Truckmilk Manual

Jobs executor

June 2019

Sommaire

[1. Introduction 4](#_Toc52267652)

[2. Usage 4](#_Toc52267653)

[3. Installation 4](#_Toc52267654)

[3.1. Get the resource 4](#_Toc52267655)

[3.2. Install the page 4](#_Toc52267656)

[3.3. Install the Quartz 4](#_Toc52267657)

[3.4. Force a complete installation 5](#_Toc52267658)

[4. De installation 6](#_Toc52267659)

[4.1. Stop the Quartz job 6](#_Toc52267660)

[4.2. Remove the page 6](#_Toc52267661)

[4.3. Advanced setup 6](#_Toc52267662)

[5. Use jobs 6](#_Toc52267663)

[5.1. Create a job 7](#_Toc52267664)

[5.2. Schedule 7](#_Toc52267665)

[5.3. Parameters 8](#_Toc52267666)

[5.3.1. Document Input parameter 9](#_Toc52267667)

[5.3.2. Test parameters 10](#_Toc52267668)

[5.4. Execute it once 10](#_Toc52267669)

[5.5. Activate / Deactivate 11](#_Toc52267670)

[5.6. Access the report 11](#_Toc52267671)

[6. Embeded plug in 12](#_Toc52267672)

[6.1. Delete case 12](#_Toc52267673)

[6.2. Email User tasks 12](#_Toc52267674)

[6.3. Ping 13](#_Toc52267675)

[6.4. Purge Archived Case 13](#_Toc52267676)

[6.5. Purge Archived Case: Get List 13](#_Toc52267677)

[6.6. Purge Archived Case: Purge from List 13](#_Toc52267678)

[6.7. Replay Failed Tasks 13](#_Toc52267679)

[6.8. SLA 14](#_Toc52267680)

[7. Maintenance 14](#_Toc52267681)

[7.1. Information 14](#_Toc52267682)

[8. Build a new plug in 15](#_Toc52267683)

[8.2. Clone the repository 15](#_Toc52267684)

[8.3. Compile, deploy locally 16](#_Toc52267685)

[8.4. Create a new Plug-in 16](#_Toc52267686)

[8.5. Use Bonita Events 17](#_Toc52267687)

[8.6. Check plug in environment 17](#_Toc52267688)

[8.7. Check Job environment 17](#_Toc52267689)

[8.8. Definition description 18](#_Toc52267690)

[8.9. Using parameters 18](#_Toc52267691)

[8.9.3. Basic type 19](#_Toc52267692)

[8.9.4. Complex type 19](#_Toc52267693)

[8.9.5. Documents type 19](#_Toc52267694)

[8.9.6. Bonita type 19](#_Toc52267695)

[8.9.7. Special usages 19](#_Toc52267696)

[8.10. Conditions with parameters 20](#_Toc52267697)

[8.11. Execution 20](#_Toc52267698)

[8.12. Measures 20](#_Toc52267699)

[8.13. Chronometers 21](#_Toc52267700)

[8.14. Advancement 21](#_Toc52267701)

[8.15. Stop mechanism 22](#_Toc52267702)

[8.16. Document management 22](#_Toc52267703)

[8.17. State of the art 23](#_Toc52267704)

Content

# Introduction

This document explains how to use the TruckMilk Page.

# Usage

The page may be use in different context:

# Installation

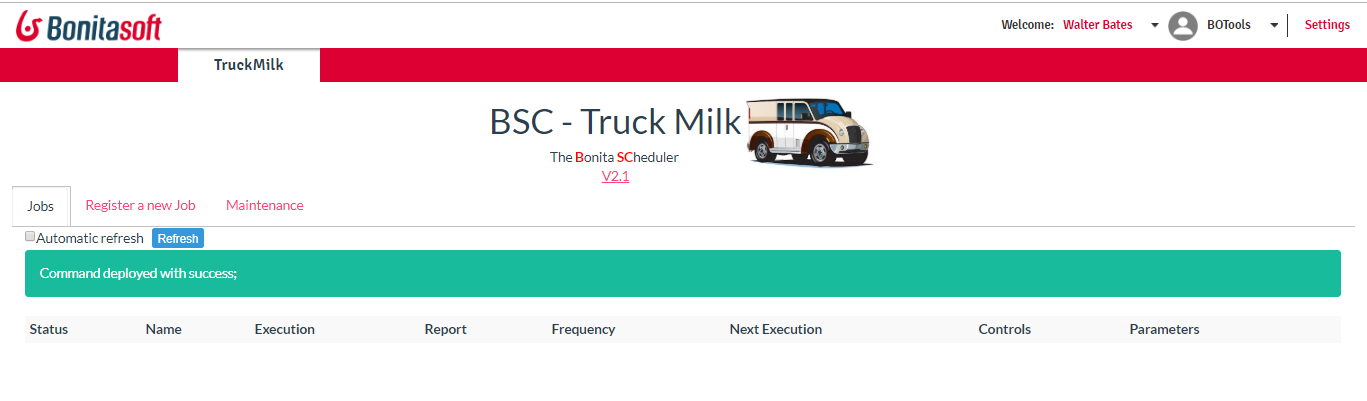
## Get the resource

Download the page from the community,

## Install the page

Then install the page as a Resource and reference it in a Profile or Application.

Access the page.



The first access, you should see a banner to explain the command is deployed with success

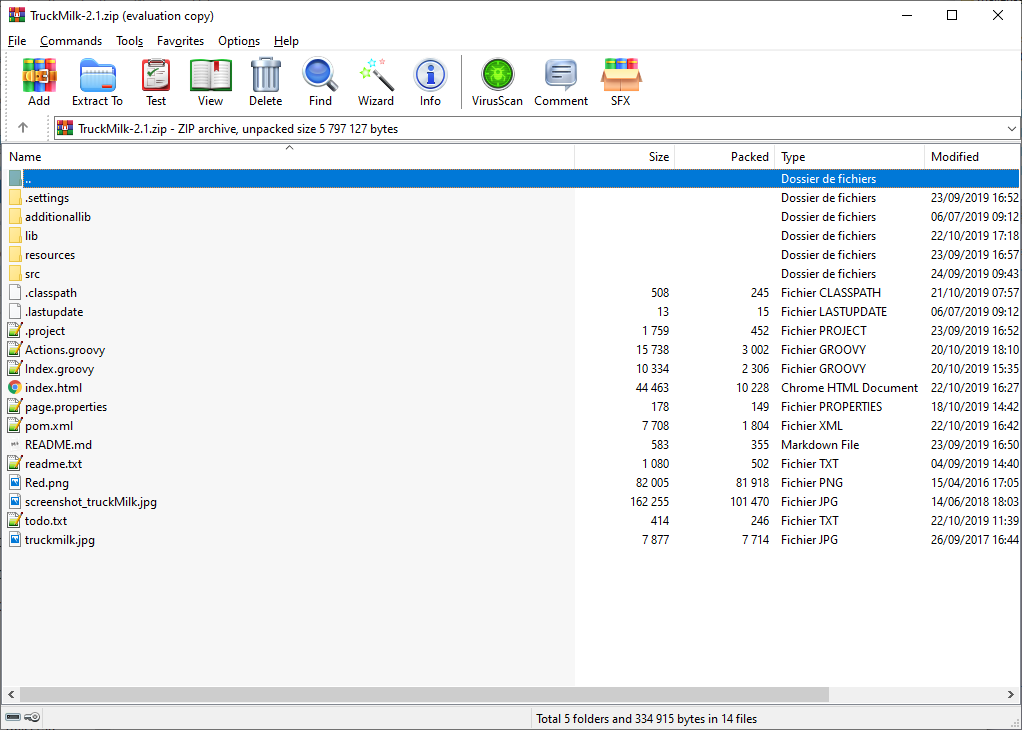
A Bonita Command is a JAVA Class deployed on the server. This allow the controller to check and start new jobs even if user is disconnected.

