Meteor Manual

Tests

January 2021

Sommaire

[1. Introduction 3](#_Toc61013734)

[2. Usage 3](#_Toc61013735)

[3. Installation 3](#_Toc61013736)

[4. Build a scenario 3](#_Toc61013737)

[4.1. Scenario definition 3](#_Toc61013738)

[4.2. Experience scenario 4](#_Toc61013739)

[4.3. Process scenario 6](#_Toc61013740)

[5. Execution 8](#_Toc61013741)

[6. Unit Test 8](#_Toc61013742)

[7. Load/Save 8](#_Toc61013743)

[7.4. Load/Save 9](#_Toc61013744)

[7.5. Export / Import the scenario 9](#_Toc61013745)

[8. External execution 9](#_Toc61013746)

[8.6. By a REST CALL 9](#_Toc61013747)

[8.6.1. Start 9](#_Toc61013748)

[8.6.2. Status 10](#_Toc61013749)

[8.7. Scheduler (Truck milk) 10](#_Toc61013750)

[9. Use case 10](#_Toc61013751)

[9.8. CI environnement 11](#_Toc61013752)

[9.9. Certification usage 11](#_Toc61013753)

[9.10. Load a server 11](#_Toc61013754)

Content

# Introduction

This document explains how to use the Meteor Page.

# Usage

The page may be use in different context:

* To advance a case at the task
* To realize a load test
* To execute unit test
* To run a Regression and performance test, in a CI environment

# Installation

The Meteor page is a Bonita custom page. To install it:

* From the Food truck, click on the icon and download it
* Directly form the Bonita Community or the Github project. Then as an administrator, go to the Bonita Portal / resource and import the ZIP file

Then, register the page in a Profile or in an Application (Community version).

# Build a scenario

A Scenario is the basis to execute a test. A scenario contains data to creates a case in a process, and the list of tasks to be executed.

A scenario contains information to execute a case (information and data to create the case, task to execute and data to execute them), and the context of execution: how many cases has to be created, with the delay between the case creation, and the number of execution in paralleled.

Let review information on a scenario.

## Scenario definition

A scenario group a list of operations. An operation is a series of action: create a case, wait for a human task and then execute it. An operation is executed by a Robot, and there is one thread per Robot.

So, basically, when a Robot is instantiated, it will execute actions. An action is to create a case in a process, then search for a human task. When the human task is accessible, the robot executes it. A Robot may replay then operations (create a new case, search for human tasks). A robot will just search for human tasks, because all services tasks are executed by the engine.

## Experience scenario

A Experience scenario is calculated from an existing case. This is the simplest way to create a scenario: create a case, execute all the different human task you need, and when the case is in the expected situation, creates the experience form it.

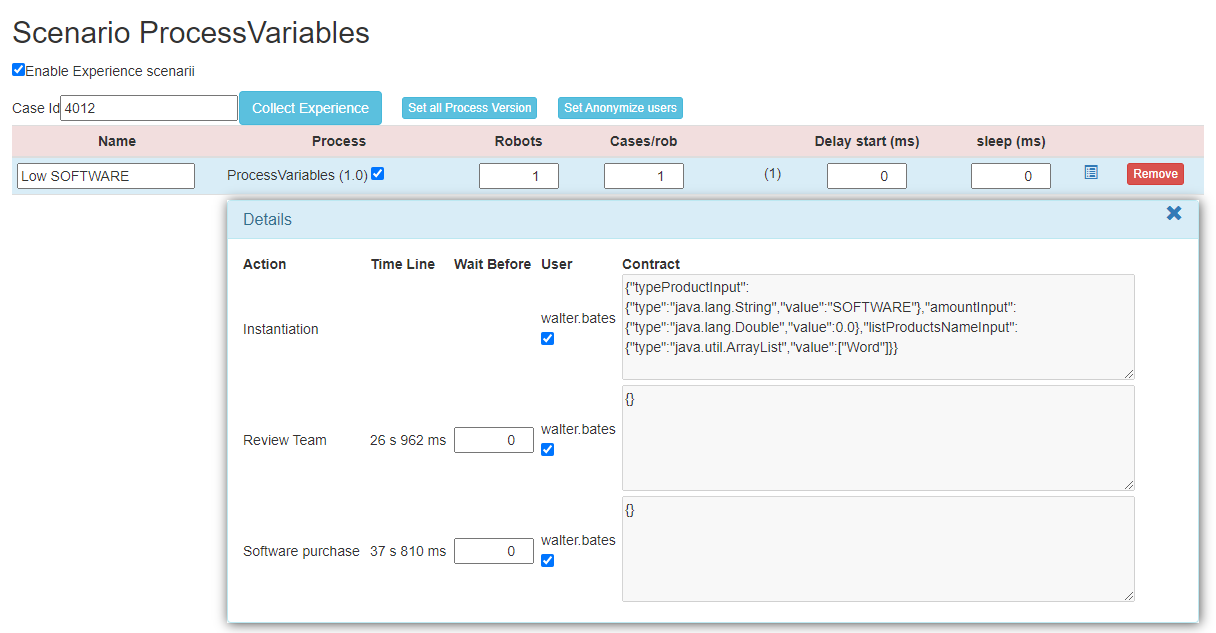
It’s very useful for any regression test: you want to be sure, day after day, that if you give the same value, case follow the same path. For example, if you give at amount 1200 €, then the human task “Review by director” appears.

