

|  |
| --- |
| Subject  Data Driven Testing Framework Overview |
| EPM-SPI |

| Related Artifacts | |
| --- | --- |
| Ref. | Name |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Abbreviations and Acronyms | |
| DDT | [Data Driven Testing](https://ctc-customs.atlassian.net/wiki/pages/viewpage.action?pageId=45318221) |
| SoapUI | The web service testing application for service-oriented architectures (SOA) and representational state transfers (REST) |
| SoapUI project | In SoapUI, work is organized into projects, which are displayed under the root node in the workspace navigator. A project can contain any number of functional tests, load tests and service simulations required for testing purposes. |
| DDT Framework Library | Library of scrips created on Groovy to improve testing process of [DDT](https://ctc-customs.atlassian.net/wiki/pages/viewpage.action?pageId=45318221) in SoapUI |
| DDT Framework Plugin | Plugin for SoapUI developed with DDT Framework Library which adds the test data descriptions to the Test Suite Log during test execution and add user cookies to request if it required |

Contents

[1 description 5](#_Toc453946273)

[2 test data file format 5](#_Toc453946274)

[2.1 File processing rules 5](#_Toc453946275)

[2.2 field description 5](#_Toc453946276)

[2.2.1 Endpoint 6](#_Toc453946277)

[2.2.2 Method 6](#_Toc453946278)

[2.2.3 Resource Path 6](#_Toc453946279)

[2.2.4 Assertion Results 6](#_Toc453946280)

[2.2.5 Expected Response 7](#_Toc453946281)

[2.2.6 Evaluation Script for Response 7](#_Toc453946282)

[2.2.7 Expected Content-Type 7](#_Toc453946283)

[2.2.8 Expected HTTP Status 7](#_Toc453946284)

[2.2.9 Request Media Type 7](#_Toc453946285)

[2.2.10 Request Body 7](#_Toc453946286)

[2.2.11 Evaluation Script for Request 7](#_Toc453946287)

[2.2.12 Request Headers 8](#_Toc453946288)

[2.2.13 Evaluation Script for Request Headers 8](#_Toc453946289)

[2.2.14 Parameters 8](#_Toc453946290)

[2.2.15 Annotations 8](#_Toc453946291)

[3 DDT Framework package 9](#_Toc453946292)

[3.1 Package Structure 9](#_Toc453946293)

[3.2 Build DDT Framework library package 9](#_Toc453946294)

[3.3 Add DDT Framework library to SoapUI 10](#_Toc453946295)

[3.4 Build DDT Framework Plugin for soapUI 11](#_Toc453946296)

[3.5 Add DDT Framework Plugin to SoapUI 12](#_Toc453946297)

[3.6 Resolving Properties in Test Data 13](#_Toc453946298)

[3.7 Resolving Variables in Test Data 13](#_Toc453946299)

[3.8 Evaluation Scripts 14](#_Toc453946300)

[3.8.1 Extract Variables 14](#_Toc453946301)

[3.8.2 Set Variables 14](#_Toc453946302)

[3.8.3 Get Variables 15](#_Toc453946303)

[3.8.4 Download and Verify PKPASS File 15](#_Toc453946304)

[4 Soap UI Project 15](#_Toc453946305)

[4.1 Soapui file encoding configuration 15](#_Toc453946306)

[4.2 Project Custom Properties 16](#_Toc453946307)

[4.3 Properties from Configuration File 17](#_Toc453946308)

[4.4 User-Defined Cookie 17](#_Toc453946309)

[4.5 TestSuite Custom Properties 18](#_Toc453946310)

[4.6 TestCase Custom Properties 19](#_Toc453946311)

[4.7 Data Driven Test Scenario 20](#_Toc453946312)

[4.7.1 Test Case Structure 20](#_Toc453946313)

[4.7.2 ‘Before’ Test Step 22](#_Toc453946314)

[4.7.3 ‘data driver’ Test Step 23](#_Toc453946315)

[4.7.4 REST Request Test Steps 24](#_Toc453946316)

[4.7.5 JDBC Request Test Steps 24](#_Toc453946317)

[4.7.6 ‘PAUSE’ Test Step 25](#_Toc453946318)

[4.7.7 ‘WAIT’ Test Step 25](#_Toc453946319)

[4.7.8 ‘goto Looper’ Test Steps 25](#_Toc453946320)

[4.7.9 ‘Looper’ Test Steps 25](#_Toc453946321)

[5 Test Execution and Result Validation 26](#_Toc453946322)

[5.1 Start Test Suite Execution 26](#_Toc453946323)

[5.2 View Test Suite Execution Log 27](#_Toc453946324)

[5.3 View DDT Framework Log 29](#_Toc453946325)

[5.4 Test Step Result Validation 30](#_Toc453946326)

[5.5 Closing SoapUI 35](#_Toc453946327)

[Appendix A. test data file Examples 36](#_Toc453946328)

[Appendix A.1. property references in data file 36](#_Toc453946329)

[Appendix A.2. Variable mappinfg file and variable references in data file 36](#_Toc453946330)

[Appendix A.3. Extract Variable in data file 37](#_Toc453946331)

[Appendix A.4. JDBC REquest in data file 37](#_Toc453946332)

[Appendix B. Troubleshooting 38](#_Toc453946333)

[Appendix B.1. Build Task for DDT Framework Failed 38](#_Toc453946334)

[Appendix B.2. Properties from Configuration File is not available in SoapUI Project 38](#_Toc453946335)

[Appendix B.3. Assertion Messages for Failed Steps are too long 38](#_Toc453946336)

[Appendix B.4. Tests are not work or Data-Driver is not works 38](#_Toc453946337)

[Appendix B.5. The “soapui.log” File is Missed 39](#_Toc453946338)

[Appendix B.6. Nothing is helped 39](#_Toc453946339)

# description

DDT Framework consists of DDT Framework Library, DDT Framework Plugin for SoapUI, SoapUI project and Test Data files. DDT Framework Library and DDT Framework Plugin created with Groovy language, Spock testing framework and Gradle build automation tool. SoapUI project with regression tests created and executed with SoapUI Community Edition. Test Data files created in Microsoft Excel and contains test data for data driving testing.

# test data file format

The test data files define test scenarios. Each test data file describes test steps with parameters and expected results. Data files may have Excel 97-2003 Workbook (\*.xls) or Excel Workbook (\*.xlsx) format.

1. Please see Appendix A for Test Data file examples.

## File processing rules

The DDT Framework Library uses following workflow for data file processing:

1. Test data file processed row by row to the row with empty CELL\_DESCRIPTION field.
2. All rows, which start with hash character ‘#’ in the CELL\_DESCRIPTION field, considered as comment lines and skipped.

## field description

All data files should have following fields by convention:

Table 1 Field descriptions for data file

| List of fields in data file | | |
| --- | --- | --- |
| Field Name | Column index | Description |
| CELL\_DESCRIPTION | 0 | Step description, row will be skipped if the first symbol in cell is "#" |
| CELL\_ENDPOINT | 1 | Endpoint for request. If this field is empty then will be used the default endpoint defined in properties |
| CELL\_METHOD | 2 | Name of test step which will be called (usually same as REST method) |
| CELL\_PATH | 3 | Resource path for request (e.g. "/manage/info") |
| CELL\_ASSERTION\_RESULT | 4 | Expected result for assertion script. If it is true then actual result should be equal to expected, else they should be different |
| CELL\_RESPONSE\_BODY | 5 | Expected response body (use @Skip to avoid data assertion) |
| CELL\_RESPONSE\_EVAL\_SCRIPT | 6 | Evaluate script to process actual result before assertion with CELL\_RESPONSE\_BODY |
| CELL\_RESPONSE\_CONTENT\_TYPE | 7 | Expected content-type of response (use @Skip to avoid data assertion) |
| CELL\_RESPONSE\_STATUS | 8 | Expected HTTP code of response (use @Skip to avoid data assertion) |
| CELL\_REQUEST\_MEDIA\_TYPE | 9 | Media-type of request |
| CELL\_REQUEST\_BODY | 10 | Request body |
| CELL\_REQUEST\_EVAL\_SCRIPT | 11 | Evaluate script to process request body |
| CELL\_REQUEST\_HEADERS | 12 | Headers for request |
| CELL\_REQUEST\_HEADERS\_EVAL\_SCRIPT | 13 | Evaluate script to process headers |
| CELL\_REQUEST\_PARAMETER\_NAME | 14 | Cells with parameter set |
| CELL\_REQUEST\_PARAMETER\_VALUE | 15 |
| CELL\_REQUEST\_PARAMETER\_EVAL\_SCRIPT | 16 |
| repeat CELL\_REQUEST\_PARAMETER\_... |  |  |

### Endpoint

If this field is not empty then need to use its value as endpoint for REST request. Otherwise, use the endpoint from the SoapUI project properties or configuration file.

### Method

This field contains test step name for execution. The following test steps available:

1. “GET” – HTTP method GET for the REST request;
2. “POST” – HTTP method POST for the REST request;
3. “PUT” - HTTP method PUT for the REST request;
4. “DELETE” - HTTP method DELETE for the REST request;
5. “OPTIONS” - HTTP method OPTIONS for the REST request;
6. “JDBC” – JDBC request to data base;
7. “WAIT” – waits for specified amount of seconds before continue test execution;
8. “PAUSE” – some manual steps required before continue test execution.

### Resource Path

If method field corresponds to REST request (GET, PUT, POST, DELETE, OPTIONS) method then this field contains resource path for REST request.

If method field corresponds to JDBC request then this field contains connection string to database.

If method field corresponds to manual step “PAUSE” then this field contains description of manual steps.

If method field corresponds to manual step “WAIT” then this field contains the time duration in milliseconds to sleep.

1. Please see Appendix A for Test Data file examples.

### Assertion Results

This field should contain map of expected results for HTTP Status, Content-Type and Response assertion scripts.

Map defined in following format.

[“status”: true, “content-type”: false, “response”: true]

All keys are case sensitive. Values considered as Boolean. If some key is missing then its value considered as true by default.

[“response”: true] == [“response”: true, “status”: true, “content-type”: true]

[“response”: false, “content-type”: false] == [“response”: false, “status”: true, “content-type”: false]

It is possible to use “true” or “false” values as short form for following maps.

true == [“response”: true]

false == [“response”: false]

It is possible to leave the cell value empty that will mean that HTTP Status, Content-Type and Response assertions are true

### Expected Response

This field should contain expected response for assertion or one of the following annotations:

1. “@SKIP” – declares that assertion of response should be skipped;
2. “@LASTRESPONSE” – declares that response from previous REST Request should be used for response assertion; or
3. “@LASTEXPECTED” – declares that expected response from previous test step should be used for response assertion.

### Evaluation Script for Response

This field should contain groovy script to evaluate actual response before assertion. If it is empty then none evaluation for response is required.

Example of groovy script:

import org.jsoup.Jsoup

def html = Jsoup.parse(x)

html.body().html()

It is also possible to use the “context” and “messageExchange” variables in evaluation script to get access to the TestCaseRunContext and MessageExchange objects of SoapUI. For example, the following evaluation script returns the first value of “Location” header from response.

messageExchange?.responseHeaders["Location"][0]

1. The evaluation script is applied only to the actual response. It means that evaluation script will not be applied to the expected result declared with “@LASTRESPONSE” or annotation. In this case, the raw response from previous REST Request will be used as expected response for response assertion.

### Expected Content-Type

This field should contain expected content-type for assertion or one of the following annotations:

1. “@SKIP” – declares that assertion of response should be skipped; or
2. “@COOKIE” – declares that assertion should be performed for cookies.

### Expected HTTP Status

This field should contain expected HTTP Status for assertion or the following annotation:

1. “@SKIP” – declares that assertion of response should be skipped.

### Request Media Type

This field should contain media type for REST Request. It may be empty if media type is not required.

### Request Body

This field should contain request content for REST Request or SQL expression for JDBC request. It may be empty if request content is not required. It is possible to use response of previous REST Request as request content with following annotation:

1. “@LASTRESPONSE” – declares that response from previous REST Request should be used as body for current request; or
2. “@LASTEXPECTED” – declares that expected response from previous test step should be used as body for current request.

### Evaluation Script for Request

This field should contain groovy script to evaluate request content before send the REST Request. If it is empty then none evaluation for request content is required.

1. It is possible to apply the evaluation script for request declared with “@LASTRESPONSE” and “@LASTEXPECTED” annotations.

### Request Headers

This field should contain map of Headers for REST Request.

Map defines in following format.

["Accept":"application/json", "Authorization":"Basic ${usr\_base\_testadmin}"]

All keys and values are case sensitive. It is possible to use response of previous REST Request as Headers for request with following annotation:

1. “@LASTRESPONSE” – declares that response from previous REST Request should be used as Headers for current request; or
2. “@LASTEXPECTED” – declares that expected response from previous test step should be used as Headers for current request.

### Evaluation Script for Request Headers

This field should contain groovy script to evaluate request Headers before send REST Request. If it is empty then none evaluation for request Headers is required.

1. Make sure that result of evaluation script have appropriate format for request headers and all keys and values encapsulated with double quotes.
2. It is possible to apply the evaluation script for request Headers declared with “@LASTRESPONSE” and “@LASTEXPECTED” annotations.

### Parameters

Each parameter defined by three fields starting with CELL\_REQUEST\_PARAMETER.