On a Cluster environment, each node will install the command. Command is saved in the database, and is backup when you back up the Bonita database

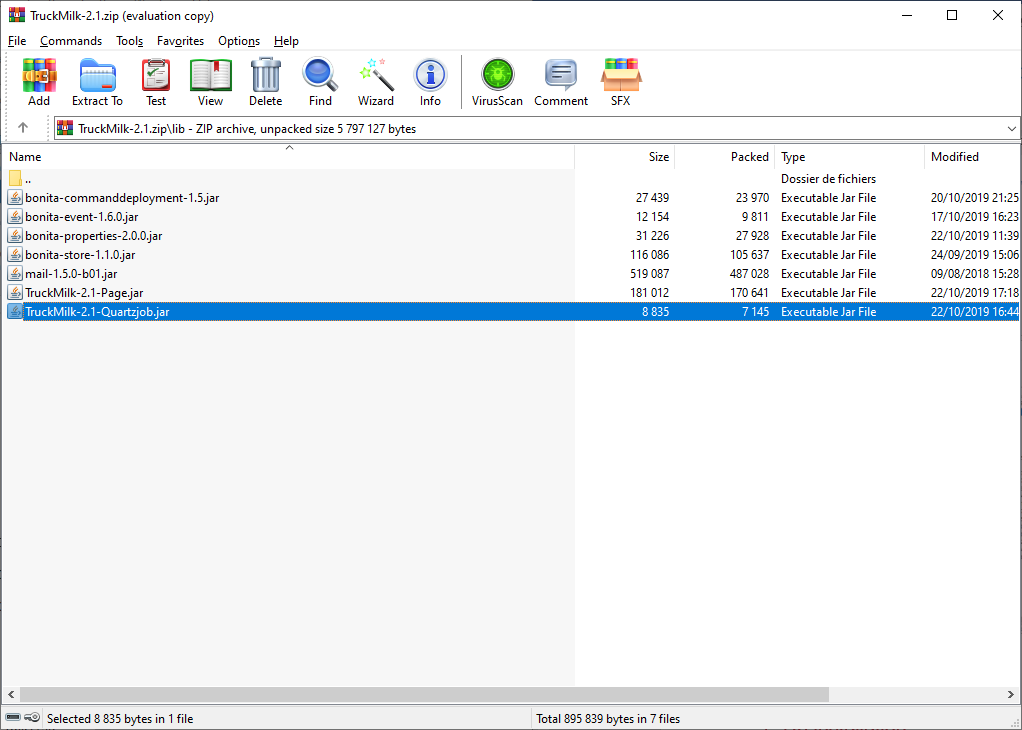
The second step consist to install the Quartz Job JAR file.

## Install the Quartz

Unzip the Truck milk ZIP file.



Access the lib directory



Copy the jar name “TruckMilk<version>-Quartzjob.jar” under the path

<BONITASERVER>/server/webapps/bonita/WEB-INF/lib

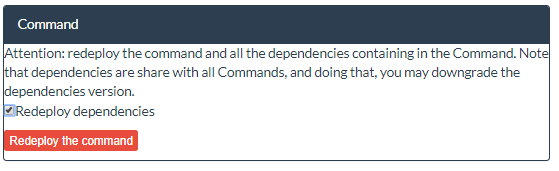
* You can copy it on your Bonita Studio: path is <STUDIO>/workspace/tomcat/server/webapps/bonita/WEB-INF/lib
* On a Cluster, you have to copy this file on each node.

Restart your server (on a studio, go to Server / Restart Web Server)

Then, access the page. Go to the tab “Maintenance” and click on START

## Force a complete installation

Dependencies are shared between command. If you need to be sure to deploy dependencies associated to the command, the Command Deployment, in the tab Maintenance, can be used



# De installation

To desinstall the page:

## Stop the Quartz job

Go to the page, tab Maintenance, and select “Stop”.

The quartz job is stopped.

## Remove the page

You can then remove the page in the Administration / resource

## Advanced setup

Nota: to remove completely the page, On the maintenance tab, use the undeploy button. This function

* Remove the TruckMilk command,
* Remove all Quartz job

To finish, remove the Quartz file

Go to

<BONITASERVER>/server/webapps/bonita/WEB-INF/lib

And remove the file “TruckMilk<version>-Quartzjob.jar”. You should need first to stop the server. On a cluster, this operation has to be done on each node.

You can check the Quartz Table in the server to verify the jobs is correctly removed in tables qtrz\_cron\_triggers, qtrz\_triggers, qtrz\_job\_details;

 Select \* from qrtz\_cron\_triggers where trigger\_name='trgMilktruckJob\_1';

 Select \* from qrtz\_triggers where trigger\_name = 'trgMilktruckJob\_1';

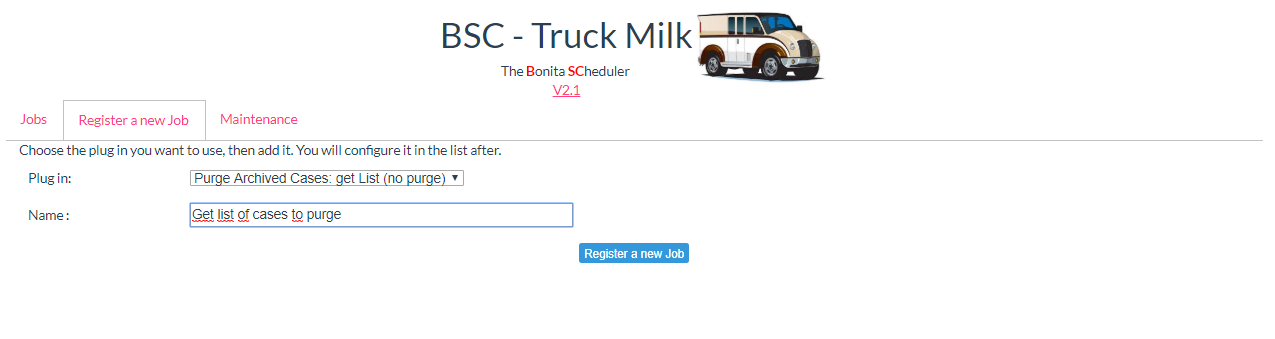
 Select \* from qrtz\_job\_details where job\_name = 'trgMilktruckJob\_1';

# Use jobs

Truck Milk execute jobs. This chapter explain how to create a job, then access all parameters

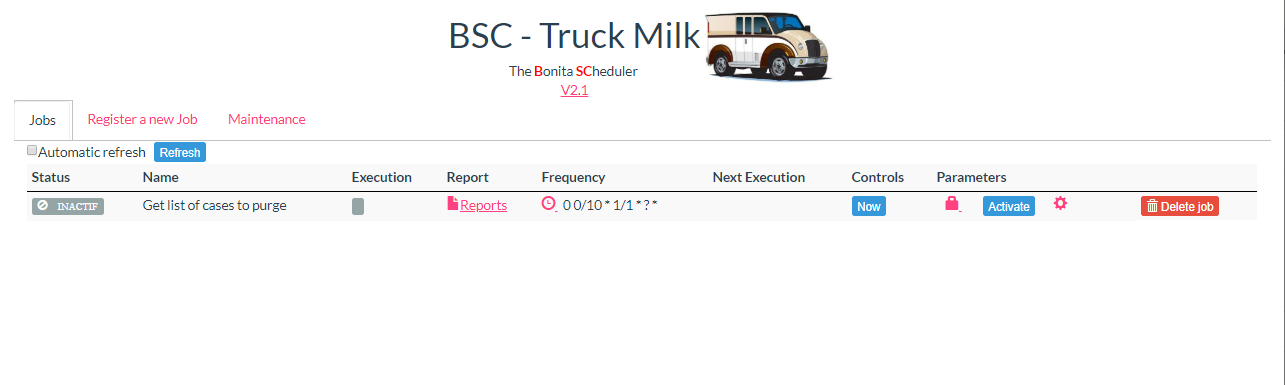
## Create a job

Click on the tab Register a new Job



Choose the Plug In in the list of existing Plug in. Give then a name. The name must be unique (it’s not possible to register two jobs with the same name).