The second main usage is to advance a case to a specific task. Imagine that you want to develop a connector in the task “Send Review”. But this task is after 10 human tasks. So, every time you deploy the process, you have to execute 10 human tasks. Create a case, execute one time the 10-human task, then build the scenario from this case. Meteor will replay then for you these tasks.

To collect the case information, give the caseId in the Collect Experience.



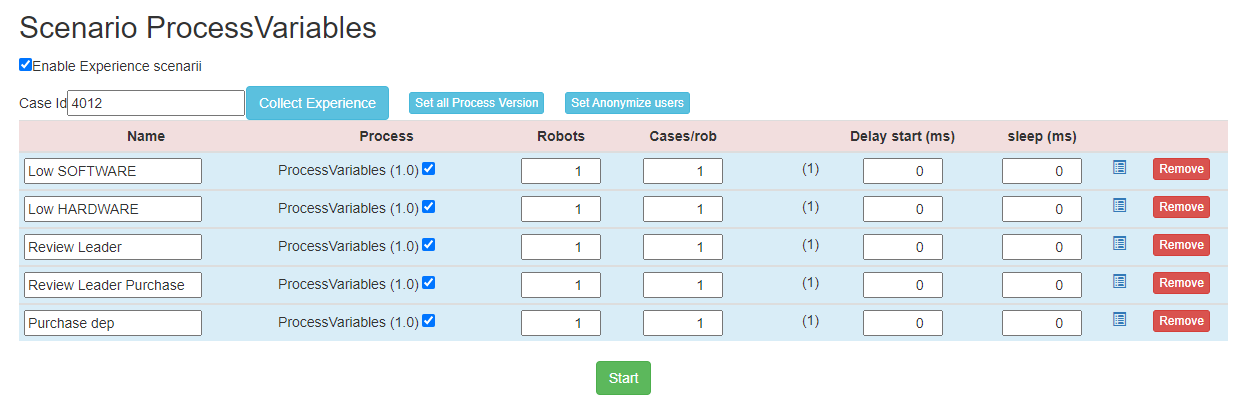
Then, a new line shows up.



During the operation:

1. Meteor investigate the case (root case and any sub process case). It will collect the contract to create the case, and each contract on any human task. It calculates to the order of the execution. For example, the task “Review team” has to be executed before the “Software purchase”. You can access this information by clicking on the “Access detail” icon.
2. **Wait Before:** By default, there is no temporization between two tasks. When the robot found and execute a task, it will immediately search for the next one. It will try to search 10 times, and between each search, it will have a delay of 500 ms. If you know that a task needs more time to show up (you execute a connector, and you know that it need one minutes to be complete), then increase the delay between each task.
3. **User:** Meteor detect which user creates the test, of execute a task, and reuse this user to replay the same operation. You can check the box “Any user”. When you do that, if the user specify doesn’t exist, any active user will be used. This is very useful when you create the scenario with an organization(on the studio, using Walter.Bates) and then execute this scenario on a Validation platform, where user may be different.
4. **Process version**: Experience detect which process the case executed (in this example, this is process variables, version 1.0. If you checked this box, then Meteor will search not an explicit version, but the last deployed version of the process. So, if you deploy a version 1.1, then the test will be executed in this new version.
5. **Robots**: You can setup the number of Robots you want to execute. For a load test, then you can specify multiple robots. Keep in mind a robot is executed in a thread, so all robots are executed in paralleled
6. **Cases/Robots**: you can specify how many times this sequence of action has to be executed.
7. **Delay start(ms)** to be realistic, you may want to delay the startup of the robot. When you execute multiple time operation (via the cases/robot parameter) Meteor will wait this delay between each loop.
8. **Sleep(ms):** this delay is used to sleep between each action on the operation. So actualy, Meteor wait between each operation (Sleep + Wait Before) millisecond.

