HP OSS Analytics Foundation

Version 1.1.1

Integration Guide

Edition: 1.0

For Linux, RHEL 6.5

October 2015

Contents

Preface **6**

Chapter 1 .... **8**

Product overview **8**

1.1 HP OSS Analytics Foundation introduction 8
1.2 HP OSS Analytics Foundation architecture 8

Chapter 2 .... **9**

OSS Analytics batch jobs **9**

2.1 Batch Engine shell tool 9
2.2 Scheduled batch setup 10
In this section we will describe how to register a new batch in the system. 10
2.2.1 Batch identifier unicity 12
2.2.2 Batch Scheduler 13
2.2.3 Default Batch Job Parameters 14
2.1 Batch Engine services 15
2.1.1 Properties inheritance 15
2.1.2 Externalized properties 16
2.1.3 Transient and Persistent context 17
2.1.4 Properties templating and placeholding 17
2.2 OSSA Batchlet Library 20
2.2.1 SQL Batchlet 21
2.2.2 CopyToVertica 24
2.2.3 ConsoleReport 27
2.2.4 Summarization 30
2.2.5 Mailer 34
2.2.6 Http 37
2.2.7 Resource 39
2.2.8 Java scripting 41
2.2.9 Run Batch 44
2.2.10 Run System Process 45
2.3 Batch Job examples 46

Figures

Figure 1 - View default job input parameter from OSSA Foundation Administration Console 14
Figure 2 - Batchlet properties inheritance sample 15
Figure 3 - Freemarker template engine overview 17
Figure 4 - Generate dynamic property value with template 19
Figure 5 - HP Analytics Batchlets package 20
Figure 6 - ossa.Sql batchlet sample usage 23
Figure 7 - Loaded test/demonstration package in repository 46

Tables

Table 1 - Scheduled Batch configuration details 11
Table 2 - Calendar-Based Scheduler Attributes 13
Table 3 - Default batch job input parameters details 14
Table 4 - Templating context 18
Table 5 - ossa.Sql batchlet interace 22
Table 6 - ossa.CopyToVertica batchlet interface 25
Table 7 - ossa.ConsoleReport batchlet interace 28
Table 8 - Summarization batchlet interface 32
Table 9 - ossa.Mail bachlet interface 35
Table 10 - ossa.http batchlet interface 38
Table 11 - ossa.Resource batchlet interface 39
Table 12 - ossa.javascript batchlet interface 42
Table 13 - ossa.batch batchlet interface 44
Table 14 - ossa.run batchlet interface 45
Table 15 - Test package overview 47

Preface

This guide describes how to use the OSS Analytics Foundation functionalities for customizing your OSS Analytics solution.

Software component name: HP OSS Analytics Foundation

Software component version: 1.1.1

Software kit version: V1.1.1

Intended audience

- This integration guide is for anyone who is responsible for customizing an OSS Analytics solution based on OSS Analytics Foundation:
  - Solution Architects
  - Integrators
- The readers are assumed to understand Linux shell concepts.

Typographical conventions

`Courier` font:

- Source code and examples of file contents
- Commands that you enter on the screen
- Path names
- Keyboard key names

Italic text:

- File names, programs, and parameters
- The names of other documents referenced in this manual

Bold text:

- To introduce new terms and to emphasize important words

Associated documents

- HP OSS Analytics Foundation Release Notes
- HP OSS Analytics Foundation Installation Configuration and Administration Guide

Support

Visit the HP Software Support Online website at https://softwaresupport.hp.com/ for contact information, and for details about HP software products, services, and support.

The software support area of the website includes the following:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information

Chapter 1

Product overview

1.1 HP OSS Analytics Foundation introduction

Please refer to the *HP OSS Analytics Foundation Installation, Configuration and Administration guide* for an introduction to OSS Analytics Foundation.

1.2 HP OSS Analytics Foundation architecture

Please refer to the *HP OSS Analytics Foundation Installation, Configuration and Administration guide* for a description of the architecture of OSS Analytics Foundation.

Chapter 2

OSS Analytics batch jobs

2.1 Batch Engine shell tool

During integration or testing activities, you certainly won't schedule your batch jobs, and, it will be more useful to start your batch jobs manually. Two shell scripts allow to start batch execution either in synchronous mode or in asynchronous mode.