Table 2 Parameter field description

| Table heading | | |
| --- | --- | --- |
| Position | Description | Example |
| CELL\_REQUEST\_PARAMETER\_NAME | Parameter name. There are two types of parameters:   * QUERY parameter – default type; * Path/TEMPLATE parameter - when name enclosed by {} brackets. | * lang * {login} |
| CELL\_REQUEST\_PARAMETER\_VALUE | Parameter value. | testuser1@epam.com |
| CELL\_REQUEST\_PARAMETERS\_EVAL\_SCRIPT | Evaluate script to process parameter value. | x.bytes.encodeBase64().toString() |

It is possible to use response of previous REST Request as parameter value with following annotation:

1. “@LASTRESPONSE” – declares that response from previous REST Request should be used as parameter value for current request; or
2. “@LASTEXPECTED” – declares that expected response from previous test step should be used as parameter value for current request.

Field with evaluation script should contain groovy script to evaluate parameter value before send REST Request. If it is empty then none evaluation for parameter value is required.

1. It is possible to apply the evaluation script for parameter value declared with “@LASTRESPONSE” and “@LASTEXPECTED” annotations.

### Annotations

There are some annotations defined in the DDT Framework Library.

Table 3 Annotations available in DDT Framework Library

| Annotation List | |
| --- | --- |
| Annotation | Description |
| @LastResponse | Get the response body or cookies of last executed REST Request TestStep |
| @LastExpected | Get the expected response from previous TestStep |
| @Skip | Skip data assertion for defined field |
| @Cookie | Use “Set-Cookie” for response assertion |

1. Annotations are case insensitive.
2. It is possible to use the @LastResponse and @LastExpected annotations for expected response, request content, request headers and parameter values fields.

# DDT Framework package

## Package Structure

The latest version of DDT Framework is available on GIT repository <https://gitbud.epam.com/aleksei_galkin/ddt-framework.git> and have following structure:

1. ‘/docs’ – folder with library documentation, demo example and templates;
2. ‘/gradle’ – folder with gradle wrapper for build automation;
3. ‘/src’ – folder with sources and tests for library;
4. ‘build.gradle’ – gradle script with build configuration;
5. ‘gradlew’ and ‘gradlew.bat’ – gradle wrappers for Linux shell and Windows command line to build project without gradle installation;
6. ‘settings.gradle’ – settings for build automation tool; and
7. ‘README.md’ – basic instruction to run build automation tool.

## Build DDT Framework library package

Gradle build configuration contains following tasks:

1. ‘clean’ – clean artifacts of previous builds;
2. ‘build’ – compile source classes, creating JAR files for DDT Framework Library and DDT Framework Plugin, and run tests;
3. ‘jar’ – compile source classes and creating JAR files for DDT Framework Library and DDT Framework Plugin;
4. ‘copyLibs’ – compile source classes, creating JAR files for DDT Framework Library and DDT Framework Plugin, and copy JAR files to the ‘SOAPUI\_HOME/lib’ and ‘SOAPUI\_HOME/bin/plugins’ folders respectively if the ‘SOAPUI\_HOME’ is defined in system. It is possible to define SoapUI home directory with property ‘-Psoapui.home’.

gradlew copyLibs -Psoapui.home="C:\opt\SoapUI-5.1.3"

To build DDT Framework package need to perform following steps:

1. Locate to the ‘ddt-framework’ directory; and
2. Run the ‘gradlew.bat build’ command (or ‘./gradlew build’ command for Linux environment).

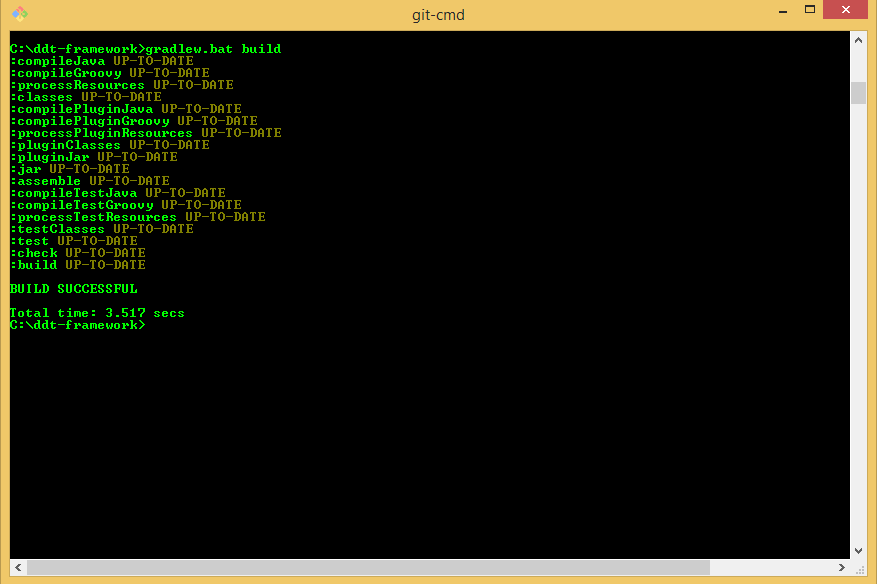


Figure 1 Build DDT Framework package from Command Prompt on Windows OS.

The JAR files for DDT Framework Library and DDT Framework Plugin could be found in ‘build\libs’ folder when build task successfully completed.

The ‘jar’ or ‘copyLibs’ tasks could be used instead of ‘build’ task if needed to create JAR files for DDT Framework Library and DDT Framework Plugin for SoapUI and skip tests or to copy the created JAR files to ‘SOAPUI\_HOME/lib’ and ‘SOAPUI\_HOME/bin/plugins’ folders respectively.

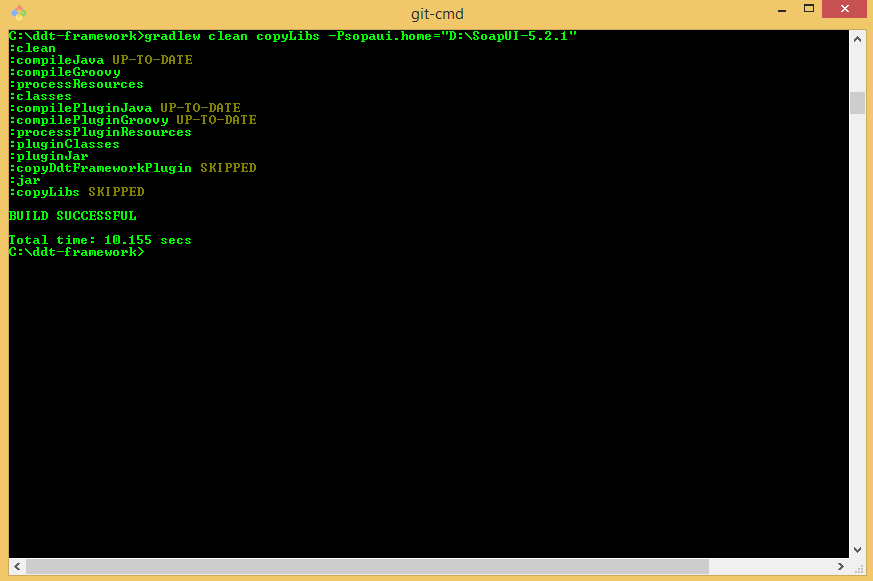


Figure 2 Clean project, create JAR files for DDT Framework package and copy to the ‘SOAPUI\_HOME’ folder from Command Prompt on Windows OS.

1. If build process is failed with some error try to use the “gradlew.bat jar” command instead to eliminate the library tests execution. If it does not helps please contact to the DDT Framework Team.

## Add DDT Framework library to SoapUI

To make DDT Framework library available in SoapUI, need to copy JAR file with DDT Framework library to the ‘SOAPUI\_HOME/lib’ folder where SOAPUI\_HOME is the installation folder of SoapUI. This may be done manually or with the ‘copyLibs’ gradle task.

1. Please see the Build DDT Framework library package section for more details on the ‘copyLibs’ gradle task.

When the DDT Framework Library is added to ‘SOAPUI\_HOME/lib’ folder, it is possible to call DDT Framework library functions from groovy scripts and assertion scripts in SoapUI using import or direct call.

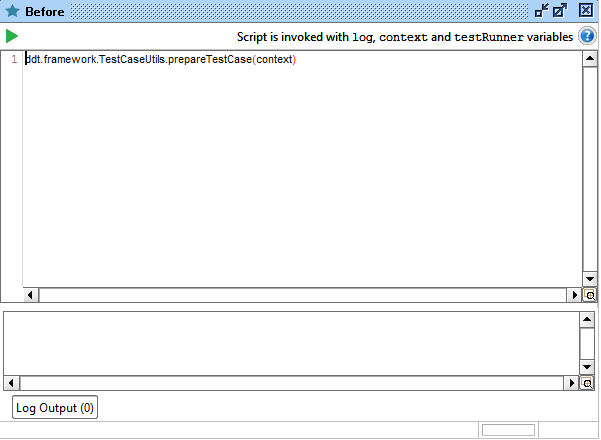


Figure 3 Example of method call from SoapUI Groovy Script test step.

## Build DDT Framework Plugin for soapUI

Gradle build configuration contains following task to build DDT Framework Plugin for SoapUI:

1. ‘pluginJar’ – compile source classes of DDT Framework Plugin and create a JAR.

To build DDT Framework Plugin for SoapUI need build DDT Framework library package or perform following steps:

1. Locate to the ‘ddt-framework’ directory; and
2. Run the ‘gradlew.bat pluginJar’ command (or ‘./gradlew build’ command for Linux environment).



Figure 4 Build DDT Framework Plugin for SoapUI from Command Prompt on Windows OS.

The JAR file for DDT Framework Plugin could be found in ‘build\libs’ folder when build task successfully completed.

## Add DDT Framework Plugin to SoapUI

To make DDT Framework plugin available in SoapUI, need to copy JAR file with DDT Framework Plugin to the ‘SOAPUI\_HOME/bin/plugins’ folder where SOAPUI\_HOME is the installation folder of SoapUI. If the ‘plugins’ folder does not exist under the ‘SOAPUI\_HOME/bin’ please create it.

When the DDT Framework Plugin is added to ‘SOAPUI\_HOME/bin/plugins’ folder, then the test step descriptions are added to the test suite log during test execution for more convenient navigation on test data and test execution results.

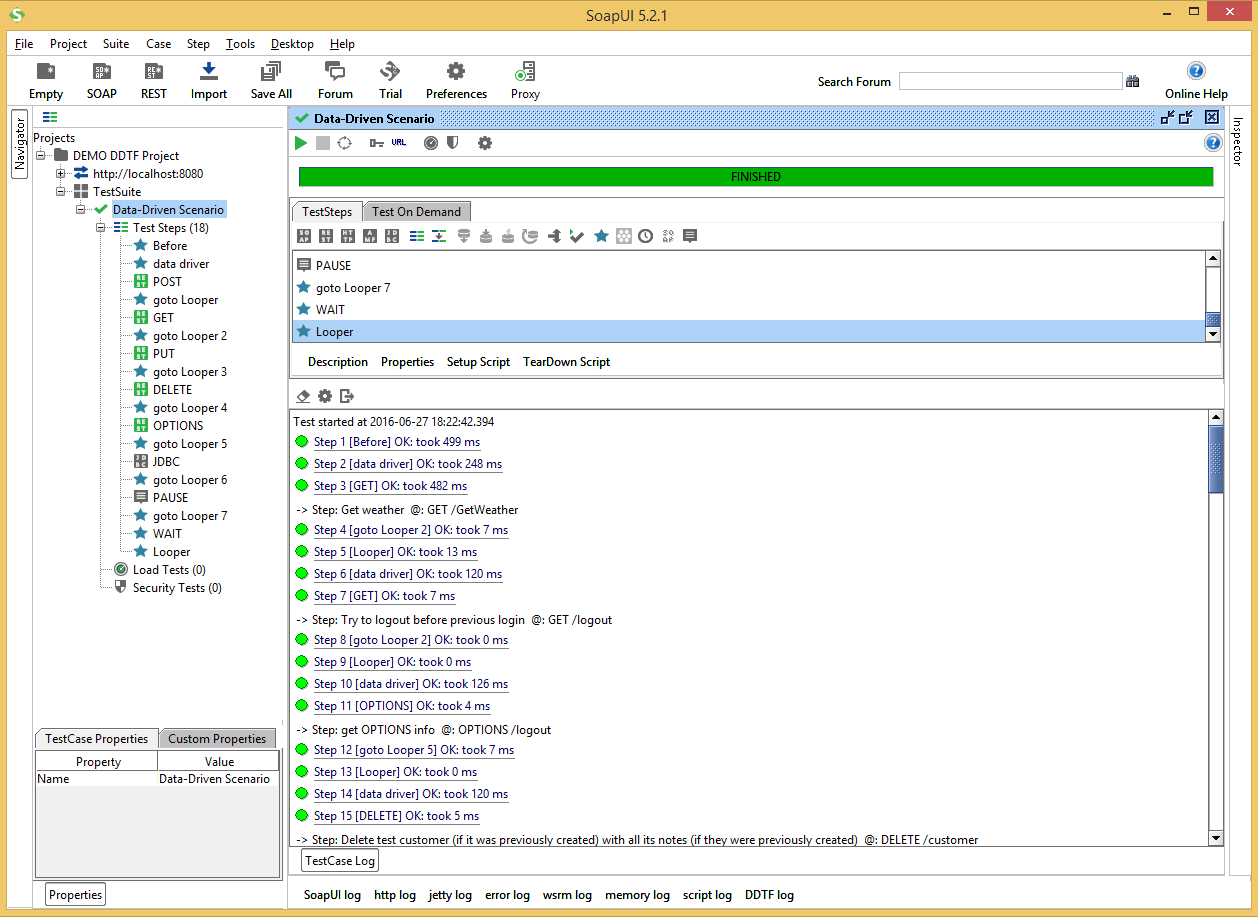


Figure 5 Example of TestSuite Log with test step descriptions

## Resolving Properties in Test Data

DDT Framework Library has ability to resolve references to properties from SoapUI project or configuration file.

