

TMar, executable specification and automated acceptance tests

Version 1.5, July 2014

Tmar – Introduction



- Tmar is a companion for standards java/groovy testing Framework (JUnit, TestNG, Spock)
- It brings the capacity to share executable specifications, test descriptions and test results between functional team and developers
- Originally Inspired by Fitness, we tried to keep its utilization as well as its integration as simple as possible.
- Tmar comes from the frustration to have to change our actual testing framework to be able to have external test descriptions shared with our functional team members

Tmar – first simple example (Junit/TestNG)



TestDemo.multiply_test1.tmar

Multiplication test /each [number 1 | number 2 | result ? | | 2 | 2 | 4 | | 2 | 3 | 6 |

TestDemo.groovy

```
class TestDemo extends Tmar4JUnit {
    @Test
    def multiply() {
        eachIteration('multiply_test1'){ tmar ->
            tmar.result = tmar.number1 * tmar.number2
        }
    }
}
```

TestDemo.multiply_test1.html



OK - Multiplication test

2 iteration(s) with 0 in error - 26/10/13 00:51:06 - TestDemo.multiply_test1

EACH

number1	number2	result
2	2	4
2	3	6

Tmar – first simple example (Spock)



TestDemo.multiply_test1.tmar

Multiplication test

/each
[number 1 | number 2 | result ? |
| 2 | 2 | 4 |
| 2 | 3 | 6 |

TestDemo.groovy (with Spock)

```
class TestDemo extends Tmar4Spock {
    def multiply() {
        when:
            tmar.result = tmar.number1 * tmar.number2
            then:
                 tmar.asserts()
            where:
                 tmar << getData('multiply_test1')
            }
        }
}</pre>
```

TestDemo.multiply_test1.html



OK - Multiplication test

2 iteration(s) with 0 in error - 26/10/13 00:51:06 - TestDemo.multiply_test1

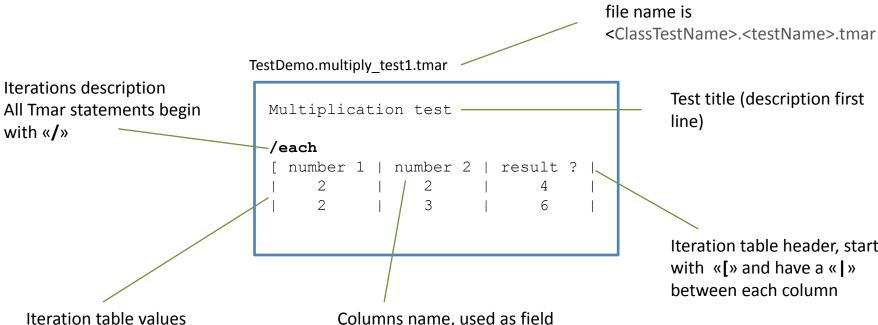
EACH

number1	number2	result
2	2	4
2	3	6

Tmar – first simple example



Simple Tmar description file



Iteration table values Columns are delimited by a « | » The test method will be fired for each line of the /each table

name for each Iteration. Camel case convention is applied when fields are given to the test method

number 1 \rightarrow number 1 First Name → firstName

Iteration table header, start with «[» and have a «|»

By convention Tmar test description

When a column name is followed by «?» Tmar will compare the value of this column with the value returned by the test method.

If there is no value in the column Tmar will display the value returned by the test method.

Tmar – first simple example



Simple JUnit or TestNG - Tmar test class

@Test

JUnit or TestNG annotation for Test method

tmar (iteration 1)

number1 = 2 Number2 = 2 Result = ?

For each iteration the business rules assign values to the fields expected by the test description (here: result)

TestDemo.groovy

```
class TestDemo extends Tmar4JUnit {

@Test
def multiply() {
    eachIteration('multiply_test1'){

2 tmar.result = tmar.number1 * tmar.number2
}

}
```

TestDemo.multiply_test1.tmar

```
Multiplication test

/each
[ number 1 | number 2 | result ? |
| 2 | 2 | 4 |
| 2 | 3 | 6 |
```

Test class extends **Tmar4JUnit** or **Tmar4TestNG**

eachIteration() brings values of each iteration in a Tmar object from the Tmar description

tmar (iteration 1)