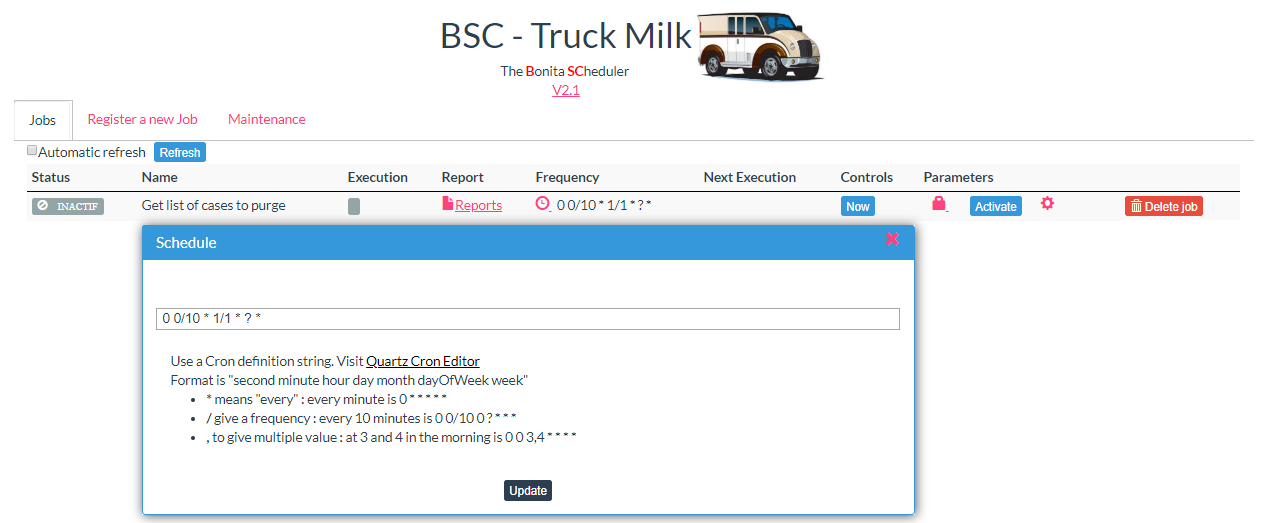
Access the Jobs tab: job is visible in the list. By default, jobs are deactivated.



## Schedule

The scheduler part specify the frequency of execution. Visit <https://www.freeformatter.com/cron-expression-generator-quartz.html> to help you to calculate the frequency.

Click on the icone  and setup the value



Click on Update to validate the change.

Note 1: the default value, 0 0/10 \* 1/1 \* ? \*, means:

At second: 00, every 10 minutes starting at minute: 00, every hour, every day starting on the 1st, every month

Note 2: Frequency apply only when the job is activated.

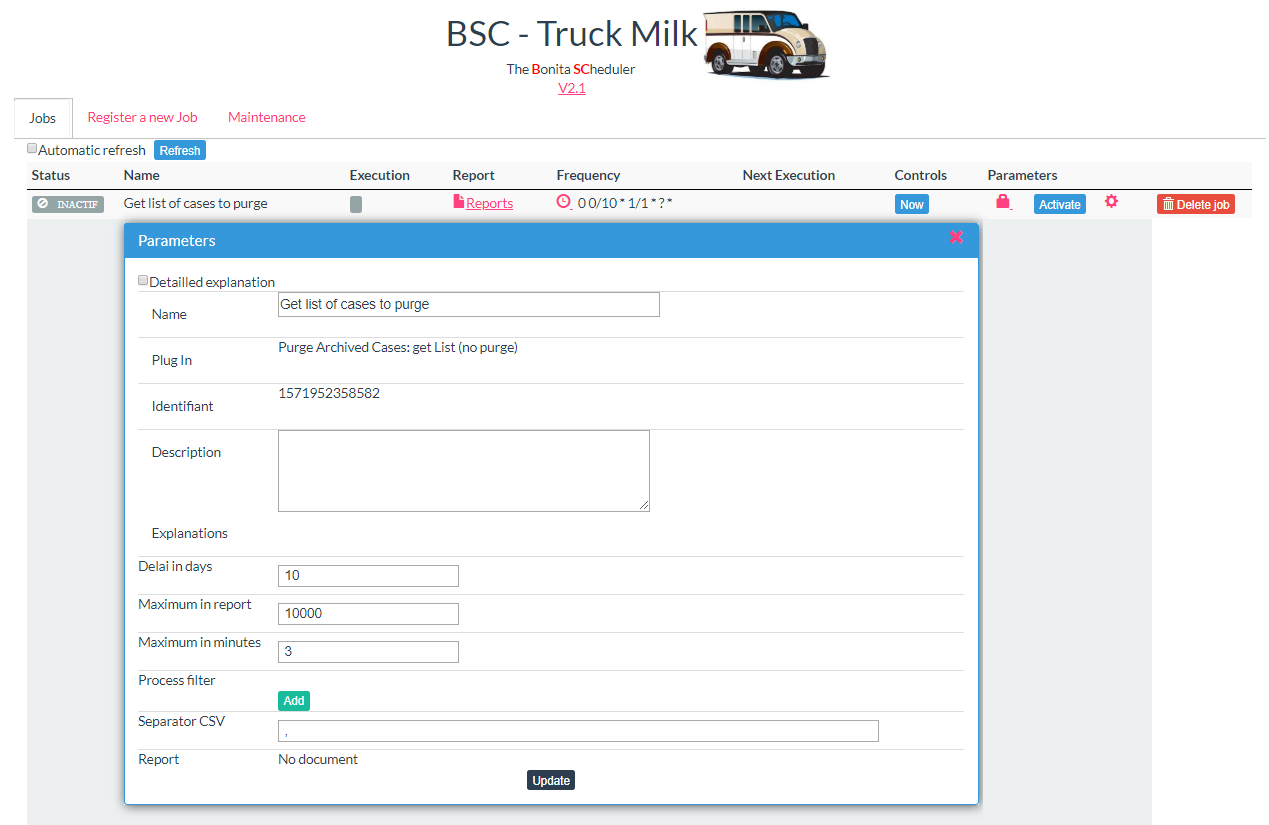
## Parameters

Jobs executing a Plug in. Each Plug in has parameters.

Each plug in has main parameters: name, description.

The identifiant is generated by Truck milk when you created the jobs and is usable on log file.