By convention, the property references used to easy switch test cases between different environments.

Test data may contain references to properties from SoapUI project or configuration file. The definition of property provided in following form.

%{propertyName}

Where “propertyName” is the references to some property with the same name in configuration file or SoapUI project.

1. Please see the Appendix A.1 for example of property references in Test Data file.

All properties in test data resolved at step of data reading. All property references in test data will be resolved if their values are known.

## Resolving Variables in Test Data

DDT Framework Library has ability to resolve references to variables stored in current data file or in external map file. The variable references used to store and update all commonly used values in test data in one map file.

Test data may contain variables from map file. The definition of a variable provided in following form.

${variableName}

Where “*variableName*” is the references to some variable in map file.

1. Please see the Appendix A.2 for example of variable references in Test Data file.

Full path to the map file should be defined in the SoapUI project properties or configuration file. Map file should contain variable names and values.

1. Please see the Appendix A.2 for example of variable mapping file.

All variables in test data resolved at step of data reading immediately after properties. All variable references in test data will be resolved if their values are known. Variable replacement works in a recursive way.

Table 4 Example of recursive substitution for variable references

| Recursive variable references | |
| --- | --- |
| Name | Value |
| error\_for\_pcode | {"errors": ["03002"],"log-message":"HYB: Product with code '${pcode}' not found!"} |
| pcode | 0000316P |

Thus, in example provided in Table 4, if we have reference for following variable.

${error\_for\_pcode}

Then it should return the following value.

{“errors”:[“03002”],”log-message”:”HYB: Product with code ‘0000316P’ not found!”}

Recursive substitution in variables names is also available.

Table 5 Example of recursive substitution in variable names

| Recursive in variable names | |
| --- | --- |
| Name | Value |
| error\_for\_0000316P | {"errors": ["03002"],"log-message":"HYB: Product with code '0000316P' not found!"} |
| pcode | 0000316P |

Thus, in example provided in Table 5, if we have reference for following variable.

${error\_for\_${pcode}}

Then it should return the following value.

{“errors”:[“03002”],”log-message”:”HYB: Product with code ‘0000316P’ not found!”}

1. It is possible use variable references as workaround for the Excel limitation on total number of characters that a cell can contain. For example, In case of large expected response it could be separated in several variables: *“{"product":${bigJSON1},"store":${bigJSON2}}”*, where *bigJSON1* and *bigJSON2* are variable references for two parts of one response.

## Resolving Objects in Test Data

DDT Framework Library has ability to collect JSON objects and resolve references to them in current data file.

Test data may contain objects collected programmatically via EvalScript or FIND method. The definition of an object provided in following form

@{objectAlias}

Where “*objectAlias*” is the references to some collected object

## Evaluation Scripts

DDT Framework Library contains the EvalUtils class with methods available for test data processing as evaluation scripts. These methods allow extract data from responses and save as variables.

### Extract Variables

The following method in the EvalUtils class allow extracting data from string with regex and saving in variables map with specified name.

EvalUtils.extractVaribles(string, name, regex)

It is possible to use extracted values in the next test steps as variable.

${name}

1. Please see the Appendix A.2 for example of variable mapping file.
2. Make sure that DDT Framework library is available in Java class path.

### Set Variables

All variables values are stored in static field with map structure in EvalUtils class. It is possible to set new variables with following script.

EvalUtils.setVariable(“variableName”, “variableValue”)

It is possible to use defined value in the next test steps as variable.

${variableName}

1. Make sure that DDT Framework library is added to Java classpath.

### Get Variables

All variables values are stored in static field with map structure in EvalUtils class. It is possible to get variables with following script.

EvalUtils.getVariable(“variableName”)

It is possible to use defined value in the next test steps as variable.

${variableName}

1. Make sure that DDT Framework library is available in Java class path.

### Download and Verify PKPASS File

There are several methods in the EvalUtils class to download and verify the PKPASS file in response.

1. It is possible to use the following method to verify the content length of the PKPASS file. It returns the value of Content-Length response header.

EvalUtils.getContentLength(messageExchange)

1. It is possible to validate content of the ‘pass.json’ file with following method. It returns content of file from PKPASS as JSON.

EvalUtils.getPassJson(messageExchange)

1. It is possible to obtain value of Content-Disposition response header with following method. The Content-Disposition response header usually contains the PKPASS file name.

EvalUtils.getContentDisposition(messageExchange)

1. It is possible to get the PKPASS file name directly with following method.

EvalUtils.getPKPASSFileName(messageExchange)

1. Finally, it is possible to download the PKPASS file itself with following method. It store the PKPASS file with name defined in the Content-Disposition response header in the “<SoapUI Project File Folder>/downloads/<Data File Name>/” folder.

EvalUtils.downloadPKPASS(messageExchange, context)

1. Make sure that DDT Framework library is available in Java class path.

# Soap UI Project

SoapUI is the main tool for adding, configuration and execution of test cases.

Soap UI project contains set of test cases for regression testing. It configures path to test data files, endpoints, REST Requests, JDBC Requests, Groovy Scripts and Assertion Scripts with DDT Framework Library methods calls, and Manual Steps if it is required.

The latest version of (template or demo) SoapUI project is available on GIT repository <https://gitbud.epam.com/aleksei_galkin/ddt-framework.git> in following folder: ‘\docs\examples\soap-ui-tests’ (‘\docs\examples\demo’ for DEMO).

## Soapui file encoding configuration

To be sure, that all requests containing specific symbols (umlauts for example) are displayed correctly, it is required to set UTF-8 file encoding format for SoapUI application.

1. Open the “SOAPUI\_HOME/bin” directory;
2. Open the “SoapUI-5.x.x.vmoptions” file in text editor;
3. Add following line to the “SoapUI-5.x.x.vmoptions”; and

-Dfile.encoding=UTF-8

1. Save changes.

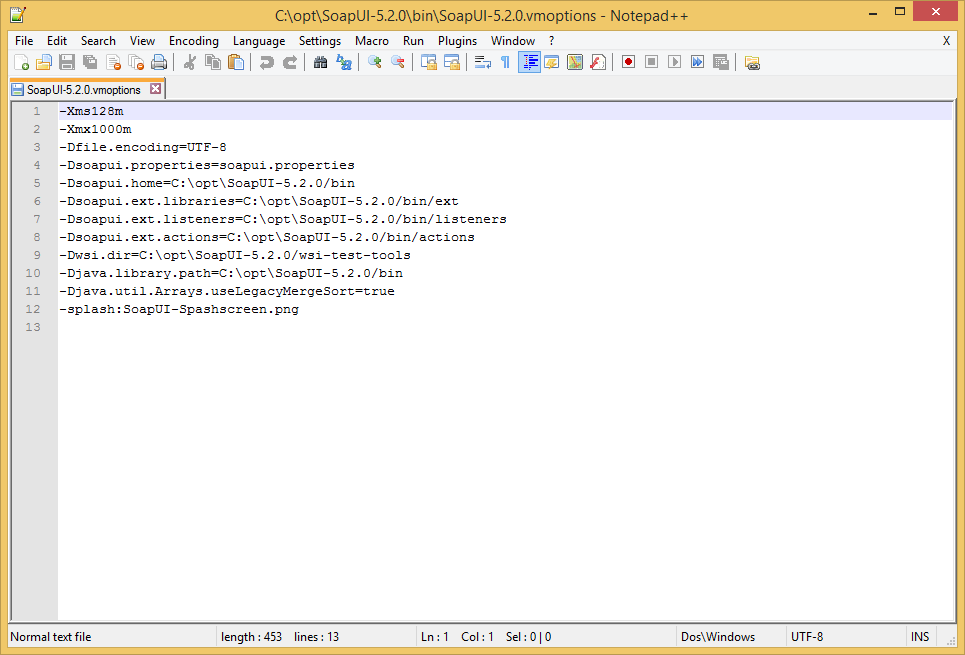


Figure 6 Example of “SoapUI-5.x.x.vmoptions” file

## Project Custom Properties

The SoapUI project can contain values in the Custom Properties tab for following properties.

Table 6 List of Custom Project properties in SoapUI project

| Custom Properties for SoapUI project | | |
| --- | --- | --- |
| Property name | Description | Example |
| endPoint | Services layer server | http://localhost:8080 |
| dataPath | Path to directory with data files | C:\ddt-framework\SoapUI\data |
| dataMappingFile | Path to file with the variables values used in data files | C:\ddt-framework\SoapUI\data\dataMapping.xls |

They can be placed in configuration file (testProps.properties). In such situation their values on Custom Properties tab will be ignored and current values will be taken from configuration file.

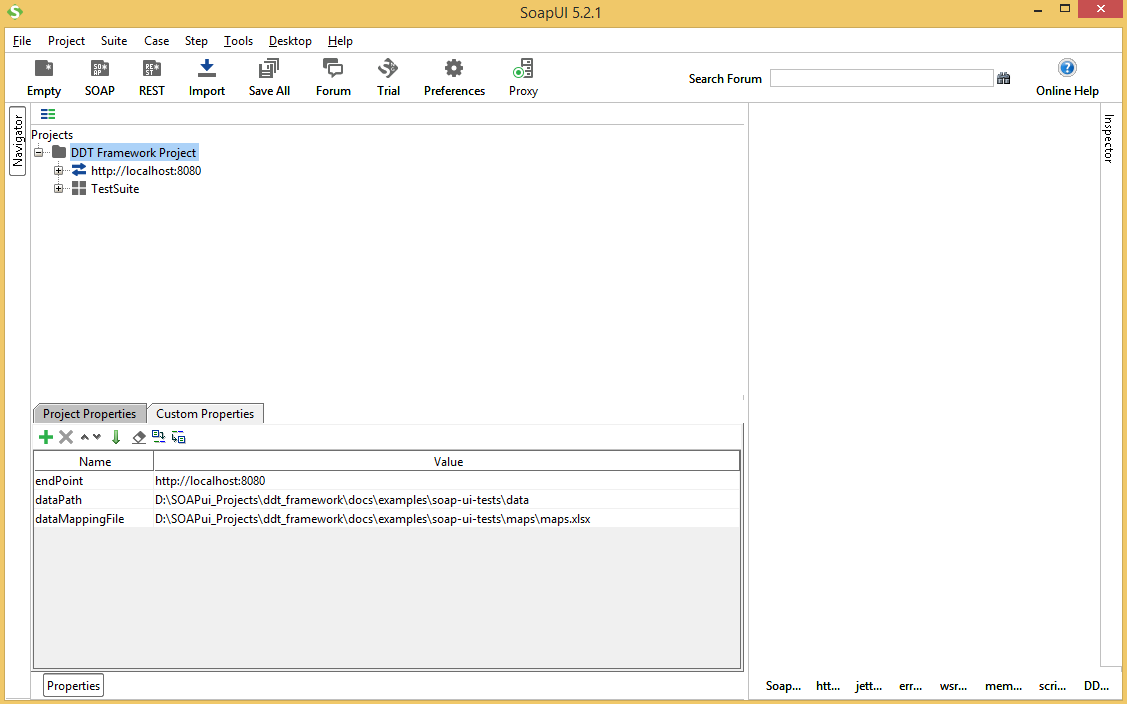


Figure 7 Custom Properties for SoapUI project

## Properties from Configuration File

1. The configuration file for SoapUI project should be located at the same directory as project itself.

Custom Properties of SoapUI project used as default properties for project. It is possible re-define or add new properties to the ‘testProps.properties’ configuration file.

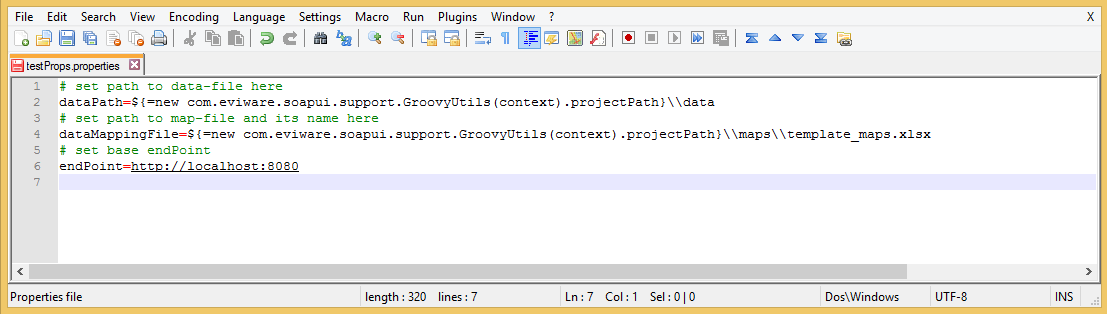


Figure 8 Custom Properties for SoapUI project

## User-Defined Cookie

1. This feature works only with DDT Framework Plugin.
2. The configuration file for SoapUI project should be located at the same directory as project itself.

It is possible to add user-defined cookies to HTTP request with DDT Framework Plugin. If the ‘cookie’ property is defined in Custom Properties of SoapUI project or in the ‘testProps.properties’ configuration file then it value will be added to cookies for HTTP requests.