In a scenario, you can collect multiple cases, in order to create a complete scenario. You can rename the different line.



In this situation, with this different use case, the goal is to have a complete cover of the process, in order to execute each part of the Process Variable process.

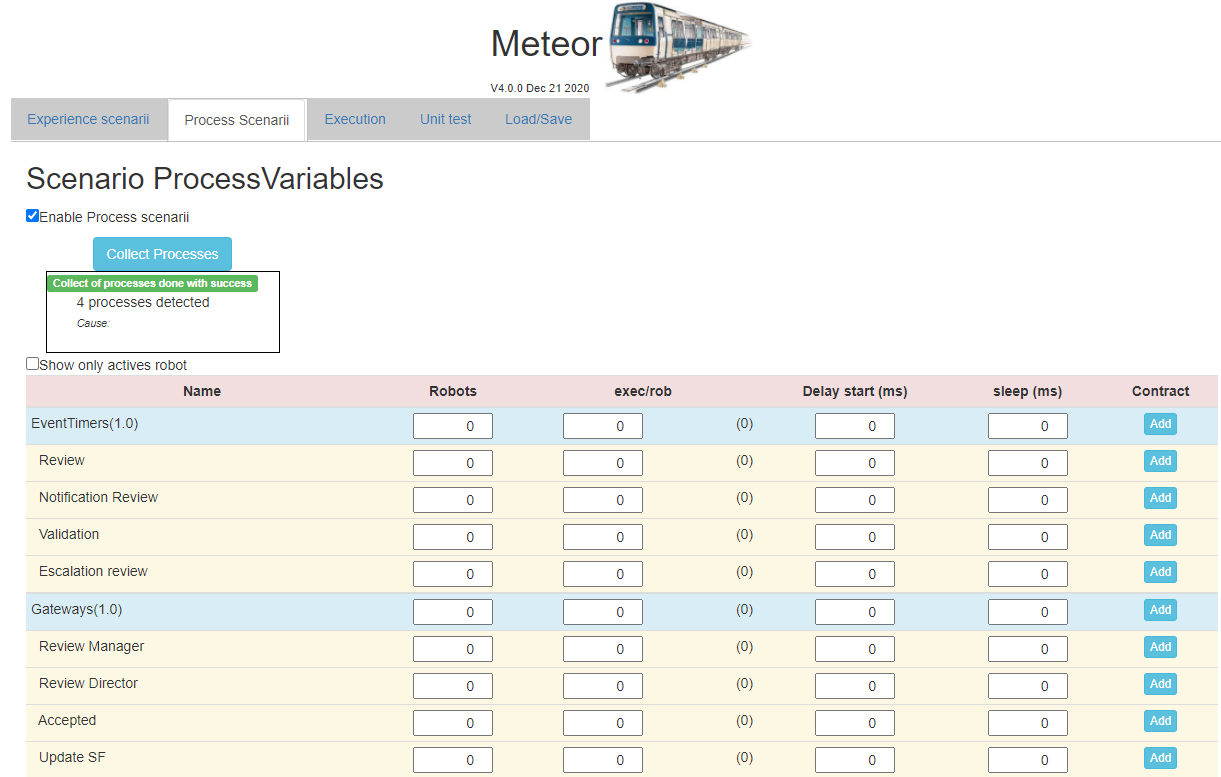
Note: when a scenario is ready, you can save it, export it to another platform (see the load/save part).

## Process scenario

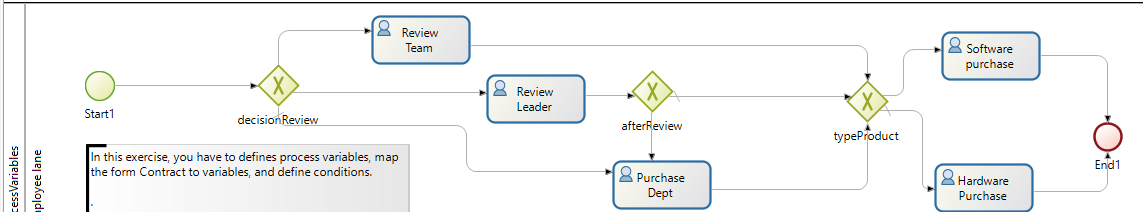
The process scenario uses a different approach. It will produce all the different action a user can do on all processes (create a case, execute a task), and then you can specify action to execute.