Click on  and gives the value

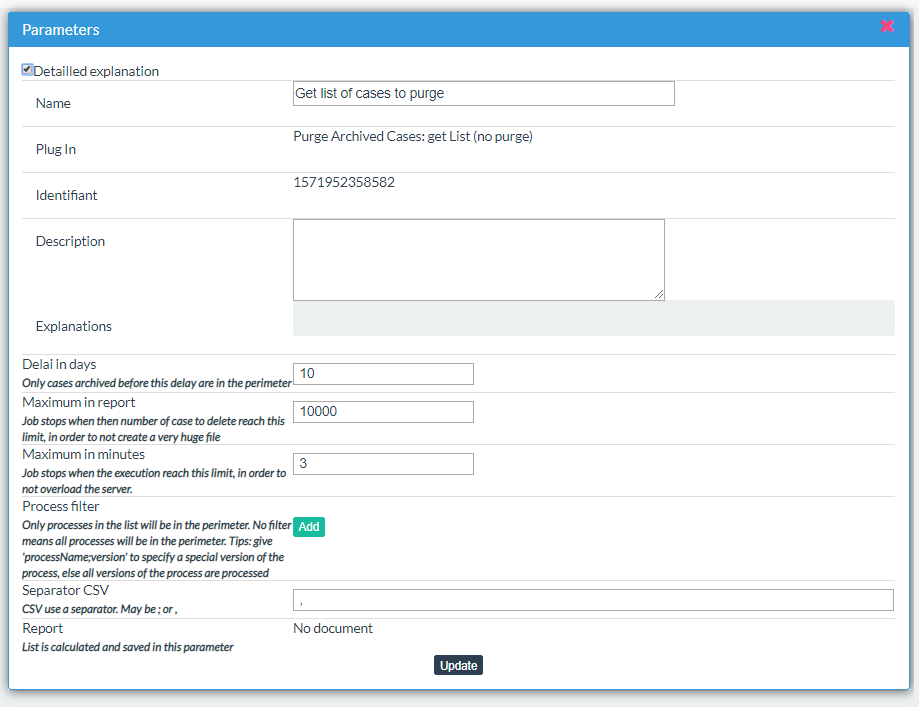


Different parameters exist:

* Integer,
* List of processes
* List of data
* Text

Click on Update to validate the change.

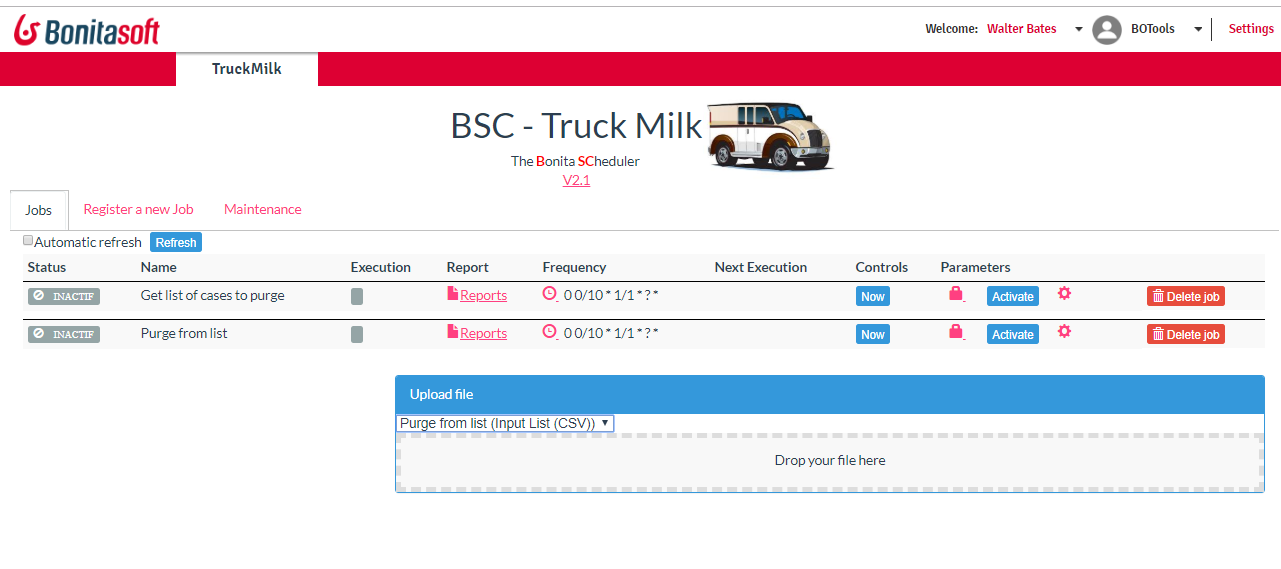
You can check the “Detailed explanation” to have more information parameter per parameter.



### Document Input parameter

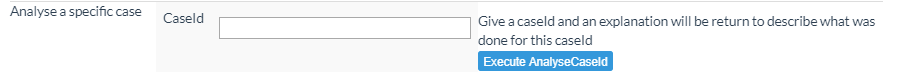
A plug in may want a document as a parameter (Plug in “Purge Archived Case: Purge from list” for example)

When a job with this kind of parameters is detected, a new box appears. Select then the job and parameter and drag and drop the document to upload it on server.



### Test parameters

In some Plug in, you may have some tools to help you. Then, a button is visible.

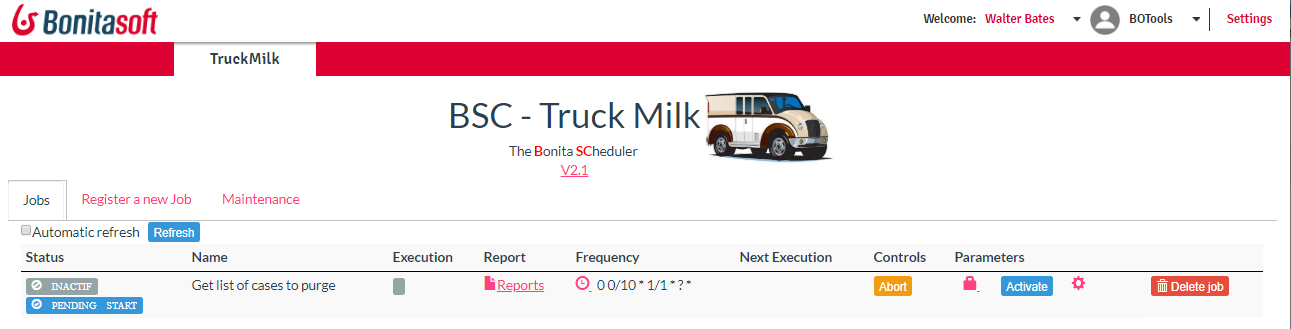


## Execute it once

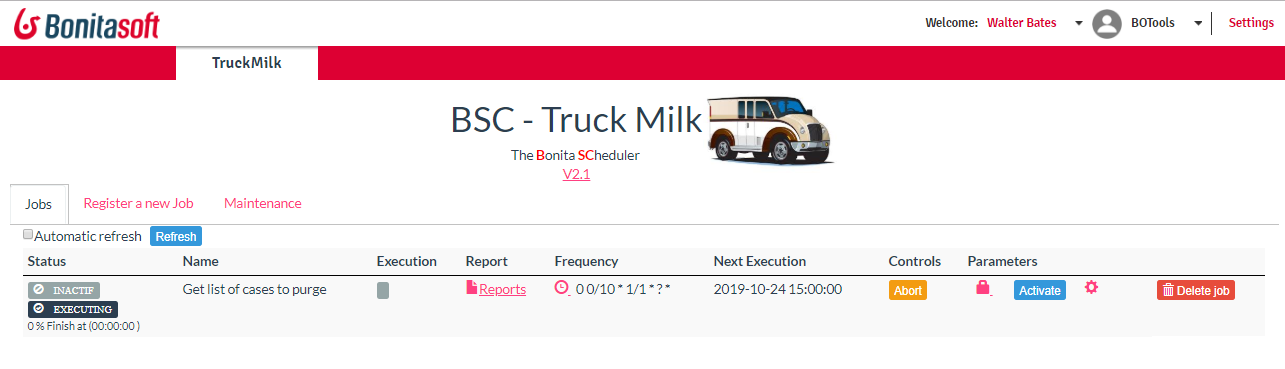
To verify parameters or have an immediate start, you can click on the button “Now”.