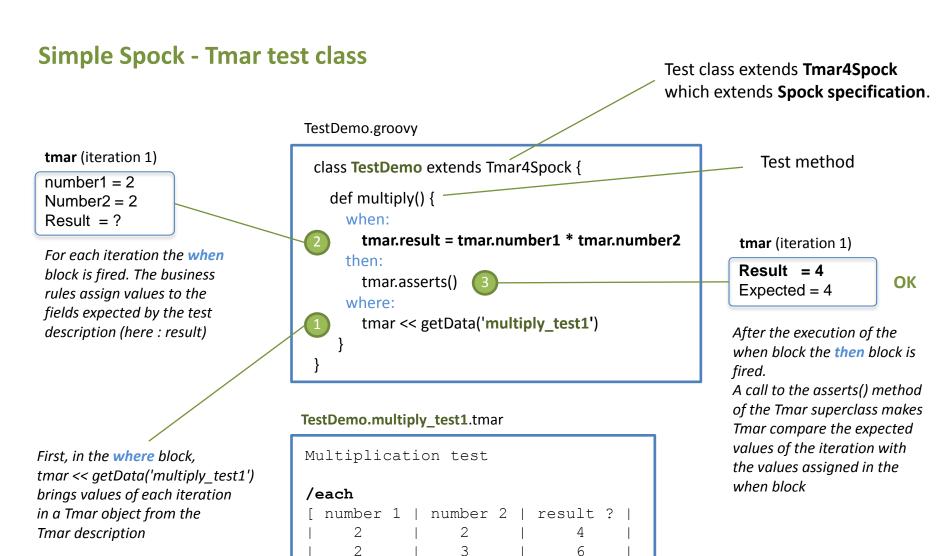
Result = 4 Expected = 4

OK

Tmar compare the expected values of the iteration with the assigned values

Tmar – first simple example





Tmar – use of text



 Tmar allows for text in your description in order to comment or explain your test case

Multiplication test
This text explain the purpose of the test
/each
[number 1 number 2 result ? 2 2 4
It's also possible to have text between table's lines to explain the iteration
2 3 6

Tmar – /inline statement



- To be even more expressive Tmar supports the /inline statement.
 It allows to spread the fields of an iteration in a text bloc
 - each /inline and each in tabulation can be freely mixed in the same description
- It also supports text fields value longer than one line of text using """ """

```
Multiplication test
/each
[ number 1 | number 2 | result ? | text 1
/each
/inline
If we have [number 1] 2 | and [ number 2 ] 4 |
the multiplication will have for [ result ? ] 8 | and [text 1| height |
/inline
The same operation can be done with [number 1] 2 | and [ number 2 ] 5|
In this case the multiplication will have for [ result ? ] 10 |
[text 1] """
This text doesn't fit in one
line of text
11 11 11
```

Tmar – values type & comparisons



- By default Tmar automatically manages four types of data
 - Text (default), integer, decimal and boolean (yes, no, true, false)
 - This means that Tmar will try to automatically convert the input data before passing them to the test method (we will see later how to override this behavior)
 - It's possible to enforce the string type by using "" (example "003")

```
Multiplication test
/each
                           code
                                    married
  name
  Jenny
                                      ves
                           "002"
                  35
                                                 1.80 I
  peter
      String
                     Integer
                               String
                                        Boolean
                                                    Double
```

Values Comparison

- As we have seen previously, a trailing "?" into the column name instructs Tmar to consider the column as a result column. For each Iteration the value in the column will be compared to the returned value computed by the test method
- If there is a single value in the column, Tmar will test equality against the returned value
- It's also possible to use the following expressions (where "?" is the returned value)

Tmar – Assertion error



 Tmar displays assertion errors in the html report file and in the IDE's output console:

EACH		
number 1	number 2	result
2	7	14 <13

 If you use the Tmar Eclipse Plugin, the error is also directly highlighted in the Tmar description

Tmar – second example

Multiplication test

/each

[number	1	number	2	result	?	
	2		2		4		
	2		3		6		

It's possible to spread the fields of an iteration in a text with the statement /inline

/each

/inline

If we have [number 1] 2 | and [number 2] 4 | the multiplication will have as [result ?] 8 |

/inline

The same operation can be done with [number 1] 2 \mid and [number 2] 5 \mid

In this case the multiplication will have as [result ?] 10 \mid

/each

It's also possible to have some text between iterations lines, For example :