Asynchronous mode:

In this mode, as soon as the batch is started, the shell returns to user the started job execution instance id.

```
/opt/ossa/bin/ossa-run-batch.sh  [BATCH-NAME] [JSON_ PARAMS]
```

BATCH-NAME : (required) name of the batch to be run

JSON_PARAM : (optional) JSon object representing the batch job parameters

Synchronous mode:

In this mode, the shell starts the job and waits for its completion before returning the batch job execution details.

```
/opt/ossa/bin/ossa-run-sync-batch.sh  [BATCH-NAME] [JSON_PARAMS] [TIMEOUT]
```

BATCH-NAME : (required) name of the batch to be run

JSON_PARAM : (optional) JSon object representing the batch job parameters

TIMEOUT : (optional) Maximum time in milliseconds to wait before returning. It won't stop the job, simply give back control to caller process.

2.2 Scheduled batch setup

In this section we will describe how to register a new batch in the system.

A batch job is a valid JSR-352 Xml file which defines its processing.

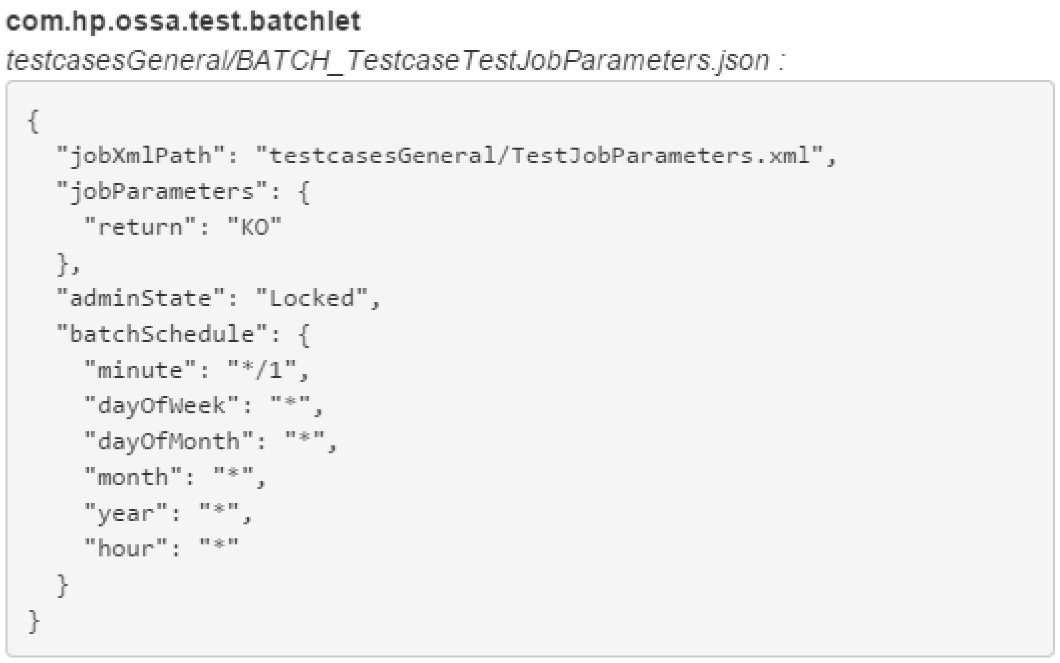
Moreover, for each batch you want load on the system, you must create a dedicated JSon file responsible to declare the batch Xml file and its schedule.

The file name should respect the following naming convention in order to be recognized by the system: file name must start with **BATCH_**. It will then be considered as a batch job configuration file.

Note

You can find some examples of batch jobs descriptions and configurations files in the chapter '*Batch Jobs examples*' of this document.

In case of multiple batch jobs, you can arrange files as you want in a folder structure. The BATCH_xxx.json file contains the relative path of the batch Xml file. Here is a simple json batch setup file :



com.hp.ossa.test.batchlet

testcasesGeneral/BATCH_TestcaseTestJobParameters.json :

```
{
  "jobXmlPath": "testcasesGeneral/TestJobParameters.xml",
  "jobParameters": {
    "return": "KO"
  },
  "adminState": "Locked",
  "batchSchedule": {
    "minute": "*/1",
    "dayOfWeek": "*",
    "dayOfMonth": "*",
    "month": "*",
    "year": "*",
    "hour": "*"
  }
}
```

Here are details about each attributes for a batch job configuration defined in a BATCH_xx Json files:

ATTRIBUTE	TYPE	DESCRIPTION
jobXmlPath	String	This is the repository parameter entry name that contains the job xml definition. The batch Xml definition should be stored in the same package than its json setup file.
jobParameters	Map<String,String>	A Key-Val Json object defining job input parameters

adminState	Locked | Unlocked	Locked : the batch is not scheduled
		Unlocked : the batch is scheduled
batchSchedule	`java.ejb.Schedule`	A Json object defining the batch scheduler. See section 2.2.2 for more details

Table 1 - Scheduled Batch configuration details

By default, the batch job you have defined will not be able to run until the previous execution of this job is completed; this is to avoid potential concurrent access on data.