Jobs is then registered to be executed.

Status is moved to “Pending Start”.

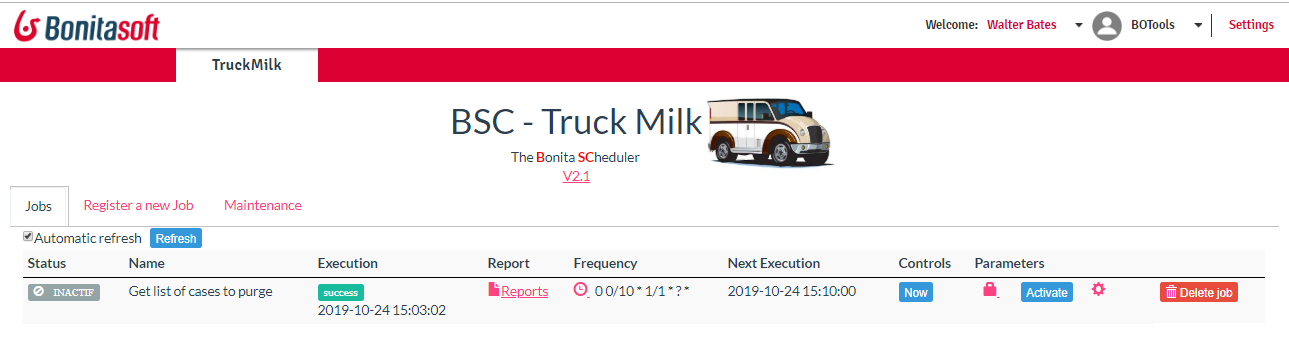


Then, job status change to “Executing”. An estimation time to finish is calculated to end when it is possible (mainly for plug in who may need time to execute).



“Abort” button may be use at any time to ask the job to finish immediately. Each Plug in test regurlarely the status, and stop during the process, at a safe point (jobs is not killed).

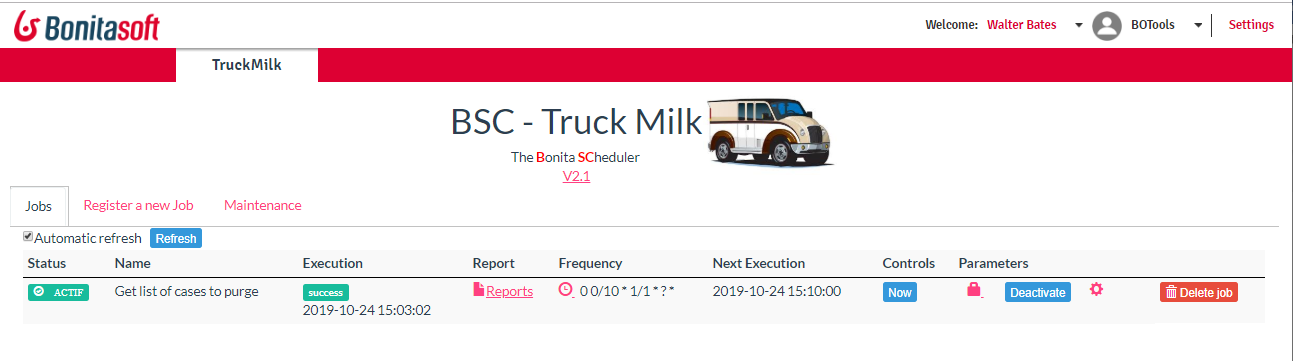
When execution finish, status change to “Success”.



Note: status may be SuccessNothing. That indicate the job runs but have nothing to do. If you asked for the list of cases to purge, and the list is empty, then the plug In return a “Sucessnothing” status. Idea is to keep then only the real execution.

## Activate / Deactivate

Activate a job by clicking on the Activate button. Truck Mil calculate the next Execution date.

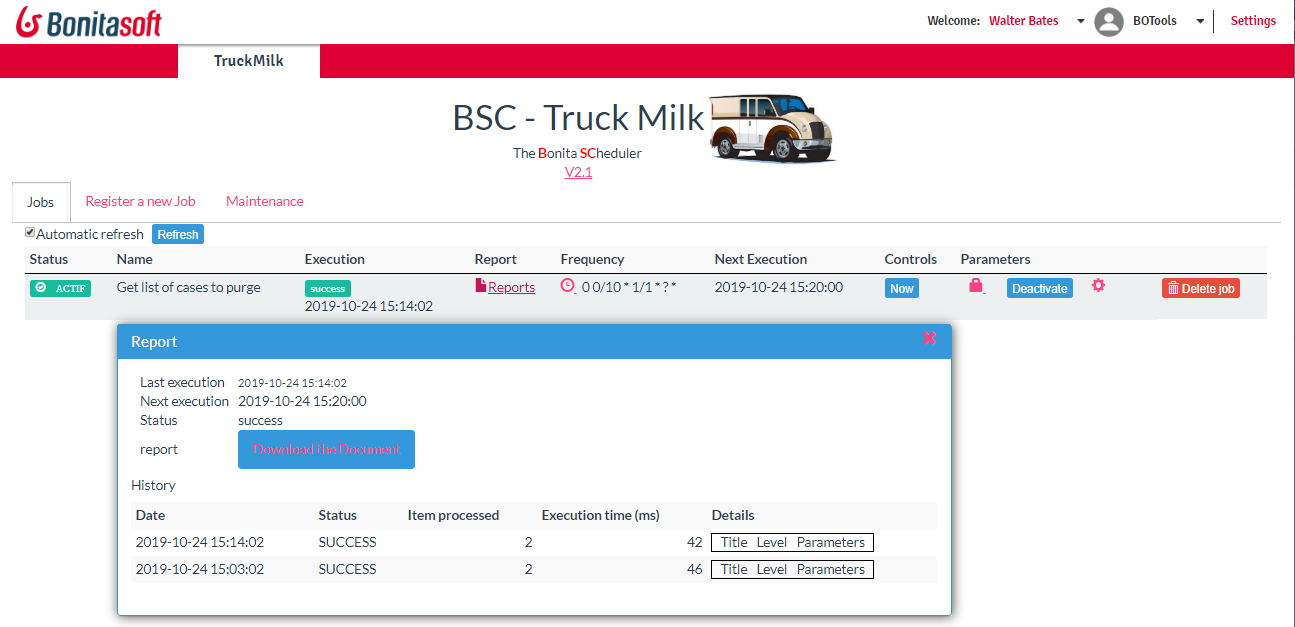


At any time, you can deactivate a job.

## Access the report

Report are accessible for each job by clicking on .

All executions are visible, with a status.



Note 1: when the Plug-in produce a document, you can download it. Only the last generated document is saved.

Note 2: the history keeps only the Success and Error execution. If the Plug-in return a SuccessNothing, it is not saved in the list, to keep history with real information.

Note 3: Only the last ten executions are kept.

# Embeded plug in

## Delete case

The Plug-in delete cases, archived or not. A list of processes can be set as parameters, else all processes. Use this plug in when its necessary to purge a lot of cases.

Deletion of the case may not work due to a transaction timeout: to delete a process, Bonita Engine open a transaction, then delete all cases. When there are too much case to remove, transaction may fail. Secondly, this operation may need a lot of time to be finished.

The parameter “Maximum cases deletion” is used to specify the maximum number of case to delete. When the number is reach, the job finish, it will re-start at the next execution, so cases may be removed “page per page” to avoid any overloading on the server.

## Email User tasks

This Plug in calculated, for a user, all visible tasks, i.e., all tasks visibles in the portal, in “my tasks”.

Then, one (and only one) email is send to the user, with the list of all tasks.

Plug in required a User Profile as parameters: all users registered in the profile will receive an email if they have active tasks.