This line is testing that 2 * 6 >= 12

and this one that 2 * 7 < 14, it will raise an error. As you can see it's not necessary to repeat the header before each raw

2 | 7 | ? < 14 |

1 ERROR(S) - Multiplication test

6 iteration(s) with 1 in error - 26/10/13 02:03:37 - TestDemo.multiply_test1

EACH

number 1	number 2	result
2	2	4
2	3	6

It's possible to spread the fields of an iteration in a text with the statement inline

EACH inline

If we have number1:2 and number2:4The multiplication will have for result:8

EACH inline

The same operation can be done with number1: 2 and number2:5 In this case the multiplication will have for result:10

It's also possible to have some text between iterations lines, For example:

This line is testing than 2 * 6 >= 12

EACH

number 1	number 2	result
2	6	12

and this one than 2 * 7 < 14, it will raise an error. As you can see it's not necessary to repeat header before each raw

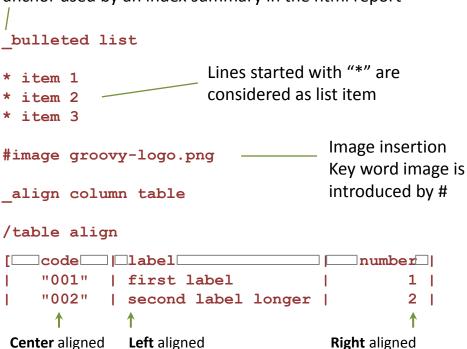
EACH

number 1	number 2	result
2	7	14 < 14

Tmar – Improved html presentation report



A line started with "_" is considered as a subtitle. It will also be an anchor used by an index summary in the html report



Tmar considers the column header to determine column alignment If the column header right white space is twice as big as the left white space the column is left align

If the left white space is twice as big as the right white space the column is right align, otherwise the column is centered

Tmar also support Markdown formating langage to improve report presentation (more detail at the end of the document)

bulleted list

- item 1
- item 2
- item 3



align column table

TABLE align

code	label	number
001	first label	1
002	second label longer	2

Tmar – images, documents and href



- Images can be added to your Tmar pages
 - #image <image name>

```
#image myImage.jpeg
```

- The image has to be in a /images subdirectory of the Tmar description directory tree
- Tmar also let you add some href links
 - #url <link>,<label>;

```
#url http://www.jspresso.org/,Jspresso site;
```

- And with a similar syntax Tmar let you add links to documents
 - #document <document name>,<label>;

```
#document spec.ppt,Specifications;
```

 The linked document has to be in a /documents subdirectory of the Tmar description directory tree

Tmar – execution in sequence with Spock



- Using the /each statement, each iteration are run separately and in parallel
- It's sometimes necessary to ensure that iterations are run in order. This is especially true if you want to use the result of the previous iteration
- To do so, you can use the /sequence statement instead of the /each one
- In that case getData() will return an Object with a list of iterations and you
 will have to use the tmar.hasNext() method to iterate in.

TestDemo.accumulation_test1.tmar

acc	umulati	on	of valu	es					
	quence umber 1	.	number	2	result	?	accumulated	?	
Ì	2	İ	2	İ	4	İ	4		
1	2		3		6	-	10		-
I	2		4		8		18		1



OK - Accumulation values

3 iteration(s) with 0 in error - 04/11/13 15:16:19

SEQUENCE

number 1	number 2	result	accumulated
2	2	4	4
2	3	6	10
2	4	8	18

TestDemo.groovy

```
class TestDemo extends Tmar4spock {
    def accumulated() {
        Integer cumul = 0
        when:
        while (tmar.hasNext()) {
            def total = tmar.number1 * tmar.number2
            tmar.result = total
            cumul = cumul + total
            tmar.accumulated = cumul
        }
    then:
        tmar.asserts()
    where:
        tmar << getData('accumulation_test1')
    }
}</pre>
```

Tmar – Dealing with scenario (Junit/TestNG) 3



- The easiest way to deal with a scenario is to use a switch statement on an iteration field.
- This lets you run different business rules depending of this field value

TestDemo.test_account_operations.tmar

Bank accoun	t	operatio	ns	tests			
/each							
[operation		amount		balance	?		
deposit		2000		2000			
deposit		500		2500			
withdraw		700		1800			
deposit		500	 	2500			



OK - Bank account operations tests 3 iteration(s) with 0 in error - 04/11/13 14:51:49