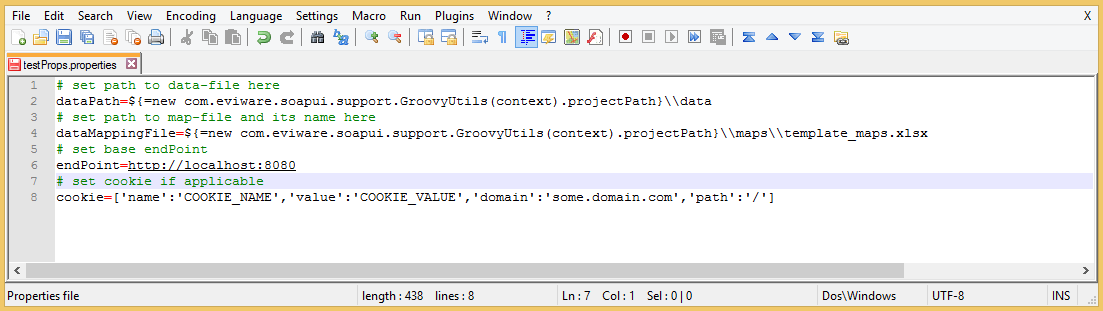


Figure 9 Example of test configuration with user-defined cookie

## Reporting Feature

1. This feature works only with DDT Framework Plugin.

It is possible to collect a report file after execution. If the ‘reportPath’ property is defined in Custom Properties of SoapUI project or in the ‘testProps.properties’ configuration file then after execution there will be created XML file for Suite Level in format suited maven-surefire-plugin and will be put to defined folder. If test case level results is required to be present in the report there should be defined ‘namingPattern’ property in Custom Properties of SoapUI project or in the ‘testProps.properties’ configuration file. There possible to set two group for matching test case name from ‘description’ field of data file.

Examples:

namingPattern=.\*?TC-([\\d+)-\\d](file:///\\d+)-\\d)+.+

namingPattern=.\*?(Negative|Positive|Alternative)-(\\d+)-\\d+.+

## TestSuite Custom Properties

Test automation performed with data-driven testing. Test data (input, expected output, etc.) are stored in external Excel files. The “Data driven scenario” Test Case read and iteratively use test data. Test data file name defined in the TestSuite Custom Properties.

Table 7 TestSuite properties in SoapUI project

| TestSuite Custom Properties | | |
| --- | --- | --- |
| Property name | Description | Example |
| datafile | Name of file with test data for the Test Suite | data\_file.xlsx |

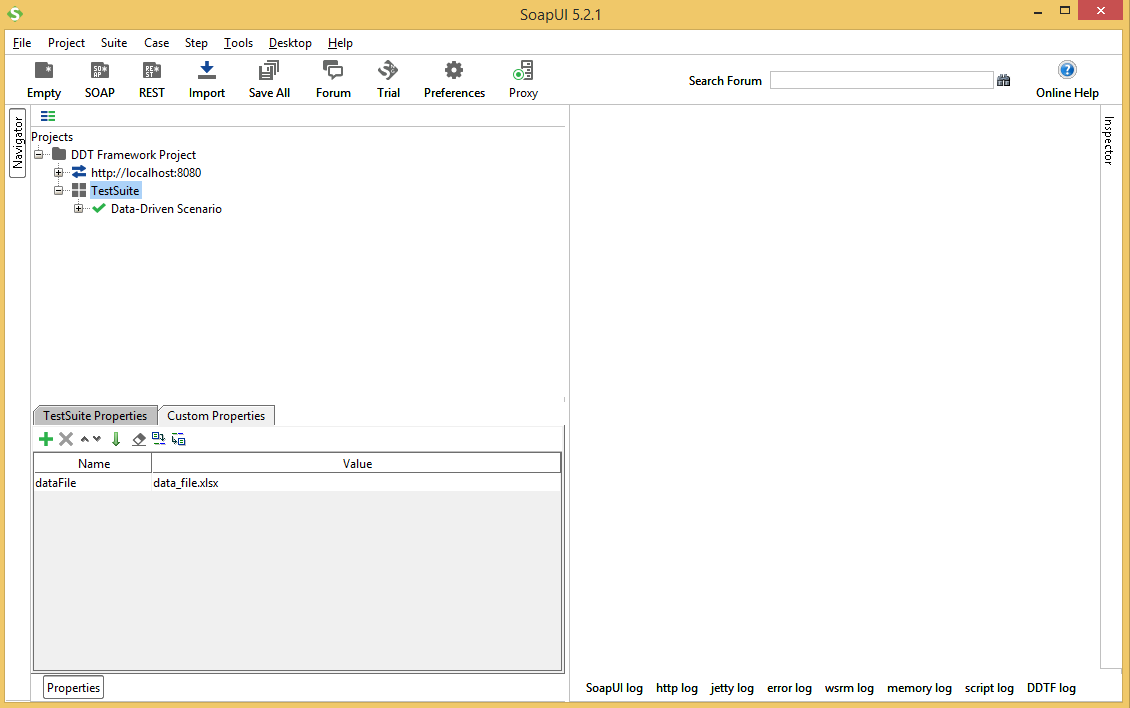


Figure 10 Custom Properties for TestSuite in the SoapUI project

It’s possible to clone TestSuite as much times as you need to have a possibility to execute several data files. To do that please highlight TestSuite for cloning and make right mouse click to open context menu, then choose ‘Clone Test Suite’ menu item. Dialog ‘Clone TestSuite’ will be opened. Enter name for new TestSuite and press ‘Clone’ button. The new TestSuite will appear in SoapUI tree-view. Correct value of custom TestSuite property ‘dataFile’ to point to another data file.

## TestCase Custom Properties

All data used during test case execution are stored in TestCase Custom Properties. The DDT Framework Library scripts set up these properties during test case execution.

Table 8 TestCase properties in SoapUI project

| TestCase Custom Properties | | |
| --- | --- | --- |
| Property name | Description | Example |
| desc | Test step description for the current test step | Step: Get weather @: GET /GetWeather |
| currentRow | Current row number from data file | 6 |
| countRows | Count rows number from data file | 46 |
| resp | Expected response for the current test step | <string xmlns="http://www.webserviceX.NET">Data Not Found</string> |
| evalResponse | Evaluation script for expected response |  |
| contentType | Expected content-type of response for current test step | text/xml; charset=utf-8 |
| status | Expected HTTP status of response for current test step | HTTP/1.1 200 OK |
| expectedAssertionResults | Map for expected assertion results | [“response”: true] |
| endPoint | Endpoint for REST requests | http://localhost:8080 |
| resPath | Resource path for REST request | /GetWeather |
| currentResponse | Actual response for current test step | <string xmlns="http://www.webserviceX.NET">Data Not Found</string> |
| currentStatus | Actual HTTP status of response for current test step | HTTP/1.1 200 OK |
| xlsFile | Full path to test data file used in test case | D:\SOAPui\_Projects\ddt\_framework\docs\examples\soap-ui-tests\data\demo\_data.xlsx |
| dataMappingFile | Full path to the variables map file | D:\SOAPui\_Projects\ddt\_framework\docs\examples\soap-ui-tests\maps\demo\_maps.xlsx |

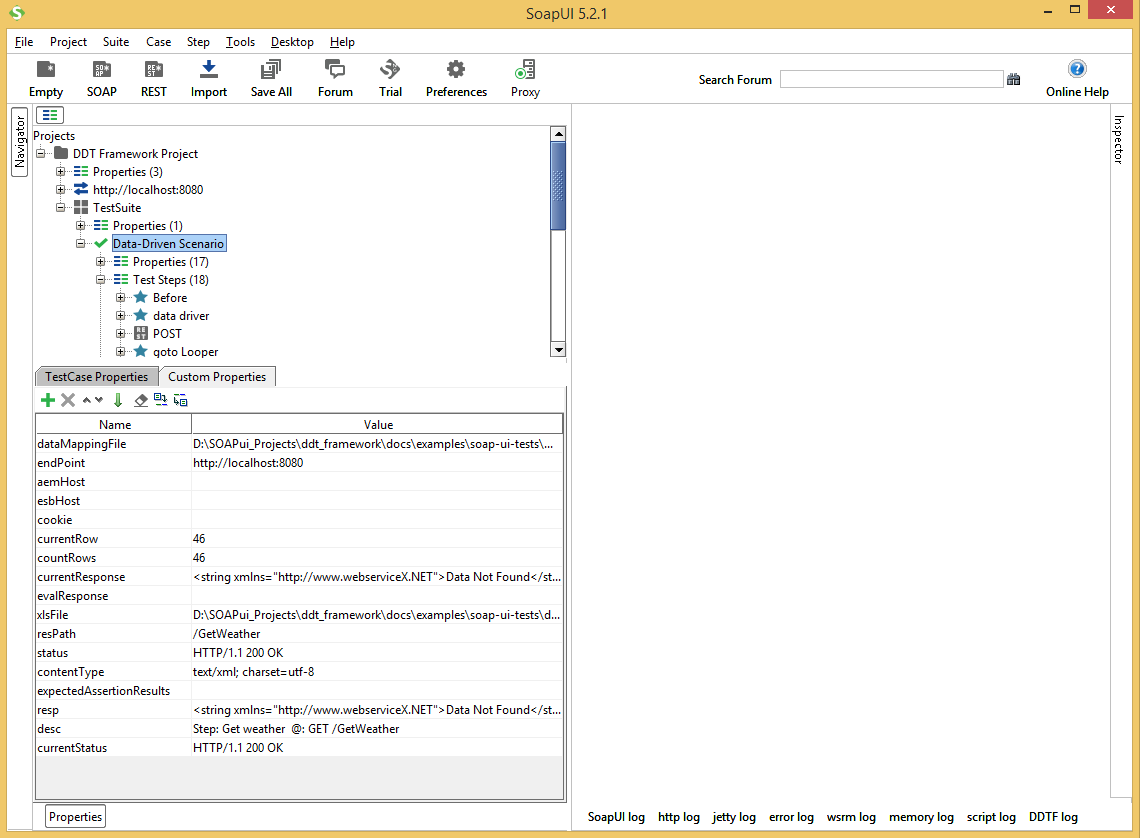


Figure 11 Custom Properties for TestCase in the SoapUI project

## Data Driven Test Scenario

1. Test case «Data driven scenario» should have following options (ctrl + shift + o):
2. “Maintain HTTP session” – set to ‘ON’; and
3. “Abort test if an error occurs” – set to ‘OFF’.

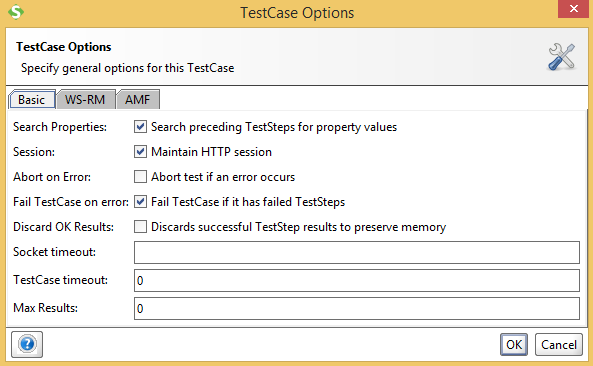


Figure 12 ‘Data driven scenario’ test case options

### Test Case Structure

Test case consists of Groovy scripts to manage test step workflow, REST Requests test steps and manual step to perform custom action if required.

Table 9 TestStep descriptions

| TestStep List | | |
| --- | --- | --- |
| TestStep name | Type | Description |
| Before | Groovy script | Run the ‘prepareTestCase’ method from DDT Framework Library |
| data driver | Groovy script | Run the ‘dataDriver’ method from DDT Framework Library |
| POST | REST request | 1. Send POST request. 2. Run the script assertion methods from DDT Framework s Library to assert response, HTTP code and content-type. |
| goto Looper | Groovy script | Go to Looper step |
| GET | REST request | 1. Send GET request. 2. Run the script assertion methods from DDT Framework Library to assert response, HTTP code and content-type. |
| goto Looper 2 | Groovy script | Go to Looper step |
| DELETE | REST request | 1. Send DELETE request. 2. Run the script assertions methods from DDT Framework Library to assert response, HTTP code and content-type. |
| goto Looper 3 | Groovy script | Go to Looper step |
| PUT | REST request | 1. Send PUT request. 2. Run the script assertion methods from DDT Framework Library to assert response, HTTP code and content-type. |
| goto Looper 4 | Groovy script | Go to Looper step |
| PAUSE | Manual step | 1. Perform the Manual step. 2. Wait actions from tester (e.g. set filters or modify data). |
| goto Looper 5 | Groovy script | Go to Looper step |
| OPTIONS | REST request | 1. Send OPTIONS request. 2. Run the script assertion methods from DDT Framework Library to assert response, HTTP code and content-type. |
| goto Looper 6 | Groovy script | Go to Looper step |
| JDBC | JDBC request | 1. Send JDBC request. 2. Run the script assertion methods from DDT Framework to assert response, HTTP code and content-type. |
| goto Looper 6 | Groovy script | Go to Looper step |
| WAIT | Groovy script | 1. Sleep for time specified by user. |
| Looper | Groovy script | Go to the ‘data driver’ step until reach the end of test data file |