The content of email may be defined, using place holder:

* Assignedtask list all tasks assigned to the user
* Pendingtasks contains all tasks assigned (except the assigned tasks)

For example, the content may be

Your assigned task <br>{{assignedtasks}}<p>Your pending tasks:<br>{{pendingtasks}}

Nota: this plug in need the Email JAR file installed. Check tab Maintenance / Information to verify that the JAR is correctly installed.

## Ping

Ping Plug In is here for test usage

## Purge Archived Case

This Plug-in purge archived case. A delay may be set in parameter, to purge case with three months delay for example.

## Purge Archived Case: Get List

This Plug In calculated a list of cases to purge, but not purge them. The list is a CSV file, containing the caseid, the process name. A status column is ready to be fulfilled.

A delay can be set in parameter, and a list of processes.

Example of CSV:

caseid;processname;processversion;archiveddate;status

5149;ExpenseNote;1.0;18/10/2019 15:02;

5150; ExpenseNote;1.0;18/10/2019 15:02;

5151;VacationRequest;1.0;18/10/2019 15:02;

## Purge Archived Case: Purge from List

This Plug In works with the previous plug In. it accepts the same list, and checl the status of each line. If the status is DELETE, then the case is removed.

Example of CSV to upload:

caseid;processname;processversion;archiveddate;status

5149;ExpenseNote;1.0;18/10/2019 15:02;DELETE

5150; ExpenseNote;1.0;18/10/2019 15:02;

5151;VacationRequest;1.0;18/10/2019 15:02;DELETE

## Replay Failed Tasks

Replay Fail task re-execute all tasks in a Failed mode. The number of retry, and the delay between two tentative are parameters.

Note: the Bonita Engine must accept to replay a failed task. Only the Entreprise or Performance subscription allow that.

## SLA

Tasks must have some due date to be executed (example, execute the task in 10 days). Before the delay is reach, different actions may be necessary:

* Send a reminder after 6 days (i.e. at 60% of the delay) to all users who can execute the task
* Send a second reminder after 9 days (90%)
* Send a alert reminder if the delay is over than 1 days (110%)
* Assign the task to a different person if the delay is over 13 days (130%)

A design in the process is possible to reach these operations, based on “non interruptible boundary event”. When process have a lot of human tasks, and rules are important, the design may become very complex. Secondly, if a rule change (add a new trigger, change the trigger at 90 to 80%), the process must be redeployed.

This Plug In does theses operations and can be update at any time.

The SLA operation is based:



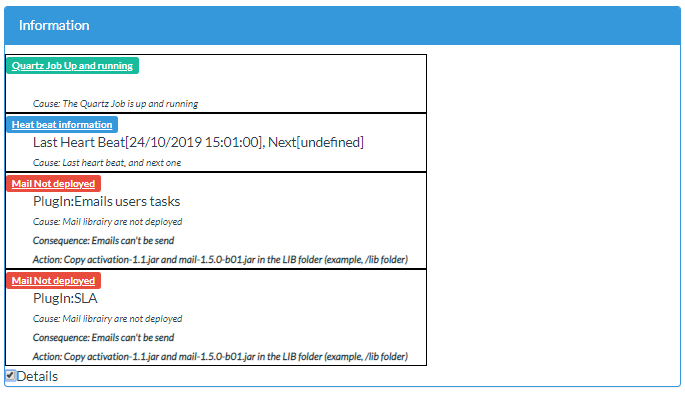
* A task’s name. If this parameter is empty, the rule is applied on all tasks
* A Percent Threshold. This parameter is active only if the task has a Due Date. Then, the percentage is calculated between the Start Date and the Due Date. When the due date is reach, the percent is 100%. It’s possible then to apply a rule over 100% (due date expired)
* Action: multiple actions are possible. Click on detail Explanation to have a list of explanation. Main actions are EMAILUSER, EMAILACTOR, EMAILCANDIDATE,ASSIGNUSER, ASSIGNSUPERVISOR

# Maintenance

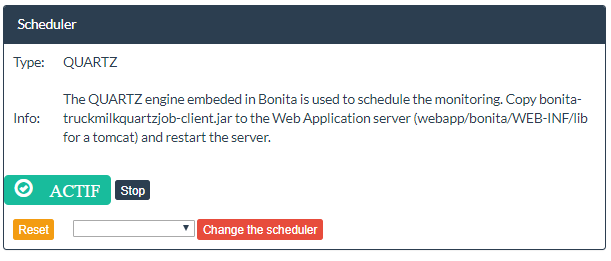
The Maintenance tab group some operations:

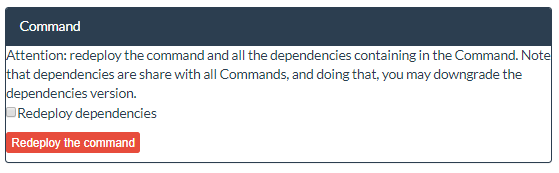
## Information

This panel contains main information on the Scheduler.









# Build a new plug in

This part explains how to develop a new Plug In. Don’t hesitate to create and send to the community any new plug in. This plug in will be available then on any new release of the page and is maintained. Please note the plug in must be general (don’t hard code any process name, any information, use parameters).

## Clone the repository

Clone the github repository

<https://github.com/Bonitasoft-Community/page_truckmilk>

You can import the process under Eclipse.

## Compile, deploy locally

To compile the page, just execute

mvn install

this order

* Download all needed component from Maven
* Compile the JAVA and generate a JAVA library
* Generate the page resource (a ZIP file, containing all libraries, HTML, resources)
* Deploy the page locally on your server, using localhost:8080 and walter.Bates to realize the job (if you need to deploy on a different server, with a different user name, modify the pom.xml)
* Create a profile named BOTools, and reference the page in this profile

To test the page, click again on the page in the menu bar: Bonita reload the page then.

## Create a new Plug-in

A plug in is a JAVA classes. Then user can create a new job from your plug in.

1. By convention, create the class under org.bonitasoft.truckmilk.plugin.xxx Start it by “Milk…. This class should extend the class MilkPlugIn.

The convention is to have a sub package per CATEGORY. The category explains on which component the job works: it may be tasks, cases, monitor...

You can set the type to TYPE\_PLUGIN.EMBEDED : that's mean this plug in is delivered with the Truckmilk page.

1. Register your plug in in MilkPlugInFactory.collectListPlugIn()

Then, the factory knows your plugin and can propose it to the users

1. Define the getDefinitionDescription()

This method returns the definition of your plug in parameters, category, measures.

1. Define the execute() method

Now you are! this is the main part of your plug in, the execution.

At input, Truckmilk gives you the value of all parameters the user gives. This is the value for parameters defined in the getDefinitionDescription().

At the output, you must produce a status (SUCCESS, ERROR or else), reports and measures.

1. Additional methods: checkPluginEnvironment() and checkJobEnvironment()

If you want, you can implement this method. The checkPluginEnvironment() is called at the beginning, first time your plugin is registered.. You may want to test if all additional information is set. For example, you may need an external JAR file: it is present?

checkJobEnvironment() is different. It depends of the job definition, and it is executed before each execution. For example, you ask as a parameter a

DataSource to connect to an external database. Is this database can be accessed before the execution

## Use Bonita Events

TruckMilk use the BonitaEvent library to display event (which may be information or error)

It's very important that the user describe the error, explain it, and give an action plan. Instead to give a SQL ERROR to the user, developers has to explain what's happen, what is the consequence, and what is the action to do.