SEQUENCE

operation	amount	balance
deposit	2000	2000
deposit	500	2500
withdraw	700	1800

TestDemo.groovy

```
class TestDemo extends Tmar 4JUnit {
  def accountOperations() {
    Integer balance = 0
    eachIteration('test account operations'){ tmar ->
       switch (tmar.operation) {
       case 'deposit':
           balance = balance + tmar.amount
           break
       case 'withdraw':
           balance = balance - tmar.amount
           break
       tmar.balance = balance
```

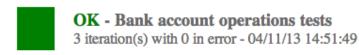
Tmar – Dealing with scenario (Spock)



- The easiest way to deal with a scenario is to use a /sequence with a switch statement on an iteration field.
- This lets you run different business rules depending of this field value

TestDemo.test_account_operations.tmar

Bank account	operations	tests
/sequence [operation	amount	balance ?
deposit	2000	2000
deposit withdraw	500 700	2500 1800



SEQUENCE

operation	amount	balance
deposit	2000	2000
deposit	500	2500
withdraw	700	1800

TestDemo.groovy

```
class TestDemo extends Tmar {
  def accountOperations() {
    Integer balance = 0
    when:
      while (tmar.hasNext()) {
         switch (tmar.operation) {
            case 'deposit':
              balance = balance + tmar.amount
              break
           case 'withdraw':
              balance = balance - tmar.amount
              break
         tmar.balance = balance
    then:
      tmar.asserts()
    where:
      tmar << getData('test account operations')
```

Tmar – get current iteration information



- Tmar provides three methods for this purpose :
 - getCurrentIterationNumber()
 - getIterationHeader() which returns a map of fields where
 key = fieldName and value = Boolean value which is true if the field is a result field
 (? In the column description)
 - getIterationValues() which returns a map of fields where
 key = fieldName and value is the value of the field from the Tmar description
 - Only non-result fields are present in this Map

TestDemo.accumulation_test1.tmar

```
tmar.getIterationHeader() == ['number1':false, 'number2':false,
'result':true, 'accumulated':true]
```

```
tmar.getIterationValues() == ['number1':2, 'number2':2] (first iteration)
tmar.getIterationValues() == ['number1':2, 'number2':3] (second iteration)
```

Tmar – Table support



- Tmar allows you to declare a table of data in your description in order to pass contextual values to your test
- A table is introduce by the /table statement followed by the table name
- Header and values follow he same rules as /each table

```
/table countries
[ code | name |
| FR | France |
| US | United States |
| BR | Brazil |
```

- In your test code you can obtain table values from the Tmar object of the current iteration
 - Example :

```
List countryCodes = tmar.getTableColumn("countries","code")
```

Tmar – Table support



The following methods are available on the tmar object

```
tmar.getListFromTable ('countries') == [['code': 'FR', 'name': 'France'],
  ['code': 'US', 'name': 'United States'], ['code': 'BR', name: 'Brazil']]
tmar.getTable ('countries') == ['code':['FR','US','BR'],
                                 'name': ['France', 'United States', 'Brazil']
tmar.getTableLine('countries',0) == ['code':'FR','name':'France']
tmar.getTableColumn('countries','code') == ['FR','US','BR']
tmar.getTableValue('countries',0,'name') == 'France'
tmar.getTableHeader('countries') == ['code','name']
tmar.getTableSize('countries') == 3
```

Tmar – Map support



- In addition to table, Tmar allows you to declare map in your description to give context value to your test
- A map is introduce by the /map statement followed by the map's name
- Header and values follow he same rules than table, the difference is than the first column is a single key

```
/map countries
[ code | name |
| FR | France |
| US | United States |
| BR | Brazil |
```

Tmar – Map support



The following methods are available on tmar object

```
tmar.getListFromMap('countries') == [['code': 'FR', 'name': 'France'],
          ['code': 'US', 'name': 'United States'], ['code': 'BR', name: 'Brazil']]
tmar.getMap('countries') == ['FR':['code': 'FR', 'name': 'France'],
'US':['code': 'US', 'name': 'United States'], 'BR':['code':'BR', name:'Brazil']]
tmar.getMapsAsTable('countries') == ['code':['FR','US','BR'],
                                  'name':['France', 'United States','Brazil']
tmar.getMapLine('countries','FR') == ['code':'FR','name':'France']
tmar.getMapColumn('countries','name') == ['FR':'France,'US':'United States',
                                          'BR':'Brazil']
tmar.getMapColumnAsTable('countries','name') == ['France,'United States',
                                                   'Brazil'l
tmar.getMapValue('countries','BR','name') == 'Brazil'
tmar.getMapHeader('countries') == ['code','name']
tmar.getMapSize('countries') == 3
```

