Spreadsheet Test Data Interface

Contents

[Revision history 1](#_Toc447033708)

[Overview 2](#_Toc447033709)

[Usage 3](#_Toc447033710)

[Step 1: set your test to use the Parameterized test runner 4](#_Toc447033711)

[Step 2: create your spreadsheet 4](#_Toc447033712)

[Step 3: create a public static method annotated with @Parameters that returns a Collection of Objects as the test data set. 4](#_Toc447033713)

[Step 4: add the necessary fields to your test class 5](#_Toc447033714)

[Step 5: reading test data from the spreadsheet into your test 6](#_Toc447033715)

[Step 6: saving data back to the spreadsheet 7](#_Toc447033716)

# Revision history

Created: 25/02/2016

Updated 29/03/2016 (added saving functionality)

# Overview

Junit4 introduced a feature called **Parameterised tests**, which allows the developer to run the same test over and over again using different values. With a little extra code these values can come from a CSV file. The extra code I’ve created to do this is the **csvTools** class located in the **spreadsheettools** package:



It relies on an external library for reliably manipulating CSV files:

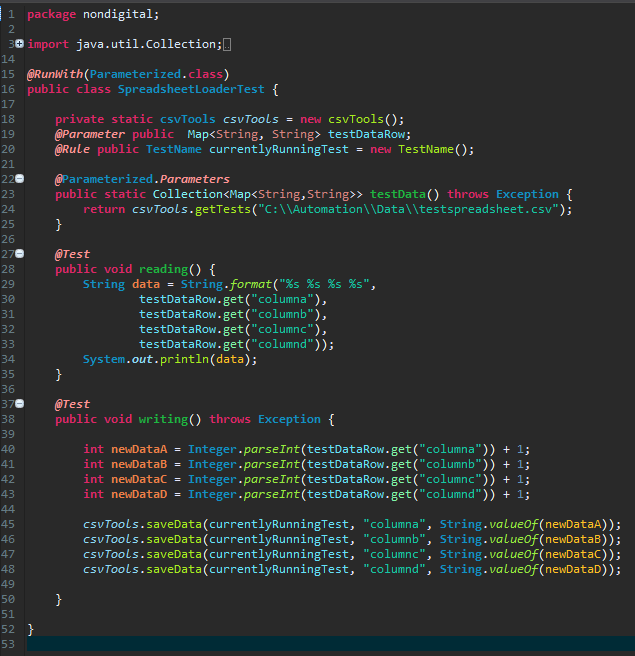


A copy of this library can be acquired form our shared drive located at [Y:\Automation\Software\opencsv-3.7.jar](file:///Y:\Automation\Software\opencsv-3.7.jar) and should be copied to your local system under “C:\Automation\Software”

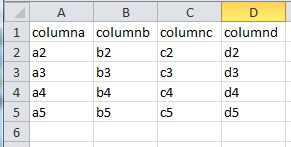
“opencsv is available under a commercial-friendly Apache 2.0 license. You are free to include it in your commercial applications without any fee or charge, and you are free to modify it to suit your circumstances.” - <http://opencsv.sourceforge.net/license.html>

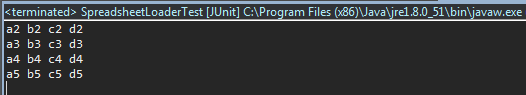
# Usage

Here are the example tests; they will be described in detail later in this document:

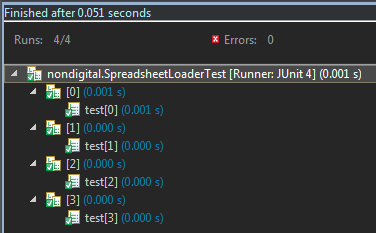


The test class contains a reference to this test spreadsheet (“C:\Automation\Data\testspreadsheet.csv”) which looks like this:



When we run the reading test it has the following output:

We can verify that it counts as four separate test runs:



You can see that one row of data in the spreadsheet corresponds to one test; in this way we can run the same test with multiple different sets of data stored in a spreadsheet.

## Step 1: set your test to use the Parameterized test runner

This is done by annotating your test class with the following line, to let Junit know how to handle it:

***@RunWith***(**Parameterized**.**class**)

**public** **class** **SpreadsheetLoaderTest** {

## Step 2: create your spreadsheet

**Important : the spreadsheet must be saved as CSV, not any other file format.**

The top row is reserved for column headers, test data will never be read from this row. The headers must be valid Java Strings, they can contain spaces, but certain characters must be escaped in accordance with the Java syntax. Generally speaking you can use any word or words you like such as “username” or “user name”.

## Step 3: create a public static method annotated with @Parameters that returns a Collection of Objects as the test data set.

***@*Parameterized**.***Parameters***

**public** **static** Collection<Map<String,String>> **testData**() **throws** **Exception** {

**return** **csvTools**.*getTests*("C:\\Automation\\Software\\testspreadsheet.csv");

}

I’ve called this method testData, but thanks to the annotation, it does not matter what you call it.

Everything else is significant. It has to be **public** **static**, it must have the return type of **Collection<Map<String,String>>** and it must throw Exceptions.