For each event, you must specify:

* a package and a number. Then, the event is unique and can be identified immediately by the administrator and the developer to figure out what's going on
* a level (INFO, ERROR, WARNING, CRITICAL)
* a title to display to the user. Example "Bad SQL Request"
* a cause: it's mandatory for an error. The SQL Request failed: what is the cause? Can't connect to the database? Bad syntax? Missing parameters?
* a consequence: it's mandatory for an error: explain to the user the consequence of this error. Ok, the SQL can’t be executed, so we have no history
* for example, but the treatment can continue.
* an action: what the user has to do? Database is done, then action is to contact the administrator of the database behind the source.

Then, you can create a new event, referencing an existing event. Doing that, you can add parameters.

**private** **static** BEvent *eventDeletionFailed* = **new** BEvent(MilkMoveArchive.**class**.getName(),

2,

Level.***ERROR***,

"Error during deletion",

"An error arrived during the deletion of archived cases",

"Cases are not deleted",

"Check the exception");

milkJobOutput.addEvent(**new** BEvent(*eventDeletionFailed*, e, "ListArchiveId: " + listArchivedProcessInstances.toString() + " " + e.getMessage()));

## Check plug in environment

Plug in can check its environment, to detect if you missed something. An external component may

* be required and are not installed.
* This call is performed when users click on "getStatus" in the maintenance panel, and before each execution.

**@return** a list of Events.

## Check Job environment

Check the Job's environment. This verification is executed before each execution, and all execution parameters is given at input. So, the verification is done with the real input.

For example, your plug-in asks a Data source as parameters. So, you can check in this method that you can connect to the database.

## Definition description

return the description your plug in. The description contains:

* a name (which must be unique), explanation and label

milkPlugInDescription.setName("Ping");

milkPlugInDescription.setExplanation("Just do a ping");

milkPlugInDescription.setLabel("Ping job");

* a category

Choose between the existing category

milkPlugInDescription.setCategory(CATEGORY.***OTHER***);

* the way to ask to stop the job. Users can decide to limit the job execution in time, or in number of items processed. Of course, if you describe you can stop in time, you have to implement it in your code.

milkPlugInDescription.setStopJob(JOBSTOPPER.***MAXMINUTES***);

* the list of parameters

milkPlugInDescription.addParameter(***cstParamAddDate***);

milkPlugInDescription.addParameter(***cstParamTimeExecution***);

* the list of measures

milkPlugInDescription.addMesure(***cstMesureMS***);

milkPlugInDescription.addMesure(***cstMesureHourOfDay***);

## Using parameters

A plug-in need parameter in general. To delete cases, the list of processes, or a delay to collect only cases older than a delay for example.

You register parameters in the getDefinitionDescription(), and you use it in the checkJobEnvironment() or in executeJob().

You have different kind of parameters. Then the parameters are visible in the Parameters page, when users access this section.

You define a parameters with the createInstance() method.

**private** **final** **static** PlugInParameter ***cstParamAddDate*** = PlugInParameter.*createInstance*("addDate", "Add a date", TypeParameter.***BOOLEAN***, **true**, "If set, the date of execution is added in the status of execution");

Different parameters are:

* Th name of the parameter (should be unique in your plug in).
* The label user will see.
* The type. See below to have a information on each type
* The defaut value. The default value depends of the type (must be a Boolean is you choose a type parameter BOOLEAN)
* Then, a complete information. When user clicks on “show information”, he will see that.

You have some shortcut, like

* createInstanceDelay(),
* createInstanceArrayMap(),
* createInstanceButton,
* createInstanceFile,
* createInstanceListValues,
* createInstanceInformation

Then, you access the value of parameters via the getInput….Parameters method

Boolean addDate = milkJobExecution.getInputBooleanParameter( ***cstParamAddDate*** );

Here the different type parameters

### Basic type

In this category, A STRING is a simple input, and a TEXT a Text area, a LONG a number, a BOOLEAN a check box

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | |
| STRING | | A simple input |
| TEXT | | A Test Area input |
| LONG | | An input, type number (only number can be gives) |
| BOOLEAN | | A Checkbox |

### Complex type

These parameters handle complex display

|  |  |
| --- | --- |
| **Type** | **Description** |
| DELAY | User give a delay. A selection allows the user to give a type (Minutes, Hours, Days, Month) and a second input the value. Then user can give “4 days” or “10 hours”.  As a developer, you give a date, and a direction, and you get back the calculated day. Example, you give “February 1, 10:00” + “advance”, for a 4 days delay, you’ll receive “February 5 10:00”. |
| LISTVALUES | A selection box |
| ARRAY | A List of String. User can add / remove items |
| ARRAYMAP | A list of Records. Record is defined as a list of “ColDefinition”.  Plugin get a List of Map : List<Map<String,Object>> |
| JSON | An input text, where the user is supposed to give a JSON string. The JSON is verified. |

Example of ARRAYMAP definition

PlugInParameter *cstParamRuleSLA* = PlugInParameter.*createInstanceArrayMap*("RuleSLA", "Rules SLA",

Arrays.*asList*(

ColDefinition.*getInstance*("TASKNAME", "TaskName", "Task name in a process. Multiple tasks may be reference one time using # like 'Validate#Review#Control", TypeParameter.***STRING***, 50),

ColDefinition.*getInstance*("PERCENT", "Percent Threashold", "0%: task creation, 100%=due date", TypeParameter.***LONG***, 20),

ColDefinition.*getInstance*("ACTION", "Action", ACTION.***EMAILUSER***.toString() + ":<userName>, "

+ ACTION.***EMAILCANDIDATES***.toString() + ", "

+ ACTION.***EMAILACTOR***.toString() + ":<actor>,"

+ ACTION.***ASSIGNUSER***.toString() + ":<userName>, "

+ ACTION.***ASSIGNSUPERVISOR***.toString() + ":[1], "

+ ACTION.***STARTPROCESS***.toString() + ":<processName (<processVersion)>;<JSONinput>,"

+ ACTION.***SENDMESSAGE***.toString() + ":<messageName>;<targetProcess (<processversion>);<targetFlowNode>;<JSONMessageContent>;<JSONMessageCorrelation>",

TypeParameter.***TEXT***, 50)),

**null**, "Give a list of rules. Each rule describes the threshold in percent 0-task start, 100% due date, and action");

### Documents type

Manipulate document, as input or at output.

|  |  |
| --- | --- |
| **Type** | **Description** |
| FILEREAD | User upload a document, and Plugin can read it. |
| FILEWRITE | Plugin write this document, and it will be available for the user as download |
| FILEREADWRITE | User can upload a document, Plugin can read, and write a new version |

### Bonita type

These types display some Bonita Artifact

|  |  |
| --- | --- |
| **Type** | **Description** |
| USERNAME | Autocomplete to select a user between all users in the server |
| PROCESSNAME | Autocomplete to select a process in all processes (ENABLE or DISABLED) in the server |
| ARRAYPROCESSNAME | Autocomplete to select one or multiple processes |

### Special usages

|  |  |
| --- | --- |
| **Type** | **Description** |
| BUTTONARGS | Use to execute a local simulation |
| SEPARATOR | Separators, when you have a lot of parameters |
| INFORMATION | Display information |

## Conditions with parameters

Some conditions can be added on parameters

For example, you can use

**private** **static** PlugInParameter *cstParamProcessFilter* = PlugInParameter.*createInstance*("processfilter", "Filter on process", TypeParameter.***ARRAYPROCESSNAME***, **null**, "Job manage only process which mach the filter. If no filter is given, all processes are inspected")

.withMandatory(false)

.withFilterProcess(FilterProcess.***ALL***);

withMandatory( boolean isMandatory) :

|  |  |
| --- | --- |
| **Type** | **Description** |
| withMandatory( boolean isMandatory) | Field is mandatory. |
| withVisibleCondition( String visibleCondition ) | Give a condition. Attention, condition is in JavaScript |
| withVisibleConditionParameterValueDiff( PlugInParameter parameter, Object value ) | Give a condition based on a value of an different attribute. |
| withVisibleConditionParameterValueEqual( PlugInParameter parameter, Object value ) |  |
| withFilterProcess(FilterProcess filterOnProcess)  enum FilterProcess { ALL, ONLYENABLED, ONLYDISABLED } |  |

withVisibleConditionParameterValueDiff( PlugInParameter parameter, Object value )

Example, you have a BOOLEAN parameters, and you want to show this condition only if the user click the checkbox

Then use

## Execution

\* You get the value on each parameters, and you have to produce a result.