1. All TestSteps names are case sensitive.

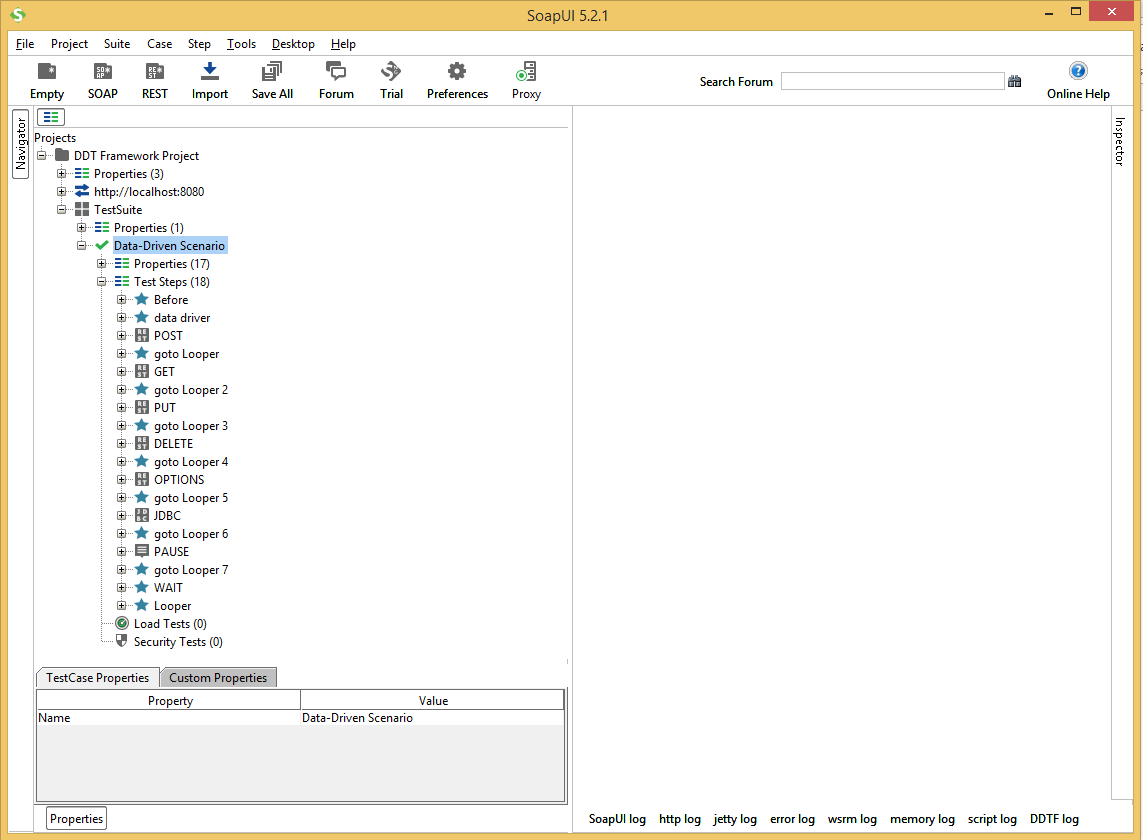


Figure 13 TestCase structure in the SoapUI project

### ‘Before’ Test Step

The ‘Before’ test step prepares test case for execution. It loads properties from SoapUI project custom properties or from properties configuration file and sets following properties to TestCase Custom Properties:

1. ‘endPoint’ – default endpoint for REST Requests;
2. ‘dataMappingFile’ – map file with values to resolve variables in test data;
3. ‘currentRow’ – set to ‘0’ and it indicates row from test data file which is processed at current iteration;
4. ‘countRows’ – set to ‘0’ and it indicates row count from test data file;
5. ‘currentResponse’ – empty string and it used to store response from REST Request to assert it with expected value;
6. ‘evalResponse’ – empty string and it used to store script for response evaluation to process it before response assertion;
7. ‘xlsFile’ – full path to data file used in current test case. It defined by ‘dataPath’ project property and ‘datafile’ test suite property: ‘xlsFile’ = ’dataPath’ + ’datafile’.
8. ‘cookie’ – value from ‘cookie’ project/configuration file property or empty
9. ‘reportPath’ – path to put reports files if ‘reportPath’ project/configuration file property is defined or empty

The ‘Before’ test step uses the ‘prepareTestCase’ method in TestCaseUtils class of DDT Framework Library.

TestCaseUtils.prepareTestCase(context)

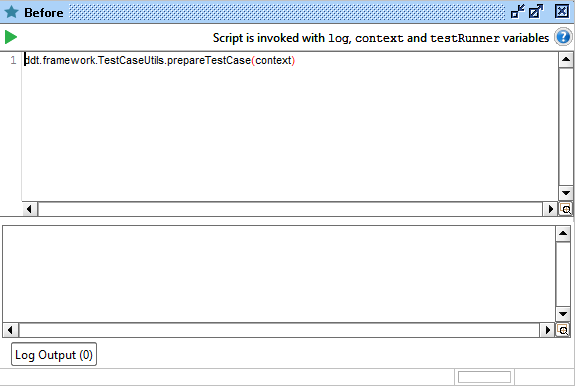


Figure 14 ‘Before’ TestStep

### ‘data driver’ Test Step

The ‘data driver’ test step iterates over test data file, sets test step parameters, resolves variables in test data and navigates to appropriate test step for execution. At each iteration, it gets data from an appropriate row until reaches the end of file or an empty row. If the row with data not commented then it may be processed:

1. Resolve all properties in test data;
2. Resolve all variables/objects(products) in test data;
3. Get test step name from test data;
4. If test step is equal to REST method, then
   1. Set endpoint from test data to REST method;
   2. Set headers to request if defined;
   3. Set resource path;
   4. Set media type;
   5. Process and set request content from data;
   6. Process and set request parameters;
   7. Set/Enable assertions
5. If test step is equal to JDBC method, then
   1. Set driver;
   2. Set connection string
   3. Process and set SQL query
   4. Set/Enable assertions
6. If test step is equal to “PAUSE” manual step, then set step description defined in resource path field of test data;
7. If test step is equal to FIND method, then
   1. Process and set appropriate test step
   2. Pass control to that test step
8. Set the test step description to the test case properties for logging; and
9. Give execution control to the appropriate test step.

The ‘data driver’ test step uses the ‘dataDriver’ method in TestCaseUtils class of DDT Framework Library.

TestCaseUtils.dataDriver(context, testRunner)

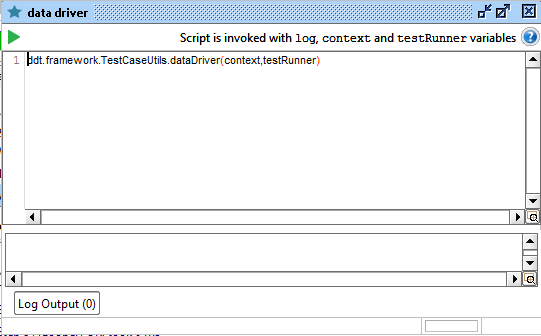


Figure 15 ‘data driver’ TestStep

### REST Request Test Steps

The REST Request test steps perform requests to services according to the test data. Each request assert its response, cookies, status and content type according to the test data. Each method compares appropriate part of response with expected values.

It is possible to assert HTTP Status, Content-Type, response body and response cookies in separate script assertions using appropriate methods in the DdtfAssertions class of DDT Framework Library: ‘assertHTTPStatus’, ‘assertContentType’ and ‘assertResponse’.

It is possible to save response body as an JSON object using appropriate method in the DdtfAssertions class of DDT Framework Library - ‘Save Object’

DdtfAssertions.assertHTTPStatus(messageExchange, context)

DdtfAssertions.assertContentType(messageExchange, context)

DdtfAssertions.assertResponse(messageExchange, context)

DdtfAssertions.saveObject(messageExchange, context)

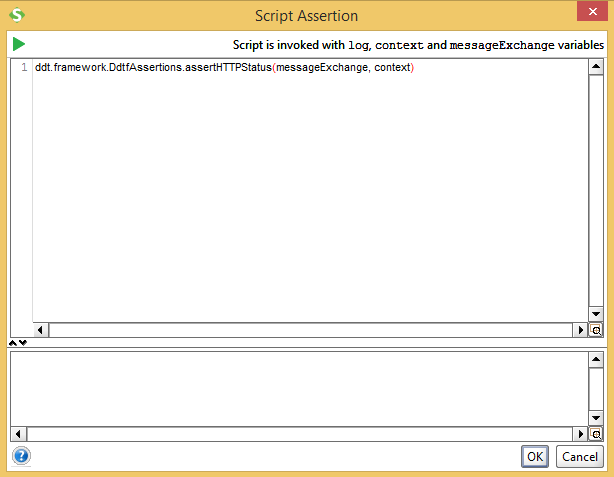


Figure 16 HTTP Status Assertion

### JDBC Request Test Steps

The JDBC Request test step perform request to database according to the test data. It is possible to assert JDBC response according to the test data. JDBC Request test step compares response with expected values.

It is possible to set database connection string with endpoint, resource path and parameters fields in test data file and set the SQL query with request body script.

It is possible to assert response body in separate script assertions using following method in the DdtfAssertions class of DDT Framework Library: ‘assertResponse’.

DdtfAssertions.assertResponse(messageExchange, context)

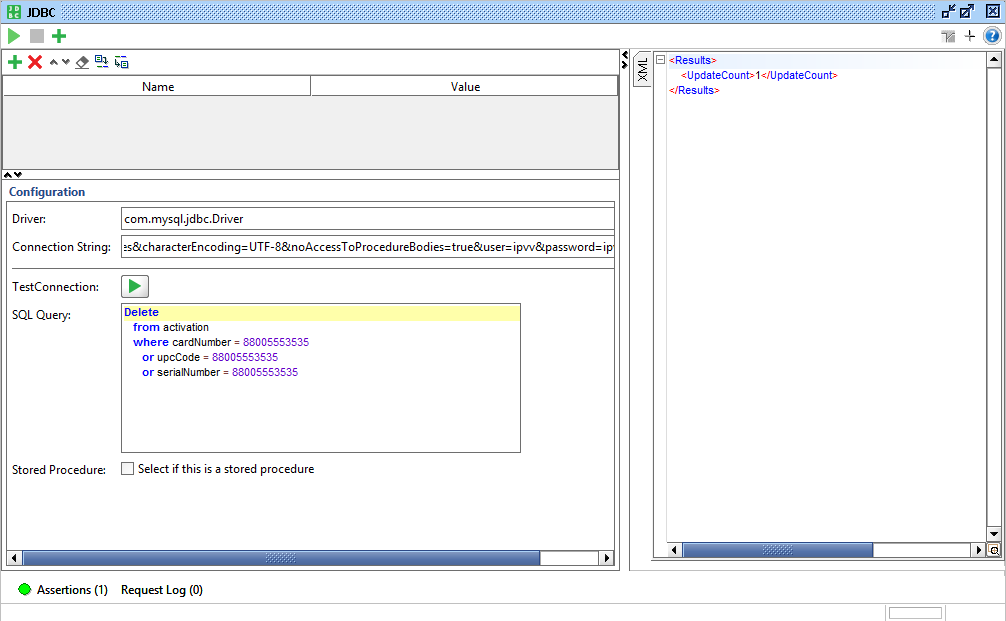


Figure 17 JDBC Request Test Step

1. Please see the Appendix A.4 for example of JDBC Request in data file.

### ‘PAUSE’ Test Step

If test case require perform manual actions from tester like set filters or modify data then ‘PAUSE’ test step will stop test execution until tester manually run execution again.

The ‘PAUSE’ test step contains description of manual steps in the expected result field according to the test data.

### ‘WAIT’ Test Step

If test case require freeze test execution for specified amount of time then ‘WAIT’ test step will stop test execution for required time and continue it again.

### ‘goto Looper’ Test Steps

The ‘goto Looper’ test steps navigate test execution workflow to the ‘Looper’ test step after each test step execution.

### ‘Looper’ Test Steps

The ‘Looper’ test step iterates on test data file and give execution control to ‘data driver’ test step until the last data row.

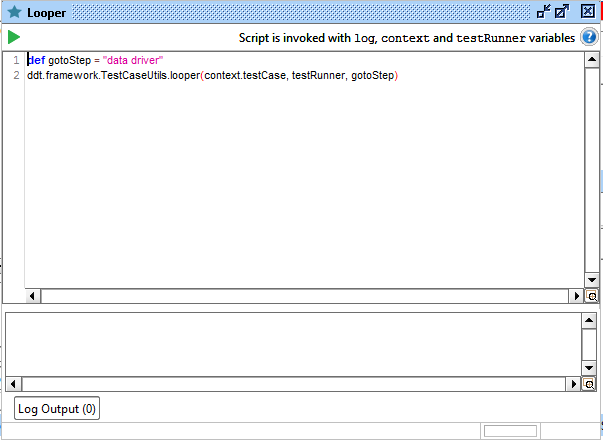


Figure 18 Looper Test Step

### ‘After’ Test Step

The ‘After’ test step uses the ‘finishTestCase’ method in TestCaseUtils class of DDT Framework Library.

If ‘reportPath’ property is set then report of execution will be created

TestCaseUtils.finishTestCase(context, testRunner)

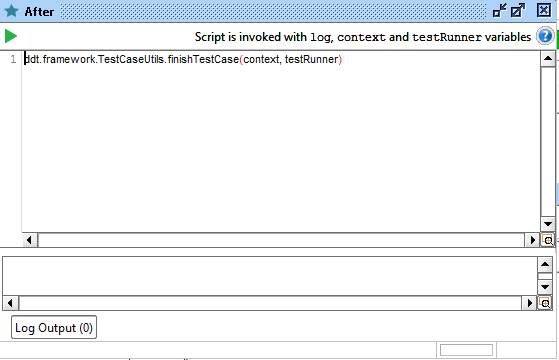


Figure 19 ‘After’ TestStep

# Test Execution and Result Validation

Test environment should configured according to the instructions provided in the “Setup test environment.docx” document.