The only value that you should change is the path to your test data spreadsheet, which should be passed into the getTests method:

***getTests*("C:\\Automation\\Data\\testspreadsheet.csv");**

In accordance with the Java syntax, backslashes in the file path must be escaped as \\.

This collection of test data is not accessible from within the tests, it’s there for jUnit to use, jUnit will run the test multiple times for each item in the collection, essentially handing out one row of data to each test. Accessing the data from within the tests is described in the next sections.

## Step 4: add the necessary fields to your test class

An instance of csvTools is needed to access the spreadsheet from the tests:

**private** **static** **csvTools** *csvTools* = **new** csvTools();

csvTools is located in the spreadsheettools package, so the following line will need to be added to your import statements:

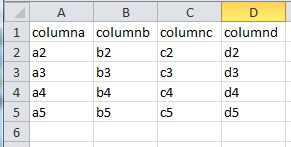
**import** spreadsheettools.csvTools;

A field is needed to store data that’s read from the spreadsheet, only one row is read each time the test is run so we can call the field testDataRow. The important part is to make sure it’s annotated with @Parameter and the data type needs to be Map<String,String>.

***@Parameter*** **public** Map<String, String> testDataRow;

Here’s a visualisation of what testDataRow will contain each time the test is run with the example spreadsheet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test 0** | **Test 1** | **Test 2** | **Test 3** |
| |  |  | | --- | --- | | **String** | **String** | | columna | a2 | | columnb | b2 | | columnc | c2 | | columnd | d2 | | |  |  | | --- | --- | | **String** | **String** | | columna | a3 | | columnb | b3 | | columnc | c3 | | columnd | d3 | | |  |  | | --- | --- | | **String** | **String** | | columna | a4 | | columnb | b4 | | columnc | c4 | | columnd | d4 | | |  |  | | --- | --- | | **String** | **String** | | columna | a5 | | columnb | b5 | | columnc | c5 | | columnd | d5 | |



When saving data back to the spreadsheet, csvTools needs to know which row to update, since only one row is used per test. This can be determined by the TestName which contains an index number. The @Rule annotation is part of jUnit and is needed to make it update each time the test is run. You can leave this line out if you don’t intend to save any data to the spreadsheet:

***@Rule*** **public** **TestName** currentlyRunningTest = **new** TestName();

But if you include it, you will need to add these import statements too:

**import** org.junit.Rule;

**import** org.junit.rules.TestName;

## Step 5: reading test data from the spreadsheet into your test

All the information for each test run is stored in the variable testDataRow (as described in the previous section). To get data for a specific column, you simply call the .get method with the column name as the argument:

**String** **data** = **String**.*format*("%s %s %s %s",

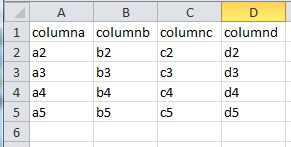
testDataRow.get("columna"),

testDataRow.get("columnb"),

testDataRow.get("columnc"),

testDataRow.get("columnd"));

**System**.***out***.println(data);

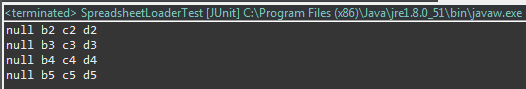


Let us observe what happens if we make the following change to the example test and run it:

testDataRow.get("columna");

Change to:

testDataRow.get("blah");



The csvTool is unable to find any column with the heading “blah”, and will output nulls instead.

You can call testDataRow.get(“column name”) anywhere in your test method where you need to use the data. The only thing to bear in mind is that the data type it returns will always be String. This is appropriate for 90% of the data in Selenium tests, but sometimes you will have to convert from String to another data type.

## Step 6: saving data back to the spreadsheet

From anywhere in your test method you can call the csvTools.saveData method like this:

*csvTools*.saveData(currentlyRunningTest, "columna", “new value”);

It takes three Strings as arguments, the first is currentlyRunningTests, which you should have created in [Step 4](#_Step_4:_add), the second String is the column heading, which will be specific to your data spreadsheet, and the third String is the value you want to save. If you want to save a different data type such as an Integer, you will need to convert it to a String first.

The example test will add 1 to every number in the spreadsheet:

***@Test***

**public** **void** **writing**() **throws** **Exception** {

**int** **newDataA** = **Integer**.*parseInt*(testDataRow.get("columna")) + 1;

**int** **newDataB** = **Integer**.*parseInt*(testDataRow.get("columnb")) + 1;

**int** **newDataC** = **Integer**.*parseInt*(testDataRow.get("columnc")) + 1;

**int** **newDataD** = **Integer**.*parseInt*(testDataRow.get("columnd")) + 1;

*csvTools*.saveData(currentlyRunningTest, "columna", **String**.*valueOf*(newDataA));

*csvTools*.saveData(currentlyRunningTest, "columnb", **String**.*valueOf*(newDataB));

*csvTools*.saveData(currentlyRunningTest, "columnc", **String**.*valueOf*(newDataC));

*csvTools*.saveData(currentlyRunningTest, "columnd", **String**.*valueOf*(newDataD));

}