\* --

\* ------------------- Input are accessible via the jobExecution object.

\* - all parameters are accessible

\* - APIAccessor are accessible too.

\* --

\* ------------------- Output

\* You have multiple mechanism.

\* - executionStatus: the status has to be give to milkJobOutput.executionStatus.

\* - number of item processed : use setNbItemsProcessed() to give the number of item processed.

\* - listEvents: the detail of execution is provided by a list of Events. Use milkJobOutput.addEvent()

\* - Measure : you can add all measure in the report too, else they are in the measure table (user need to access it separately)

\* - Chronometers: add the chronometers informations.

\* - addReportInHtml() to add a HTML sentence

\* - addReportTableBegin() / addReportTableLine() / addReportTableEnd() : give data to safe it in HTML. Table has a protection to not keep a big amount of

\* data

\* - Document output

## Measures

\* A measure is a number that you want to calculate and return.

\* Measure is saved in a different table, and you can save only the two last execution, but keep 1000 measure, to display a graph.

\* Two measure are automatically saved : the number of items processed and the time of execution.

\* First, you have to declare the measure in the Description :

\* PlugInMeasurement cstMesureMS = PlugInMeasurement.createInstance( <Code>, <Label>, <Explanation>);

\* > > milkPlugInDescription.addMesure( cstMesureMS );

\* then in the execution, you can set a value to the measure

\* > > milkJobOutput.setMeasure( cstMesureMS, <value>);

\* If you want to add all measure in the report, use

\* > > milkJobOutput.addMeasuresInReport(boolean keepMesureValueNotDefine, boolean withEmbededMeasure);

\* > > > > keepMesureValueNotDefine if true, a measure not defined in the execution is added in the report, with the value 0

\* > > > > withEmbededMeasure Some measure are embedded : number of items processed and time to execute. They can be added in the report.

\* --

## Chronometers

\* For a long execution time, you want to register the time to execute a piece of code. Then, theses value can be added in the final report

\* To Start a chronometer, use

\* > > Chronometer sleepTimeMarker = milkJobOutput.beginChronometer( <ChronometerName> );

\* and to stop it, use

\* > > milkJobOutput.endChronometer( sleepTimeMarker);

\* -

\* Chronometer save the time from Begin to End, and the number of occurrence. So, you get the final information on the total number of execution, and the

\* number of occurence.

\* Add all chronometers in the final report by

\* > > milkJobOutput.addChronometersInReport(addNumberOfOccurrence, addAverage);

\* > > > > addNumberOfOccurrence if true, the number of occurrence is added in the report

\* > > > > addAverage if true, the average is added in the report (total time / numberOfOccurent

\* --

## Advancement

\* To give a feedback to the user, an advancement can be calculated and send back to the user.

\* Note: the stop mechanism can stop immediately the job, and then the advancement may be stuck at final at 45% for example

\* You have two ways to give back a status to user.

\* > > milkJobOutput.setAvancementTotalStep( <totalStep>) / milkJobExecution.setAvancementStep(<StepValue>)

\* You setup the total number of step you detect. Imagine that you want to delete cases. You detect that you have to delete 454 cases. Then, set the

\* totalNumber to 454, and after each deletion, set the avancementStep to the number of case deleted.

\* or use

\* > > milkJobOutput.setAvancement( advancementInPercent )

\* then it's up to you to calculate the % of advancement.

\* To give more feedback on what's is going on, use the milkJobOutput.setAvancementInformation(String information) to set some information.

\* Please note:

\* - Truckmilk does not update in the database the advancement at each time. it do that only every X seconds (30 secondes). This is to avoid to slow down

\* the performance.

\* Imagine that you set the TotalStep to 4 412 044, and you update the advancement for each step. You don't want that Truckmilk do 4 412 044 updates in the

\* database.

\* So, if the last setAvancement() was done in the last 30 seconds, it register the value in memory only.

\* Same for the percentage: if the percentage does not change, there is no update in the database

\* --

## Stop mechanism

\* if your treatment take time, you have to implement a stop mechanism.

\* Truckmilk will not kill your thread, in order to not abort brutally a treatment. So you can choose when you can stop.

\* Just check the method

\* > > milkJobExecution.isStopRequired()

\* it is is true, then the stop is required

\* For your information, there are three way to stop:

\* - user explicitly require it by clicking on the stop button

\* - user specify a "maximum execution time" and your execution reach this limit

\* - user specify a "maximum item to process", and, via the setNbItemsProcessed(), you reach this number

\* --

## Document management

\* Your execution may need to access a document, or will create a document.

\* The report is limited in size, so if you need to produce a list of information, you should not use the Report, but produce a document (a CSV file for

\* example).

\* A document can be accessed in READ, in WRITE (you produce it) or in READ/WRITE (you update a list of information for example).

\* -

\* First, you have to declare the parameter. You have to give a file name and a content type (used by the browser when you upload the document)

\* > > PlugInParameter cstParamMyReadDocument = PlugInParameter.createInstanceFile("readIt", "Read the document", TypeParameter.FILEREAD, null, "Read", "Read.csv", "application/CSV")

\* -

\* PlugInParameter cstParamMyWriteDocument = PlugInParameter.createInstanceFile("writeIt", "Write the document", TypeParameter.FILEWRITE, null, "Write", "Write.csv", "application/CSV")

\* -

\* PlugInParameter cstParamMyReadWriteDocument = PlugInParameter.createInstanceFile("readWriteIt", "Read-Write the document", TypeParameter.FILEREADWRITE,

\* null, "List is calculated and saved in this parameter", "ReadWrite.csv", "application/CSV")

\* so you read the document by:

\* List<BEvent> milkJobExecution.getParameterStream(<PlugInParameter>, <OutputStream>)

\* all the content is sent to out OutputStream.

\* and you write the document by :

\* milkJobOutput.setParameterStream( <PlugInParameter>, <InputStream>)

\* Note: documents are store in the database as a BLOB.

## State of the art

* Don’t hard code any value in the plug in. Use parameters. For example, if your plug-in has a delay to calculate the scope (delay when a process was deployed, delay when a case was archived), don’t hard code this delay, ask it as a parameters.
* Use parameters. Your plug in works on process? Add a parameter to filter the scope of processes. If the parameter is empty, then check all processes
* Use the correct parameters type: for a process, use ARRAYPROCESS, not STRING.
* If your plug in is very heavy (for example, purge cases can need time if you want to purge 100 000 cases), then use the setAvancement() method. Administrator will see the advancement and can stop it.
* For a long running, ask as a parameter a maximum operation, or a maximum time to process. Then, you will not execute a “4 days works”, and the administrator can configure the treatment to run only 3 hours every night for example.
* Think big. Keep in mind your plug in can work on a large panel of input, so you have to create a robust plug in. use the setAdvancement(), allow the work to be stop after a certain number of items.
* Use the BEvent library. To report information, error, prefer the BEvent library. A BEvent contains the error title, plus the consequence and action to fix it. Who is the best person to document the “what to do”? The developer when he references the error. It’s a (little) more works for the developer, but really help the administrator. Then, because each BEvent has a unique number, it’s easy to find in the code where is the error.

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