## Start Test Suite Execution

It is possible to execute any test suite according to the following instruction:

1. Expand project tree in SoapUI;
2. Double click on selected Test Suite to open Test Cases window;

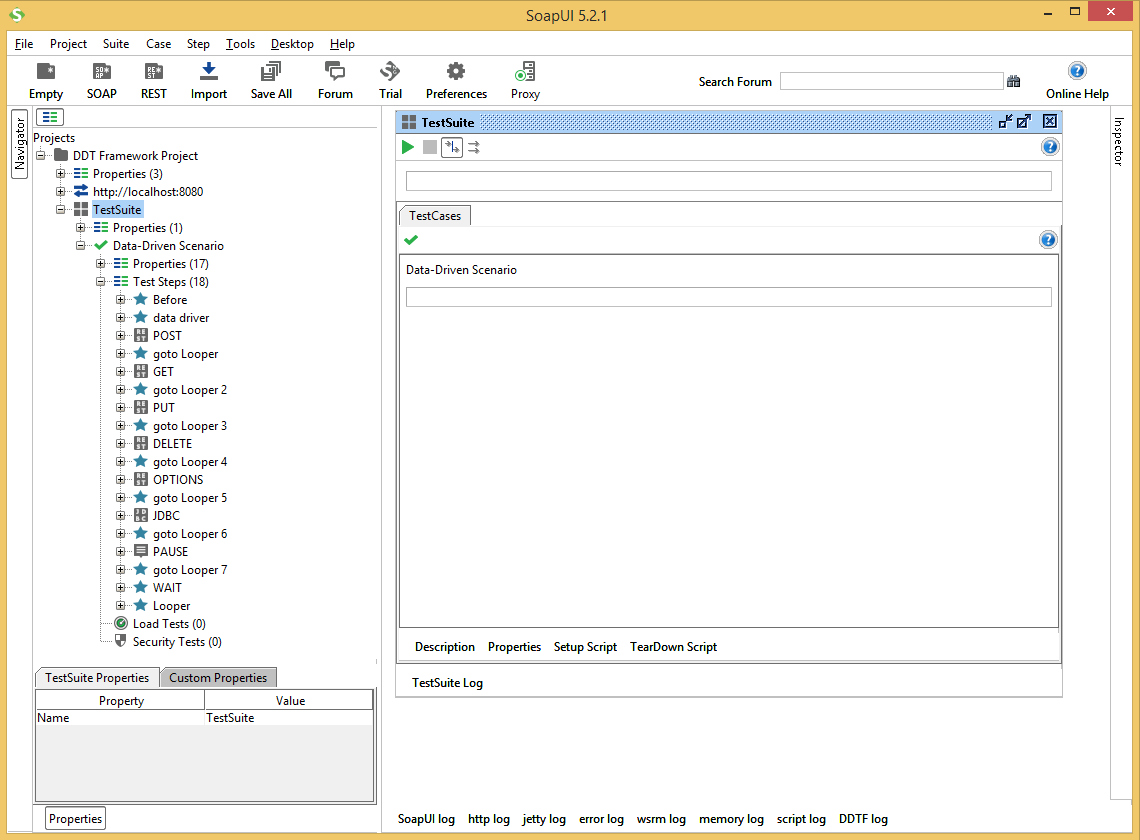


Figure 20 TestSuite window

1. Click on the ‘Runs the selected TestCases’ button (green arrow) to start Test Suite execution.

## View Test Suite Execution Log

All information on status of TestCases execution available in ‘TestSuite Log’ at bottom of the Test Cases window.

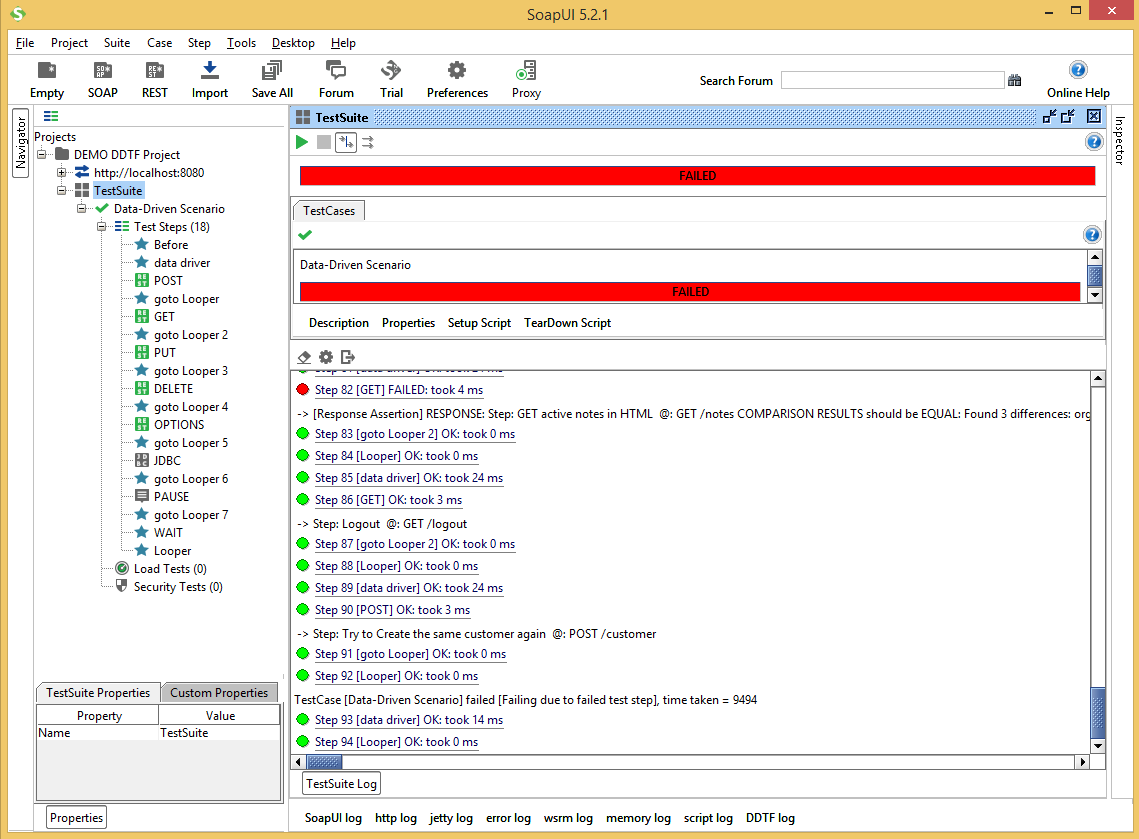


Figure 21 TestSuite Execution Log

If the DDT Framework Plugin successfully added to the SoapUI then the additional information with test descriptions displayed in the TestSuite Log.

1. Please see the Add DDT Framework Plugin to SoapUI section for more details.

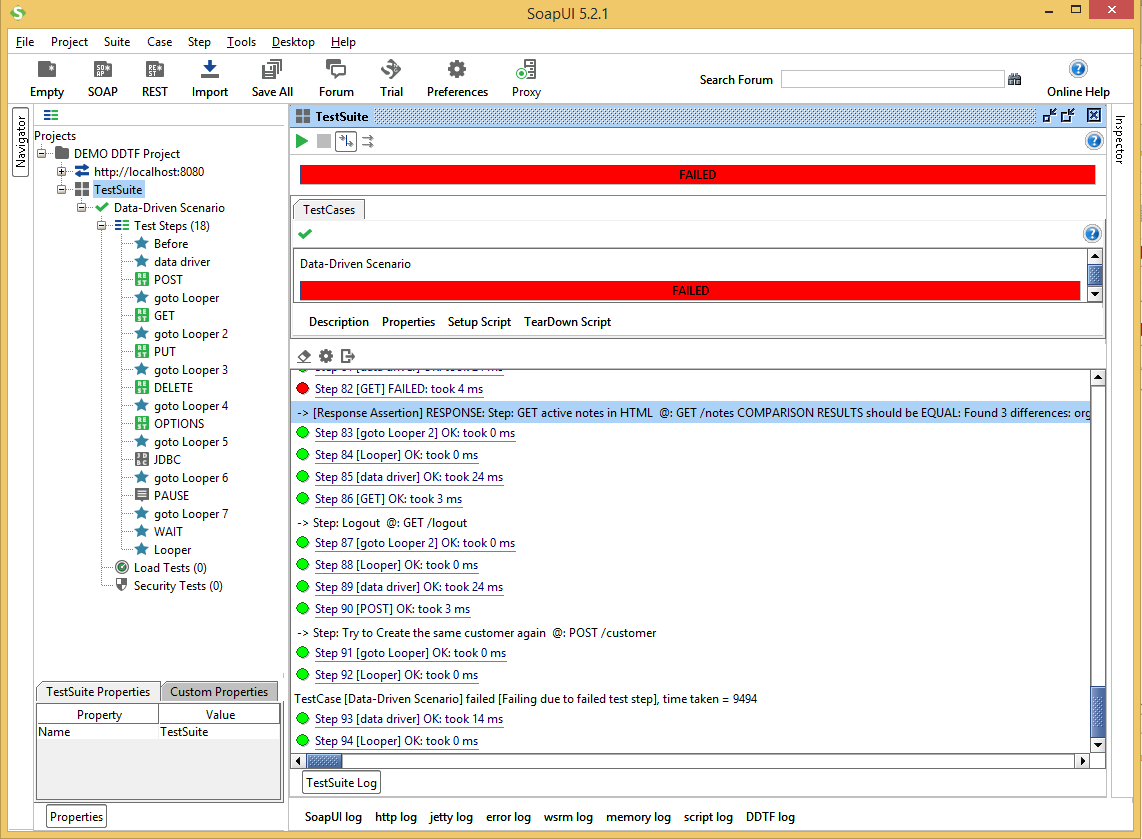


Figure 22 TestSuite Log with test step description.

## View DDT Framework Log

All information on test data, test step parameters and result assertion is logged by the DDT Framework Library and available in ‘DDTF log’ tab at bottom of the SoapUI window.

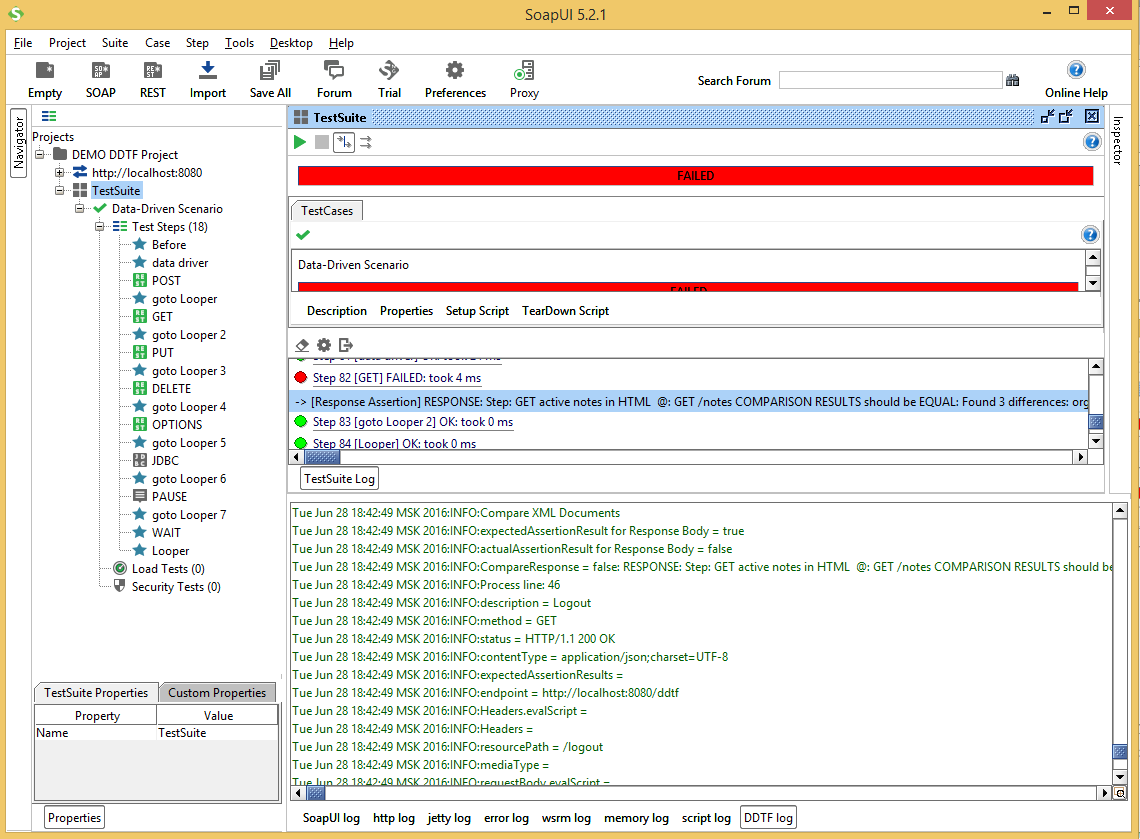


Figure 23 DDT Framework Library Log

1. It is possible to configure the maximum number of rows displayed in ‘DDTF log’ with the “Set Max Rows” menu item of context menu from the ‘DDTF log’ tab.