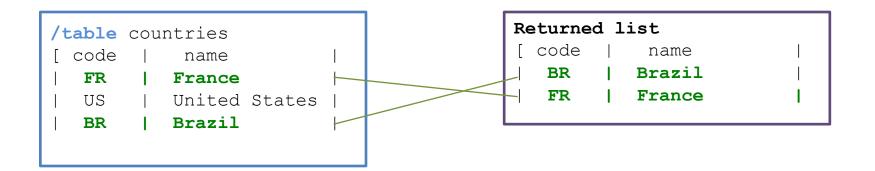
Tmar – Table & list comparison



- In some cases it's interesting to check the validity of a list returned by a test method
- Tmar has four operators to compare a returned list with a table content
 - "?" is the returned list and @tableName the table content
 - With in and orderedIn, the result list has to be a subset of the referenced table
 - With match and orderedMatch, both must have the same number of lines

The "?" can be put at the right of the expression. In that case, with in and orderedIn operators, the referenced table has to be a subset of the result list





? in @countries

ok

? orderedIn @countries

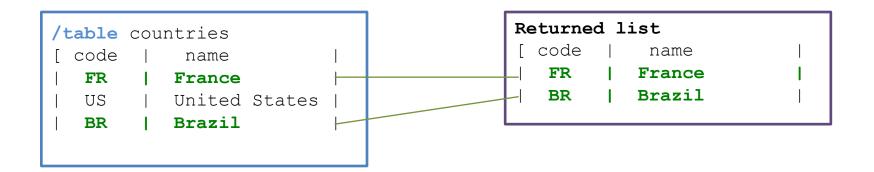
ko

? match @countries

ko

? orderedMatch @countries





? in @countries

ok

? orderedIn @countries

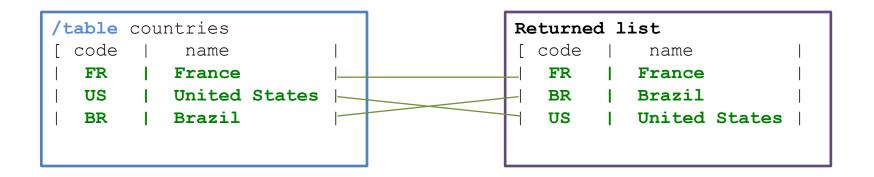
ok

? match @countries

ko

? orderedMatch @countries





? in @countries

ok

? orderedIn @countries

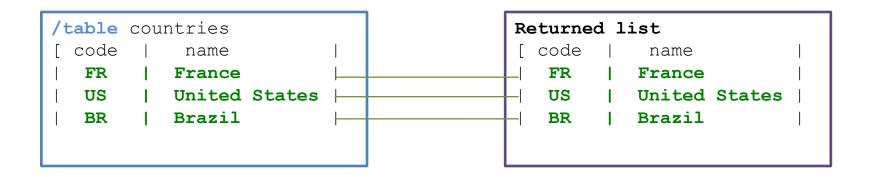
ko

? match @countries

ok

? orderedMatch @countries





? in @countries

ok

? orderedIn @countries

ok

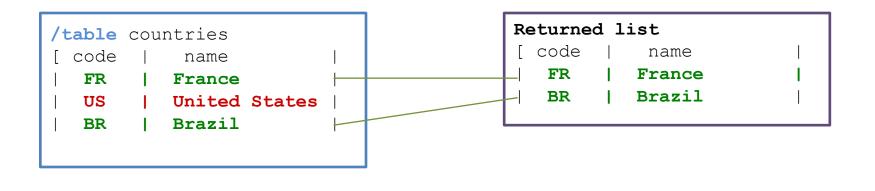
? match @countries

ok

? orderedMatch @countries

ok





? in @countries

ok

@countries in?



