**How to Achieve CSV Reporting in SoapUI (Open Source)**

<https://dzone.com/articles/how-to-achieve-csv-reporting-in-soapui-open-source-1>

## Introduction

SoapUI is a tool designed for testing of web services. It aims at functional, load, and security testing of SOAP and REST web services. It is a product of SmartBear and comes in the following 2 variants:

* SoapUI OS (Open Source)
* SoapUI Pro (Licensed)

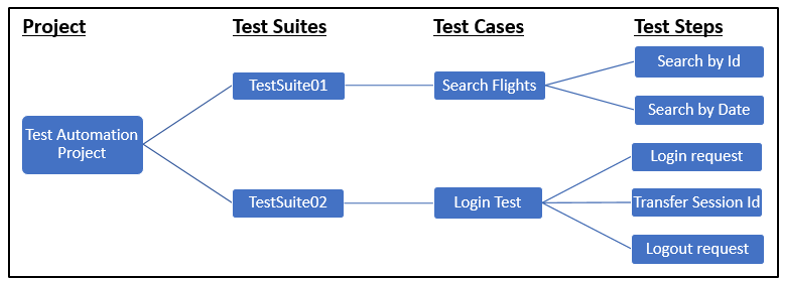
SoapUI OS is a free version available for download and use. The following is the list of a few major features that it lacks in comparison to SoapUI Pro (licensed version):

* data collection
* test debugging
* code-less assertion
* generating external reports based on test run results

This article focuses on generating external reports based on test run results in SoapUI OS.

## Use Case

A project team is working with SoapUI OS for functional testing of web services in their application. The following structure is created for testing the application using SoapUI OS.



As a test deliverable, the client wants the project team to generate a report of test results. Report generated should be in the .csv format and should contain the following information:

* Test Suite, Test Case, Test Step, Step Type, Step Status, Result message, Execution Date
* Request sent and Response received for each test case must be stored locally along with the report

Due to the absence of inbuilt reporting feature in SoapUI OS, an alternate approach should be implemented using the testRunner object.

The testRunner object can access the entire project model and contains all the information about a test case and test results.

In a SoapUI test case, a Groovy Script Step can be added to extract test case results from the testRunner object and generate the report in .csv format.

The following are the steps needed to generate a report:

* In the existing SoapUI project, create a new Test Suite with the name: Library
* Disable the Library test suite to avoid its execution along with the project
* Create a new test case with the name: Reporting\_Utility inside the test suite
* Add Groovy Script Step with the name: GenerateCSVReport in the test case
* Write the following script in GenerateCSVReport:

// Try-catch block to handle exceptions

try {

// 1. Create a "SoapUIResults" folder in the project path

// Retrieve the project root folder

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

// Specify a folder inside project root to store the results

String folderPath = projectPath + "/SoapUIResults";

// Create a File object for the specified path

def resultFolder = new File(folderPath);

// Check for existence of folder and create a folder

if(!resultFolder.exists())

{

resultFolder.mkdirs();

}

/\* ------------------------------------------------------------------------------- \*/

// 2. Create a subfolder (with timestamp) to store the request-response local copy

// Retrieve the latest execution date-time

Date d = new Date();

def executionDate = d.format("dd-MMM-yyyy HH\_mm");

// Specify the subfolder path with name Request-Response\_CurrentTimeStamp

String subfolderPath1 = folderPath+ "/Request-Response\_"+executionDate;

// Create this sub-folder

new File(subfolderPath1).mkdirs();

/\* ------------------------------------------------------------------------------- \*/

// 3. Create another subfolder "CSV Reports" to store the reports file

// Specify the subfolder path with name CSV Reports

String subfolderPath2 = folderPath+ "/CSV Reports";

// Create this sub-folder

new File(subfolderPath2).mkdirs();

/\* ------------------------------------------------------------------------------- \*/

// 4. Create a Report.csv file inside the CSV Reports folder

// Create a File object for Report csv file (with timestamp)

def reportFile = new File(subfolderPath2, "Report\_"+executionDate+".csv");

// Check for existence of report file and create a file

if(!reportFile.exists())

{

reportFile.createNewFile();

// Create required column names in the report file

reportFile.write('"Test Suite","Test Case","Test Step","Step Type","Step Status",'

+'"Result message","Execution Date"');

}

/\* ------------------------------------------------------------------------------- \*/

// 5. Inserting data in the file

// Iterate over all the test steps results

for(stepResult in testRunner.getResults())

{

// Retrieve Test Suite name

def testSuite = testRunner.testCase.testSuite.name;

// Retrieve Test Case name

def testCase = testRunner.testCase.name;

// Retrieve Test Step

def testStep = stepResult.getTestStep();