## Test Step Result Validation

Test suite log contains results for each executed test step. All passed steps marked with green and all failed steps marked with red.

1. It is possible to open each REST request step with double click to view its Request Message, Response Message and Properties.

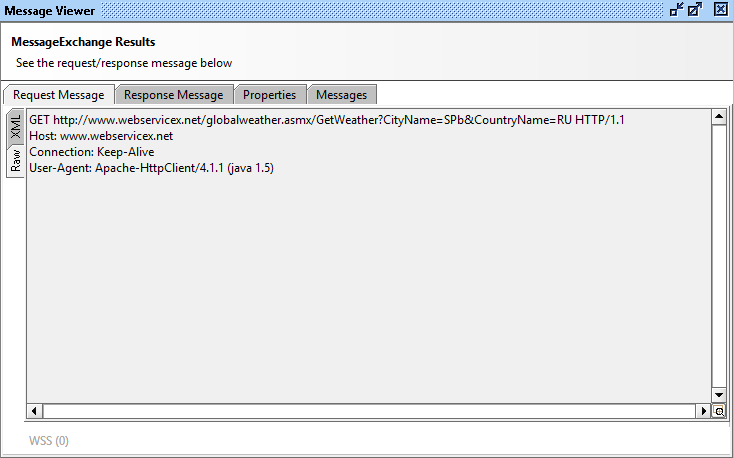


Figure 24 Message Viewer window

1. All failed REST requests contain assertion message.

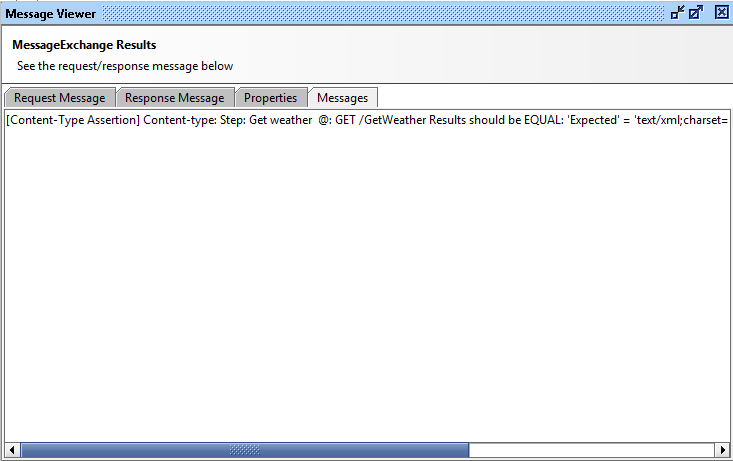


Figure 25 Messages tab with assertion messages

1. It is possible to copy the assertion message from the “Message Viewer” dialog with the “Ctrl+C” keys combination and paste it to the text editor. It is more convenient to analyze of assertion result in test editor. The “Message Viewer” dialog in SoapUI is not support text formatting and indentations for the assertion message.

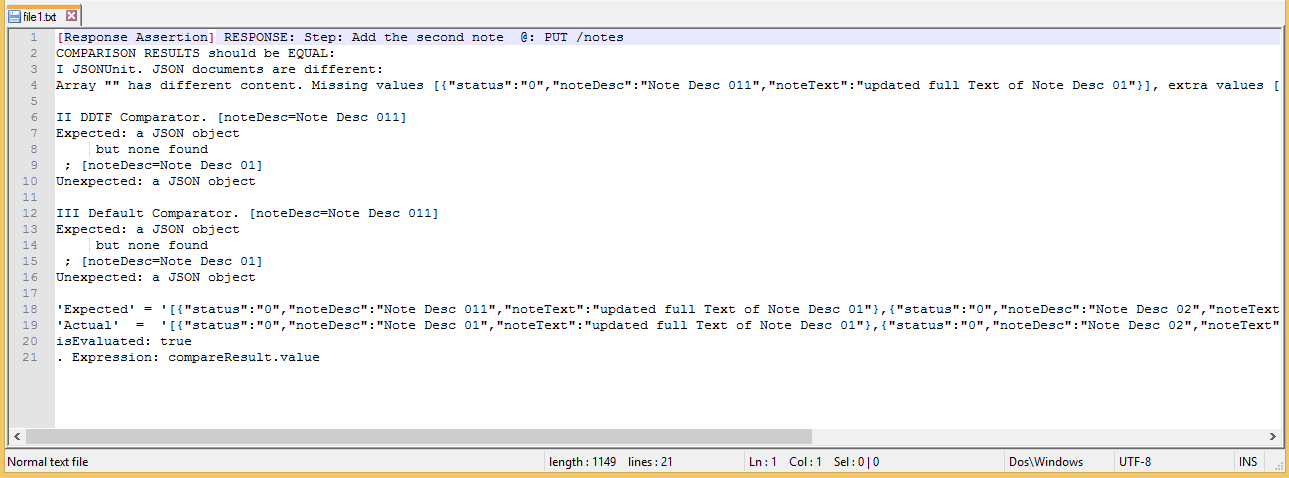


Figure 26 Assertion message in text editor

1. All failed test steps contains step description. Step description start with combination of CELL\_ DESCRIPTION, CELL\_METHOD and CELL\_PATH.

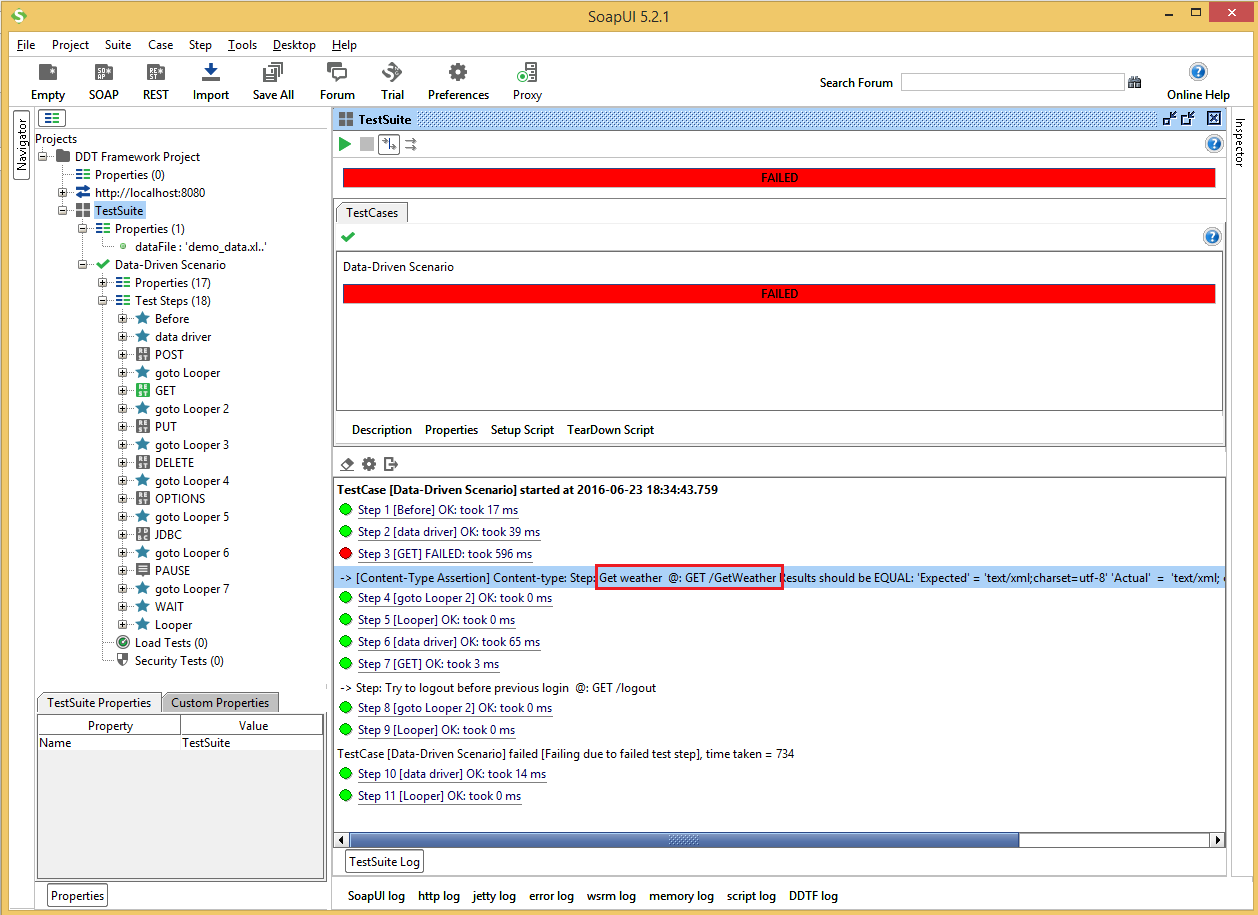


Figure 27 Description of failed test step in TestSuit log

1. All assertion messages start with the same combination of CELL\_ DESCRIPTION, CELL\_METHOD and CELL\_PATH for efficient navigation on test data file.

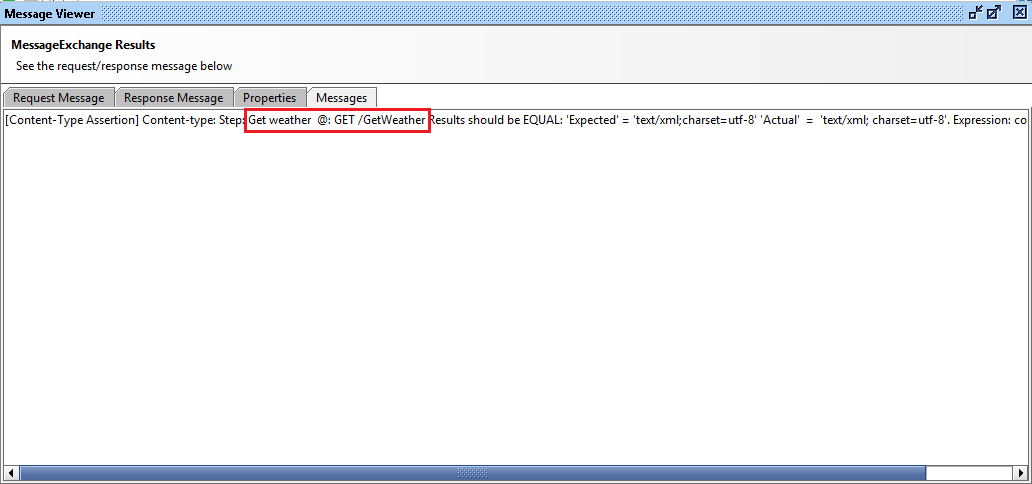


Figure 28 Description of failed test step in assertion message

1. It is possible to verify corresponding test step in test data file using this combination with the appropriate data file.

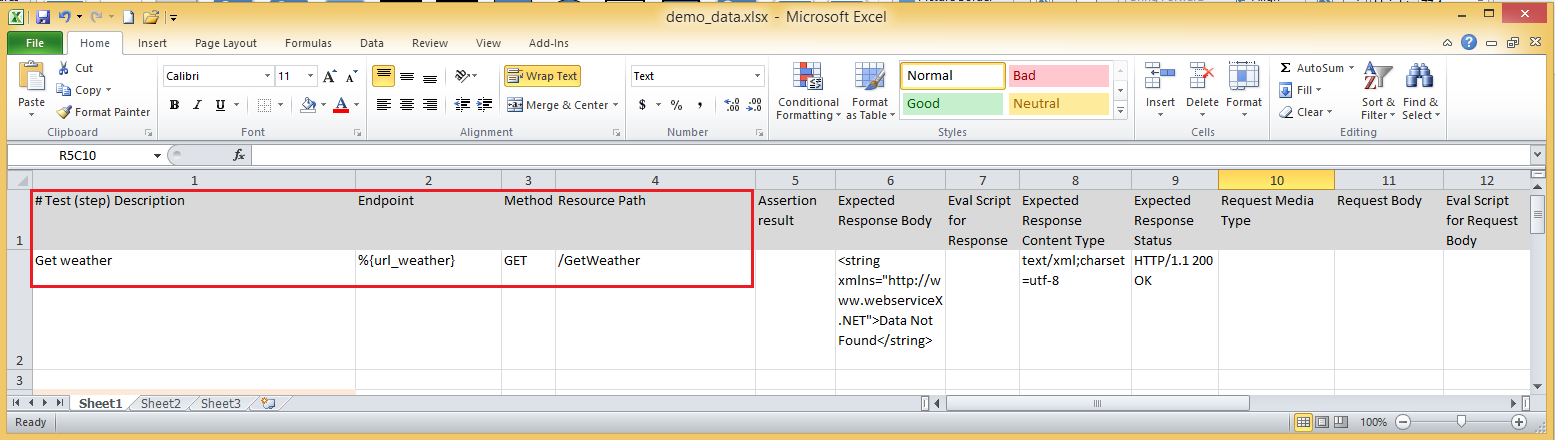


Figure 29 Combination of CELL\_ DESCRIPTION, CELL\_METHOD and CELL\_PATH in Test Data file