This is usefull where there is no “experience” possible. Example, an external system creates multiple cases, and you want to search and executes tasks. Or you want to create multiple cases at a moment, and tasks created is not in the same moment, but may show up hour after.

In this mode, click on the “Collect Processes” button. Meteor checks all processes / tasks available in your server and give you a list of all items.



For example, for a process like this



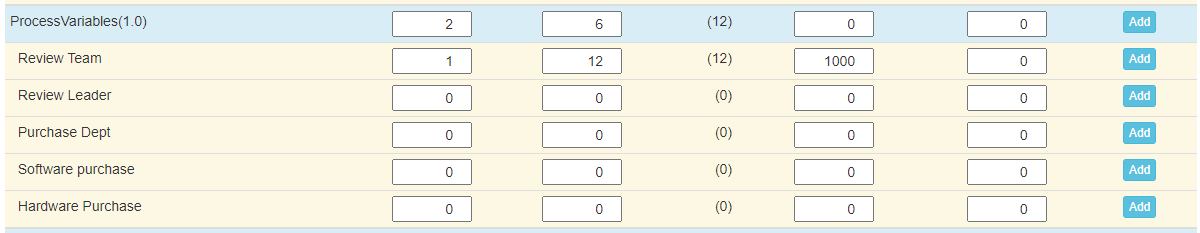
Meteor will detect theses items



The first line (in blue) is used to create new cases. Attributes are

1. **Robots**: a robot will execute the operation. A robot is a thread, so specifying 10 robots means to have the execution in parallels.
2. **Exec/Robot**: the number of operations done by the robots. Giving 5 meaning the robots will creates 5 cases or execute 5 tasks.
3. **Delay Start**: each robot can wait this delay before starting the operation
4. **Sleep**: when there is multiple execution, the robot will sleep this time.

For example, to if you specify 2 as a Robot, and 6 executions, the scenario will create 2\*6=12 cases. Then, you have 1 robot will execute 12 tasks. This task robot will start 1000 ms after, then cases have time to be created.

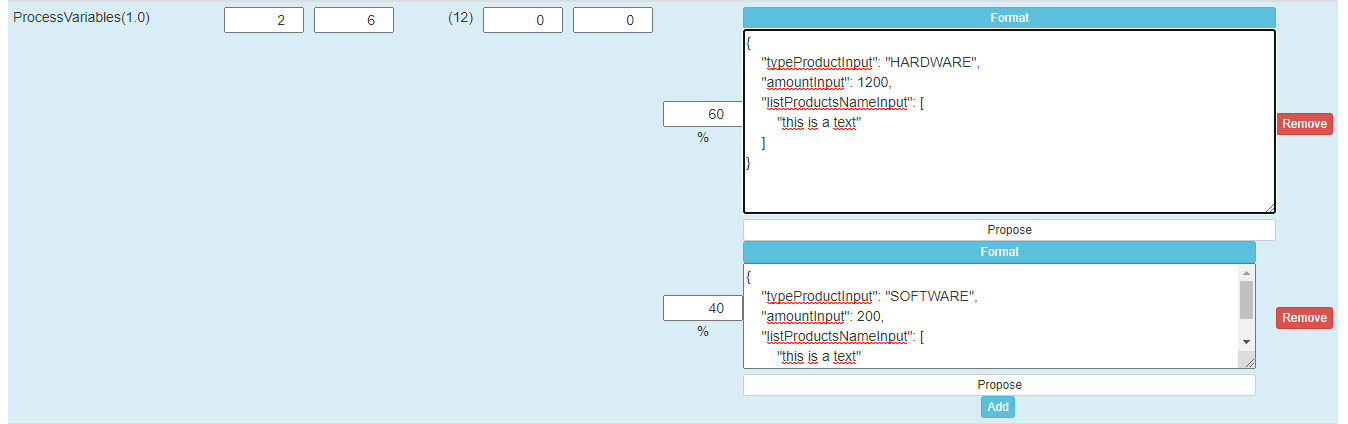


**Data**

To create a case, to execute a task a contract has to be fulfilled. To give this information, click on Add.

You can use “propose”. Meteor give you then the JSON expected by the creation / execution.

You can add multiple lines, to give a complete dataset. The percentage is used to specify the ratio for each data. If the percent is null, then the percentage is by default the average (50-50 for two lines for example).