If you want a different behaviour, you can set the “concurrentFlag” to “true” within the “jobParameters”.
For example:

```
  Batch_xxx.json
  {
    “jobXmlPath” : “ ...”,
    “jobParameters” : {
        “concurrentFlag” : “true”
     }
    “adminState” : “...”,
   “batchSchedule” : { ...}
  }
```

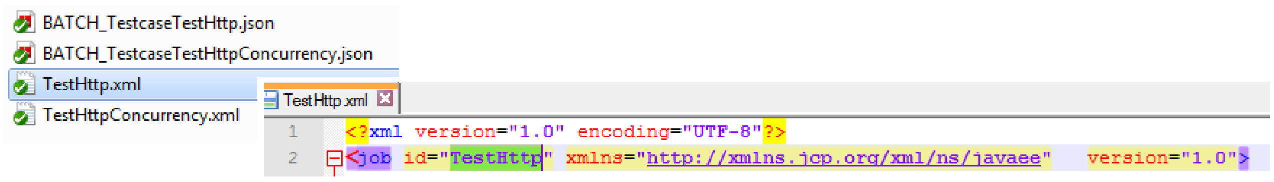
In that case, several executions of jobs can be run in parallel.

2.2.1 Batch identifier unicity

A batch is identified by a unique name in the system.

The name of a batch job is defined according to the Xml file name. If a job is defined in the MyBatch.xml file, the batch name will be MyBatch.

***WARNING**: The JobID defined in the ID job attribute in the batch xml definition file should be the same than the batch.*



BATCH_TestcaseTestHttp.json
BATCH_TestcaseTestHttpConcurrency.json
TestHttp.xml
TestHttpConcurrency.xml