// Retrieve Test Step name

def testStepName = testStep.name

// Retrieve Test Step type

def type = testStep.config.type

// Retrieve Test Step status

def status = stepResult.getStatus()

// Creating new line in report file

reportFile.append('\n');

// Write all the necessary information in the file

reportFile.append('"' + testSuite + '",');

reportFile.append('"' + testCase + '",');

reportFile.append('"' + testStepName + '",');

reportFile.append('"' + type + '",');

reportFile.append('"' + status + '",');

// Retrieve the test result messages

reportFile.append('"');

for(resMessage in stepResult.getMessages())

{

// Write messages and separate multiple messages by new line

reportFile.append('Message:' + resMessage + '\n');

}

reportFile.append('",');

//Write executionDate in the file

reportFile.append('"' + executionDate + '",');

/\* ------------------------------------------------------------------------------- \*/

// 6. Extract the request and response and save it to external file

// Verify if the test step type is request: SOAP project or restrequest: REST project

if((type=="request").or(type=="restrequest"))

{

// Extract the request from the test step

def extRequest = testStep.properties["Request"].value;

// Create a file in the reports folder and write the request

// Naming convention: request\_TestSuiteName\_TestCaseName\_TestStepName.txt

def requestFile=subfolderPath1+"/request\_"+testSuite+"\_"+testCase+"\_"+testStepName+".txt";

def rqfile = new File(requestFile);

rqfile.write(extRequest, "UTF-8");

// Extract the response from the test step

def extResponse = stepResult.getResponseContent();

// Create a file in the reports folder and write the response

// Naming convention: response\_TestSuiteName\_TestCaseName\_TestStepName.txt

def responseFile=subfolderPath1+"/response\_"+testSuite+"\_"+testCase+"\_"+testStepName+".txt";

def rsfile = new File(responseFile);

rsfile.write(extResponse, "UTF-8");

}

}

}

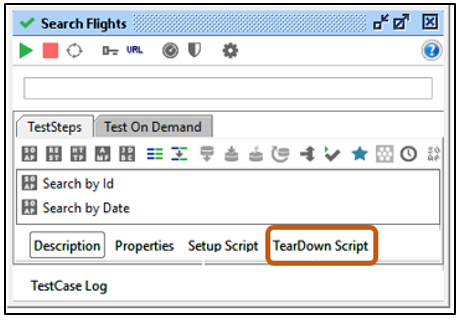
catch(exc)

{

log.error("Exception happened: " + exc.toString());

}

To generate the report, GenerateCSVReport must be executed after each test case execution by following the steps given below:

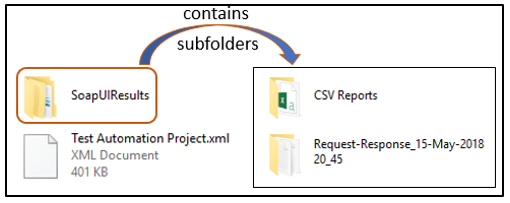
* Open the “TearDown Script” section for each test case from the test case window by clicking on the button as shown below:
* In the opened “TearDown Script” section, following line of code is written to call GenerateCSVReport

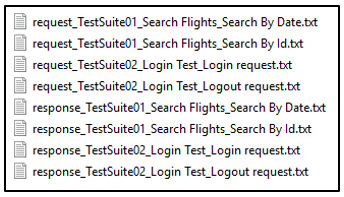
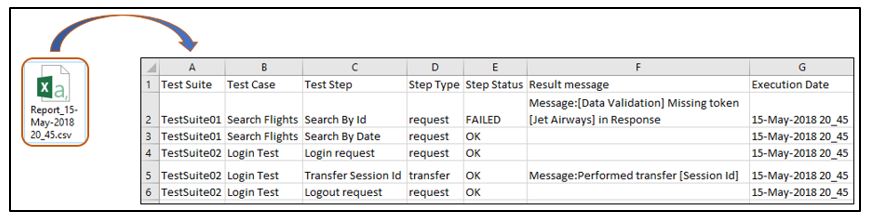
// Code to execute the GenerateCSVReport test step

testRunner.testCase.testSuite.project.testSuites["Library"].testCases["Reporting\_Utility"].

testSteps["GenerateCSVReport"].run(testRunner, context);

* Execute the SoapUI Project

Post-execution, observe project directory for the newly created folder SoapUIResults.

Inside the sub-folders of SoapUIResults, observe the files for the saved request and response and CSV Report file as shown below:

## Usage

This technique of generating reports using groovy can be used for SOAP, as well as REST project.

GenerateCSVReport test step can be customized to generate a report and populate data, based on specific requirements.