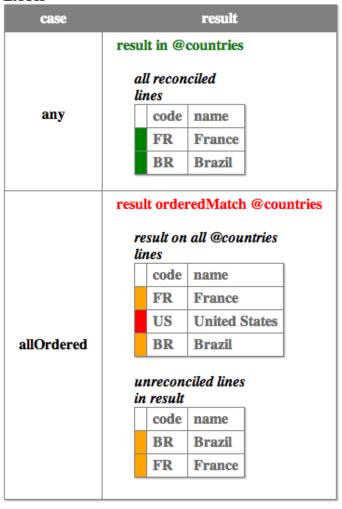
TestDemo.testCountries.tmar

```
Test matching countries
/table countries
[ code
            name
   FR
           France
   US
        | United States
           Brazil
   BR
/each
[ case
                      result ?
                 ? in @countries
   any
                 ? orderedMatch @countries
   allOrdered |
```

TestDemo.groovy

```
def matchCountries() {
   when:
     tmar.result = [[ 'code', 'name'],
                      ['BR','Brazil'],
                       ['FR','France']]
   then:
     tmar.asserts()
   where:
     tmar << getData('testCountries')</pre>
```

EACH



Reconciled line



Unreconciled line



Unordered reconciled line

Tmar – Table & list comparison, returned list format



- For maximum flexibility, Tmar allows for three multi-dimensional formats as return value:
 - List of List, with header columns name in the first list and lines values in the following

List of Map, with a Map of fields name and value for each line

Map of List with a Map of columns with the list of all the column's values

Tmar – Table & list comparison, null and empty



- If a test return a null object the result will always be an error
- If the test return an empty object the result will be :
 - an error for the "macth" and "orderedMatch" operators
 - A success for the "in" and "orderedIn" operators

Returned	Expression	result
null	? in @countries	ko
null	? orderedIn @countries	ko
null	? match @countries	ko
null	? orderedMatch @countries	ko
[]	? in @countries	ok
[]	? orderedIn @countries	ok
[]	? match @countries	ko
[]	? orderedMatch @countries	ko

Tmar – include files



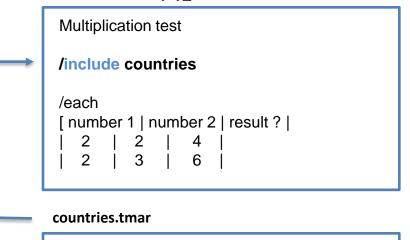
- Tmar "includes" let you share Tmar descriptions between Tmar files
- Include files are introduced by the /include statement and Tmar supports nested include files
 - For example :

TestDemo.multiply_test1.tmar

/table countries

code

BR



name France

Brazil

United States



```
Multiplication test

/table countries
[ code | name |
| FR | France |
| US | United States |
| BR | Brazil |

/each
[ number 1 | number 2 | result ? |
| 2 | 2 | 4 |
| 2 | 3 | 6 |
```

Tmar – Share context



- Tmar lets you load a Tmar description file at the class level
 - By convention this file will be named <className>.tmar and has to be in an includes subdirectory
- All tables declared in this file will be seen from all iterations of all test cases of this class
- To load this description file you have to call the method loadSharedContext()
 in the method setupTest() for JUnit/TestNG or setupSpec() for Spock

```
class TestDemo extends Tmar4JUnit {
  void setupTest() {
    def mySharedContext = loadSharedContext()
  }
  test methods ...
}
```

- The returned object mySharedContext has all the needed methods to get data from the shared context tables (getTable, getColumns, getValues... see table and Map support)
 - Those methods are useful if you need to initialize your own shared objects at this point with data coming from the Tmar shared context

Tmar – Iteration's context



- Each iteration may have a different execution context
- Fields in /each or /sequence iteration may be of different types. For example, the following description is valid

```
Bank account operations tests
/sequence
[operation |
              amount |
                         balance ?
| deposit
                2000 |
                            2000
 deposit
                 500 I
                            2500
                 700 |
                            1800
 withdraw |
/sequence
[ operation
                     account
                                      date
                                               | balance ?
  information
                                 | 01/01/2013 |
                   "9700990991"
                                                     900
```

Is that case tmar.getIterationHeader() gives the available fields

```
tmar.getIterationHeader() == ['operation':false, 'amount':false, 'balance':true]
```

```
tmar.getIterationHeader() == ['operation':false, 'account':false, 'date':false,
'balance':true]
```