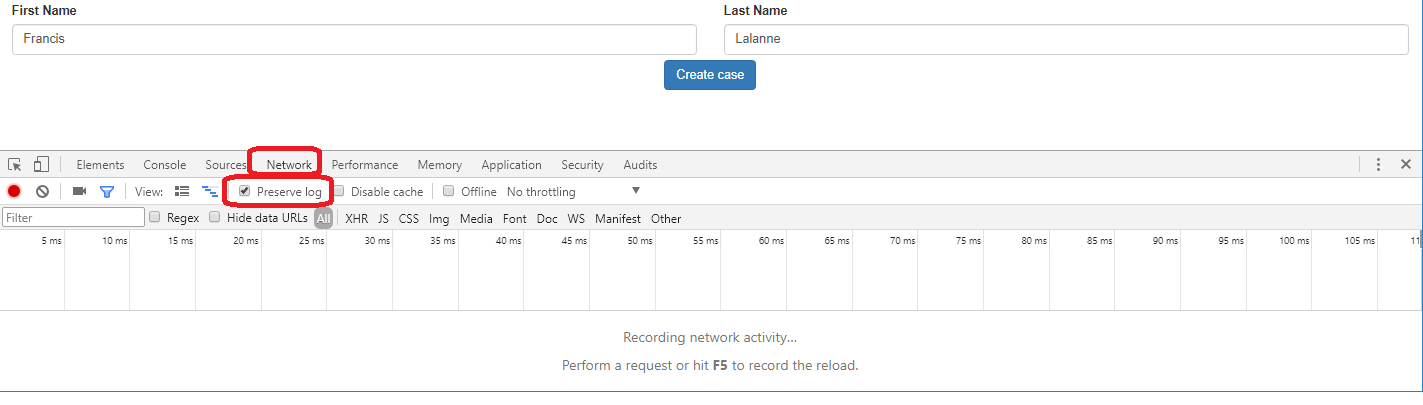


By this way, it’s easy and fast to generate a lot of cases with different value.

**Tips**

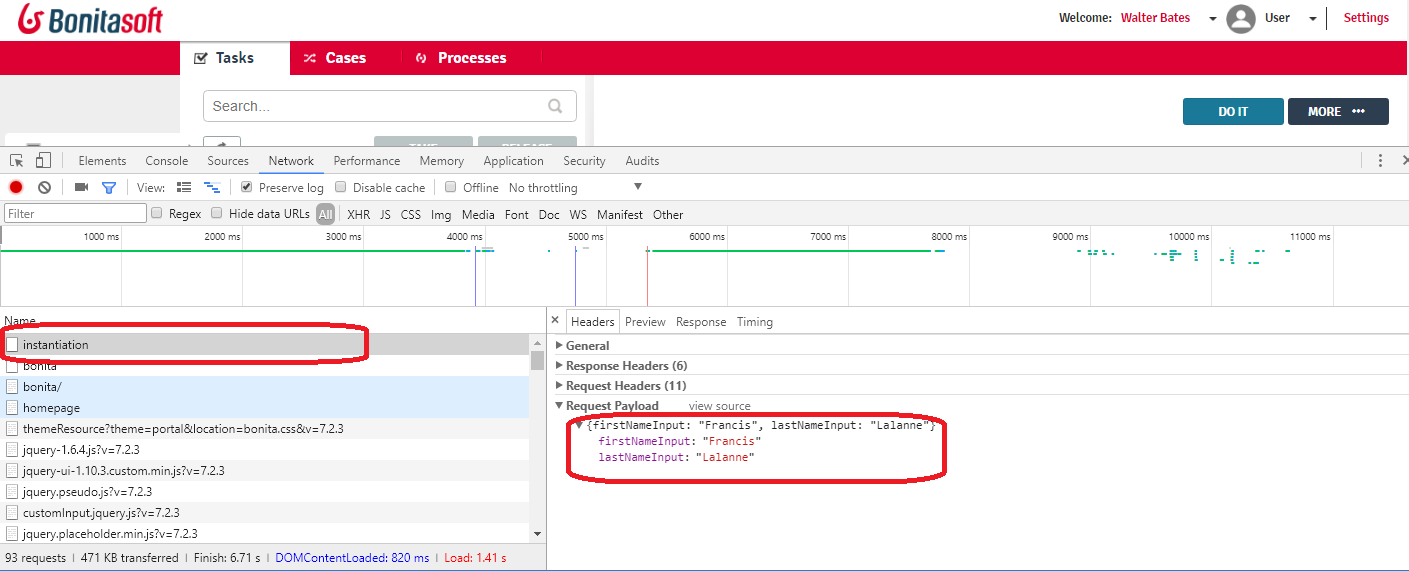
To figure out the input to give, you can get it from the browser.

