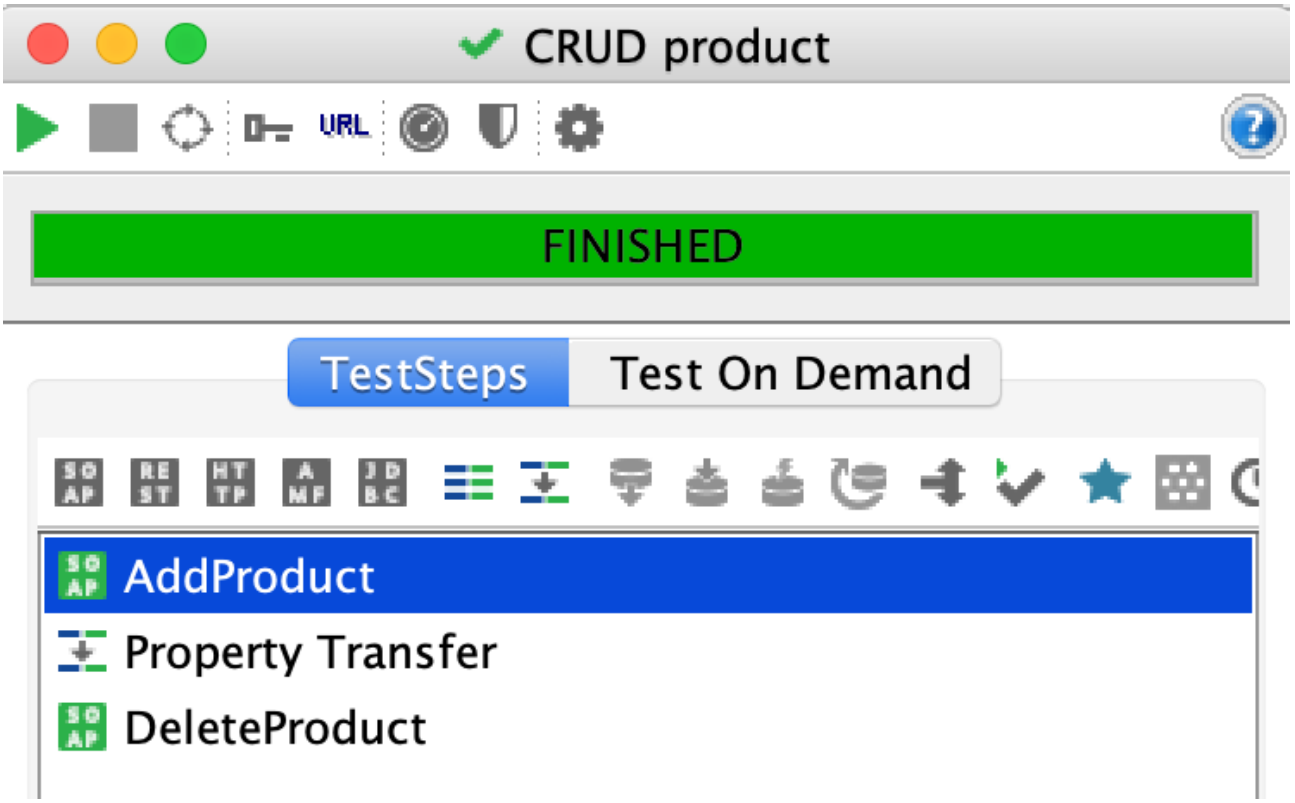


Либо указать Root Folder, куда будет складываться информация по запуску сьюта или тест-кейса в xml-формате.

После запуска откроется консоль, и мы увидим такую картину:



Если специфическая информация не требуется, то для тестовых целей запускаем тест-кейс или тест-сьют, дважды нажав на него, а затем — на клавишу Run:



Зелёным подсветятся пройденные шаги, красным — упавшие. Но это будет работать лишь тогда, когда у шагов с запросами будет хотя бы один Assertion.

Практическое задание

1. Автоматизируйте тестирование сервиса калькулятора: минимум 2 операции из четырёх.
2. Выполните примерно 10 проверок на каждый эндпоинт и минимум 2 интеграционных кейса.

Требования к практическому заданию

1. Формат практического задания: файл xml (или zip) с проектом.
2. Для каждого теста пропишите assertions.
3. Воспользуйтесь Property Transfer и Groovy Step.
4. Примените переменные сьюта или кейса. Не используйте глобальные проперти, они не экспортируются вместе с проектом!

Дополнительные материалы

1. Статья API.
2. Статья SOAP API.
3. Статья «Что такое SOAP?»
4. Сайт SoapUI.
5. Статья «Утверждение SoapUI — совпадение XPath».

Используемые источники

1. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.