Tmar – Iteration's context



- Tables content may also be different between iterations
- Indeed, iterations only see table content defined before it's own declaration in the description file. A table content can be fed several times so that the table content is progressively available.

```
Bank account operations tests
/table valid account
[account number |
 "9700990991" | John Smith
/sequence
[ operation
                  account
                                    found ?
  information
                  "9700990991"
| information
                  "5700550552" I
/table valid account
[account number |
                  owner
| "5700550552"
               | Edith Wilton|
/sequence
[ operation
                    account
                                    found ?
| information
                  "5700550552" I
                                     ves
```

Tmar – Override value transformation



- By default, Tmar tries to parse Integer, Double and Boolean types.
- To override this behavior you can use the method setTransformValue() in the method setupTest() for JUnit/TestNG or setupSpec() for Spock
- For table, or a table column, you are able to set a closure that overrides the default transformation

```
class TestDemo extends Tmar4JUnit {

void setupTest() {
    setTransformValue('countries','code'){ value ->
    return "ISO-$value"
    }

    Will returned
    "ISO-FR"
    "ISO-US"
    "ISO-BR"
}
```

Obviously this method also allows to change value type

Tmar – working directories & files



- Tmar working directories are defined in the tmar.config
 - This file has to be in a tmar directory in the test project resources directory

```
myProject/core/src/test/resources/tmar/tmar.config
```

Default tmar.config content

- Tmar's directories
 - DESCRIPTION_DIRECTORY : Tmar test description files
 - className.testName.tmar
 - RESULT_DIRECTORY: json execution result files per iteration
 - className.testName.iterationNum.result
 - REPORT_DIRECTORY : html test execution reports
 - className.testName.iterationNum.html (test report, one for each .tmar file)
 - TmarIndexReport.html (global index of the test html reports)

Tmar – Description directory



- Tmar provides great flexibility in organizing description and include files in the description directory and subdirectories
- The description directory may have as many subdirectories as necessary
- Description files can be anywhere in the tree
 - When a description file is call by a test method only the test name is given to Tmar
 - Tmar build the full name with the class name and a .tmar extension
 - Then Tmar scans the tree to find the description file
- With this approach the functional team may reorganize the description directory without any impact on the test code
- By convention
 - Include files have to be in subdirectories named "includes".
 - Images files have to be in subdirectories named "images".
 - Documents files have to be in subdirectories named "documents".

include, image and document files can be spread among as many directories as needed in the description tree.

Tmar description tree example

Tmar – Index.html report



- Tmar build an index.html file report to index each Tmar test report
- The project description files directories tree is used to build this report
- For each directory the report lists the test reports and point out their status
 - No error
 - At list one test iteration is in error
 - Tmar description file found but not executed
- Each test result line is a link to the detail test report

Tmar – Customize index report header



Default index.html report header is a Jspresso header



tmar.config file allow to define your own logo and title



Tmar – <name>.chapter.tmar



- A challenge for test result report is to make the link with project documentation.
 - The tags #url and #documents allow to make this kind of link from the tests report files, but often the perimeter of a documentation is much bigger than a test description file itself.
- To address this issue Tmar has special description files with the name finished by chapter.tmar
 - Those file are tmar files but without /table, /each, /sequence or /include statement. Just formatted text, #image, #url and #documents
 - Each time Tmar find a chapter.tmar file in a directory, it include his content in the index.html file.
 - Then the directory name become a link in the index.html file which show/hide the content of the chapter.tmar file
 - This way your able do make descriptions and link to documentations at a chapter level for a group of Tmar test description.

Tmar – Markdown formatting language 🔀 Jspresso



- To allow text formatting, TMar support the Markdown formatting language (thanks to txtmark processor)
- Due to syntax overlap with Tmar, <u>link and image Markdown syntax is not</u> <u>supported</u>. You have to use #url, #image and #document Tmar tags instead.
- Markdown syntax example :

```
# Big Title (HTML H1)
##Other title (HTML H2)

*Italic*, **bold**
```

Big Title (HTML H1)
Other title (HTML H2)

Italic, bold

Tmar – Manually running the global test report



- Each time you run a test, the corresponding html report is generated
- Generally, your global html report is generated by the maven build or by the Tmar Eclipse plugin
- However, whenever needed, it's very simple to generate it by running this simple test

```
class TestTmarReport extends GroovyTestCase {
  void testTmarReport() {
    TmarReport tmarReport = new TmarReport()
    tmarReport.generate()
  }
}
```

Tmar – Plugin