In Chrome, access the form, and then click on F12. Then, select the Network. Click the check box “preserve log”.



Click on the button to create a case.

Then, search the Instantiation URL (should be the first)



The RequestPayload is the information you look for (click on “view source” to get the correct one): here it’s

{"firstNameInput":"Francis","lastNameInput":"Lalanne"}

# Execution

To do

# Unit Test

To do

# Load/Save

To do

## Load/Save

## Export / Import the scenario

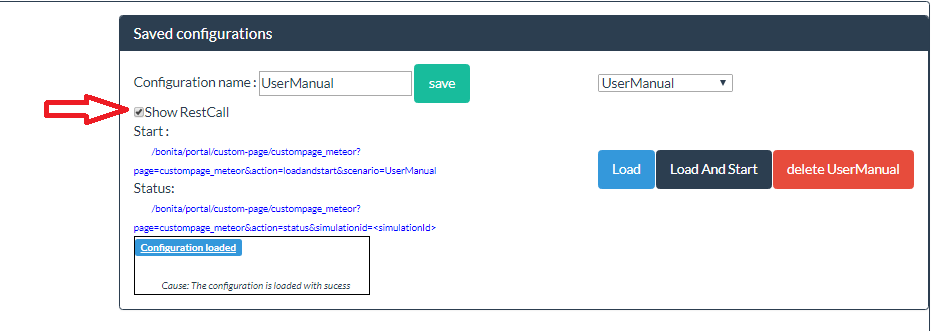
To do

# External execution

## By a REST CALL

Two REST API is available to run a RestCall. Select a configuration, and then click on “load”.

Click on the check Box “ShowRestCall” to access them



### Start

The RestAPI start is used to start a new test. The form is

<ServerBonita>/bonita/portal/custom-page/custompage\_meteor?page=custompage\_meteor&action=loadandstart&scenario=<Config name>

Nota: if the name is complexe and contains special character, you can use the JSON form (which is the URL use by the page) :

[http://localhost:8080/bonita/portal/custom-page/custompage\_meteor/?page=custompage\_meteor&action=loadandstart&paramjson={%22confname%22:%22<ConfigName>%22}](http://localhost:8080/bonita/portal/custom-page/custompage_meteor/?page=custompage_meteor&action=loadandstart&paramjson=%7b%22confname%22:%22%3cConfigName%3e%22%7d)

To create this URL, use this JS script

**var** param = { "confname": **this**.config.currentname};

**var** json = encodeURI( angular.toJson( param, **false**));

$http.get( '?page=custompage\_meteor&action=loadandstart&paramjson='+json )

The resultis a JSON information, which contains two main information:

* The status
* The simulationID

{

"timeStarted": "08/08/2017 11:08:33",

"robots": [

...

],

"config": {

....

},

"simulationid": "1502217803974",

"status": "STARTED"

}

You use the simulationId to get the status

### Status

To get the status of a simulation, use

<Server>/bonita/portal/custom-page/custompage\_meteor?page=custompage\_meteor&action=status&simulationid=<simulationid>

Result is a JSON loke

{

"timeEnded": "08/08/2017 12:08:54",

"timeStarted": "08/08/2017 12:08:33",

"robots": [

...

],

"listSimulations": [

...

],

"nbErrors": 0,

"status": "DONE"

}

The different status is:

* DONE: the test is finish
* NOSIMULATION: the simulation id given not exist
* STARTED: the simulation is in progress

When the simulation is running (state STARTED), you have some additional attributes

percentAdvance is the percentage of advancement(0 to 100)

## Scheduler (Truck milk)

# Use case

This chapter explore the different usage of the Meteor page, and the way you must configure the page and the scenario.

## CI environnement

## Certification usage

## Load a server

HEADQUARTERS

PARIS, FRANCE

76 boulevard de la République

92100 Boulogne-Billancourt

EMEA, ASIA & LATIN AMERICA

GRENOBLE, FRANCE

32, rue Gustave Eiffel  
38000 Grenoble

NORTH AMERICA

SAN FRANCISCO, USA

44 Tehama Street  
San Francisco, CA 94105

NEW YORK, USA

33 Nassau Avenue

Brooklyn NY 11222