1. Each test data file can contain test cases descriptions on the “TestCases” sheet of document.

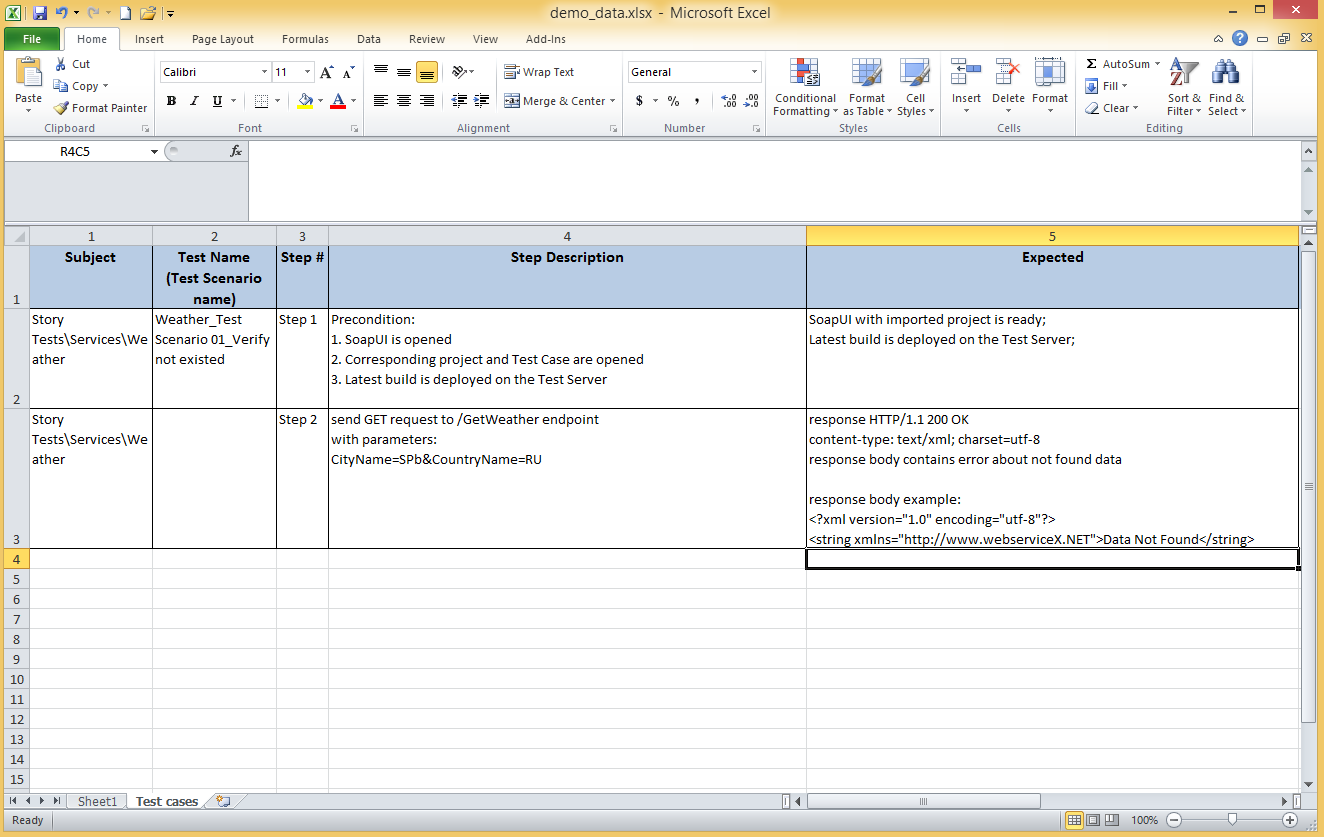


Figure 30 Test scenarios on the “TestCases” sheet of document in Test Data file

1. Test step description contains reference to appropriate test scenario, e.g. “TS 04\_02” means “Test Scenario 04, Step 2”.
2. The “Step Description” and “Expected” fields on the “TestCases” sheets might by updated with short delay. Please use the test data from the first sheet as it contains up to date information.

It is possible to view information on REST request step in different formats for more convenient step validation.

Table 10 Tabs and message formats on Message Viewer window

| Available message formats | |
| --- | --- |
| Message tab | Message format |
| Request Message | * XML * RAW |
| Response Message | * XML * JSON * HTML * RAW |
| Step properties | * SoapUI table |
| Assertion message | * String |

Request and Response Messages in RAW format contain information on HTTP headers.

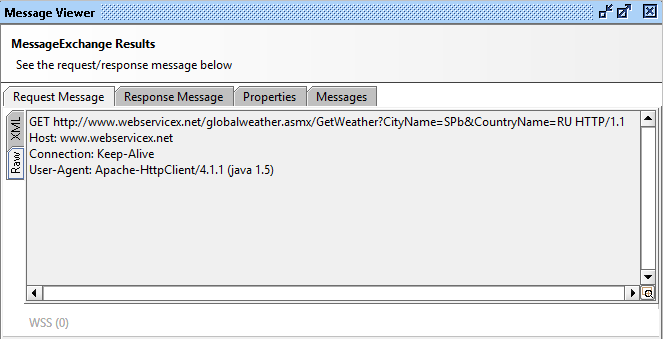


Figure 31 RAW format of Request Message

## Closing SoapUI

There is a common rule about SoapUI – keep your projects ‘closed’ in SoapUI workspace. So, it’s better to close project before closing SoapUI tool. Highlight the project, view context menu by right mouse click and choose ‘Close Project’ menu item. You can also close SoapUI without saving changes. To do that please click on the “File / Exit without saving” menu item.

1. test data file Examples

This chapter contains Test Data files examples.

* Example of Test Data file in Excel 97-2003 Workbook (\*.xls) format.



* Example of Test Data file in Excel Workbook (\*.xlsx) format.



* 1. property references in data file

This chapter contains Test Data files examples with property references.

* Example of the property references in Test Data file in Excel 97-2003 Workbook (\*.xls) format.



* Example of the property references in Test Data file in Excel Workbook (\*.xlsx) format.



* 1. Variable mappinfg file and variable references in data file

This chapter contains examples of variable mapping files and Test Data files with variable references.

* Example of the variable references in Test Data file in Excel 97-2003 Workbook (\*.xls) format.



* Example of the variable references in Test Data file in Excel Workbook (\*.xlsx) format.



* Example of the variable mapping file in Excel 97-2003 Workbook (\*.xls) format.



* Example of the variable mapping file in Excel Workbook (\*.xlsx) format.



* 1. Extract Variable in data file

This chapter contains Test Data file examples with extract variables script.

* Extract variable example in Test Data file in Excel 97-2003 Workbook (\*.xls) format.



* Extract variable example in Test Data file in Excel Workbook (\*.xlsx) format.



* 1. JDBC REquest in data file

This chapter contains Test Data file examples with JDBC request.

* JDBC Request example in Test Data file in Excel 97-2003 Workbook (\*.xls) format.



* JDBC Request example in Test Data file in Excel Workbook (\*.xlsx) format.



1. Troubleshooting

This chapter contains the list of known troubles that may occurs during DDT Framework usage.

* 1. Build Task for DDT Framework Failed

If build process is failed with some error at *“:test”* point try to use the “gradlew.bat jar” command instead to eliminate the library tests execution.

1. If it does not helps please contact to the DDT Framework Team for further support.
   1. Properties from Configuration File is not available in SoapUI Project

Please make sure that configuration file is located at the same folder as the SoapUI project file itself. The DDT Framework Library searches configuration file in directory where SoapUI project file located.

If project file has moved from other directory, please make sure it is re-import successfully into SoapUI. SoapUI has link to the project file in its workspace so it is required to re-import all moved project files to update link in SoapUI workspace or it is possible to switch to the new workspace with imported file.

* 1. Assertion Messages for Failed Steps are too long

The “Message Viewer” dialog in SoapUI is not support text formatting and indentations for the assertion message. However, it is possible to copy the assertion message from the “Message Viewer” dialog with the “Ctrl+C” keys combination and paste it to the text editor. It is more convenient to analyze of assertion result in test editor.

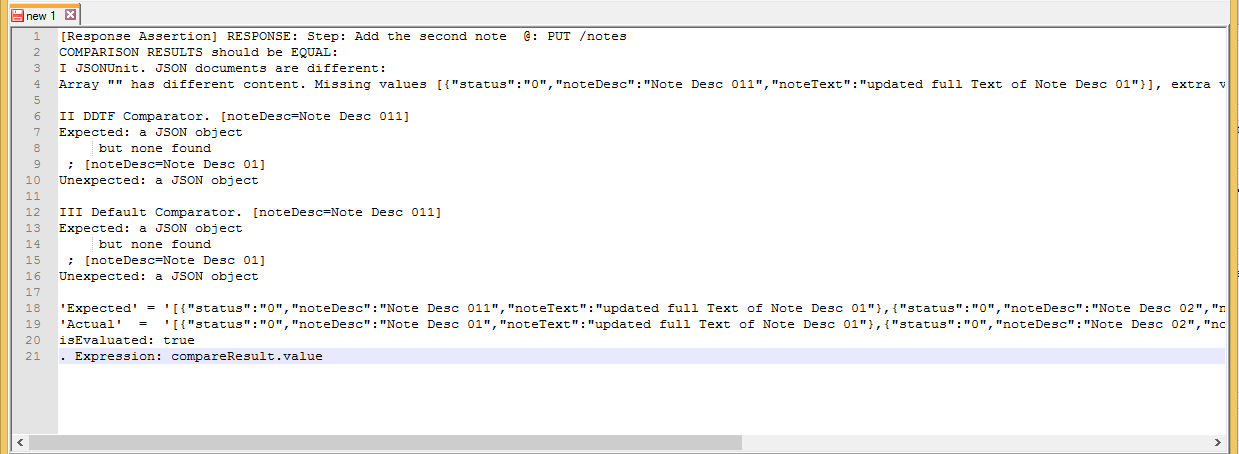


Figure 32 Assertion message in text editor

* 1. Tests are not work or Data-Driver is not works

It is possible to investigate SoapUI log file to figure out what goes wrong. SoapUI creates “soapui.log” and “soapui-errors.log” files in the “SOAPUI\_HOME/bin” directory by default. The “soapui.log” contains detailed logs for data-driver runs with test steps descriptions and parameters. The “soapui-errors.log” contains detailed errors stack traces for SoapUI errors.

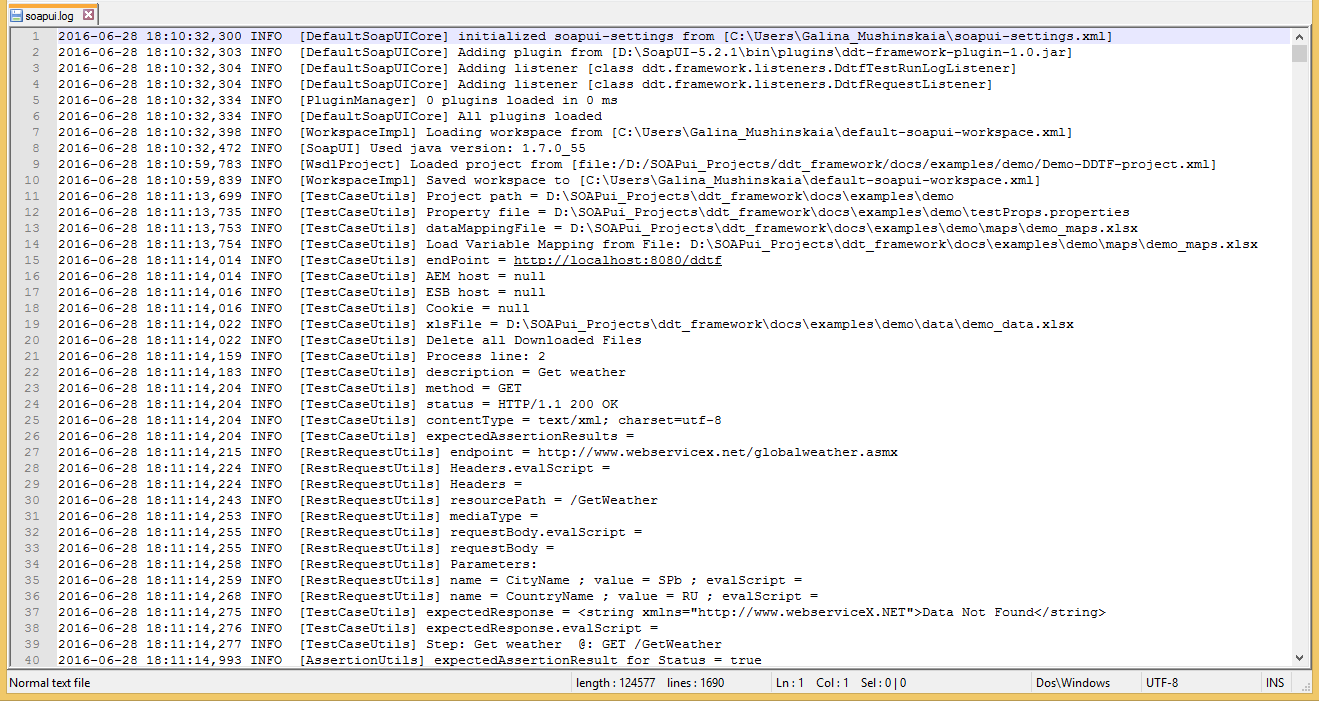


Figure 33 DDT Framework log in soapui.log file

* 1. The “soapui.log” File is Missed

Please make sure that SoapUI have access and rights to create and modify files in its “bin” directory.

* 1. Nothing is helped

Please feel free to contact the DDT Framework Team for further support and help.

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| n.n |  |  | dd-mmm-yyyy |  | dd-mmm-yyyy |
|  |  |  |  |  |  |
|  |  |  |  |  |  |