TestHttp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="TestHttp" xmlns="http://xmlns.jcp.org/xml/ns/javaee"   version="1.0">
```

2.2.2 Batch Scheduler

The HP OSS Analytics batch engine scheduler is built on top of standard J2EE Timers.

For more details about J2EE Timers, please refer to official documentation:

https://docs.oracle.com/javaee/6/tutorial/doc/bnboy.html

The following table has been extracted from this pointer. It gives details about schedule calendar attributes:

ATTRIBUTE	DESCRIPTION	DEF. VALUE	ALLOWABLE VALUES AND EXAMPLES
second	One or more seconds within a minute	0	0 to 59. For example: second="30".
minute	One or more minutes within an hour	0	0 to 59. For example: minute="15".
hour	One or more hours within a day	0	0 to 23. For example: hour="13".
dayOfWeek	One or more days within a week	*	0 to 7 (both 0 and 7 refer to Sunday). For example: dayOfWeek="3". Sun, Mon, Tue, Wed, Thu, Fri, Sat. For example: dayOfWeek="Mon".
dayOfMonth	One or more days within a month	*	1 to 31. For example: dayOfMonth="15". –7 to –1 (a negative number means the *n*th day or days before the end of the month). For example: dayOfMonth="-3". Last. For example: dayOfMonth="Last". [1st, 2nd, 3rd, 4th, 5th, Last] [Sun, Mon, Tue, Wed, Thu, Fri, Sat]. For example: dayOfMonth="2nd Fri".
month	One or more months within a year	*	1 to 12. For example: month="7". Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec. For example:month="July".
year	A particular calendar year	*	A four–digit calendar year. For example: year="2011".

Table 2 - Calendar-Based Scheduler Attributes

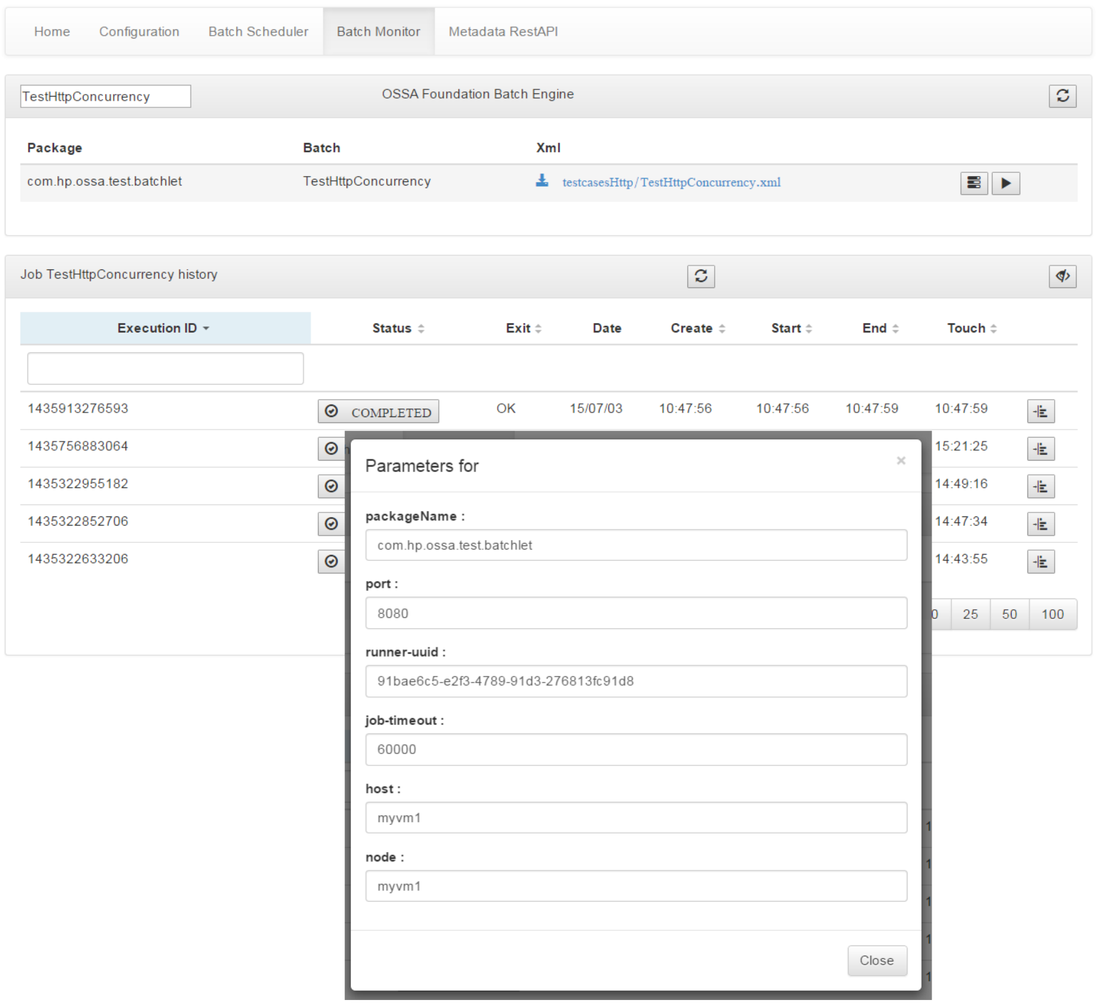
2.2.3 Default Batch Job Parameters

Before the system starts a new job instance, some technical input job parameters are automatically added to the execution context :

INPUT PARAMETER NAME	TYPE	DESCRIPTION
host	Text	The host name of the batch node runner for the job execution
port	Int	The host http port of the batch node runner for the job execution
runner-uuid	Text	Unique identifier of the batch node runner. This UUID is unique for each started HP OSS Analytics Foundation framework.
node	Text	When clustered system will be supported, this will be the name of the cluster node.
packageName	Text	The repository package name where the job definition is coming from

Table 3 - Default batch job input parameters details

Once the job is started, you can see these input parameter values directly in the HP OSS Analytics Foundation Administration console in the Batch Monitor screen by clicking on the ‘Status’ button of the job execution.



Package	Batch	Xml
com.hp.ossa.test.batchlet	TestHttpConcurrency	testcasesHttp/TestHttpConcurrency.xml

Job TestHttpConcurrency history

Execution ID	Status	Exit	Date	Create	Start	End	Touch
1435913276593	COMPLETED	OK	15/07/03	10:47:56	10:47:56	10:47:59	10:47:59
1435756883064							15:21:25
1435322955182							14:49:16
1435322852706							14:47:34
1435322633206							14:43:55

Parameters for

packageName : com.hp.ossa.test.batchlet

port : 8080

runner-uuid : 91bae6c5-e2f3-4789-91d3-276813fc91d8

job-timeout : 60000

host : myvm1

node : myvm1

Figure 1 - View default job input parameter from OSSA Foundation Administration Console

2.1 Batch Engine services

The OSS Analytics Foundation batch job processing definitions xml files are following the JSL (Job Specification Language) relying on JSR-352.

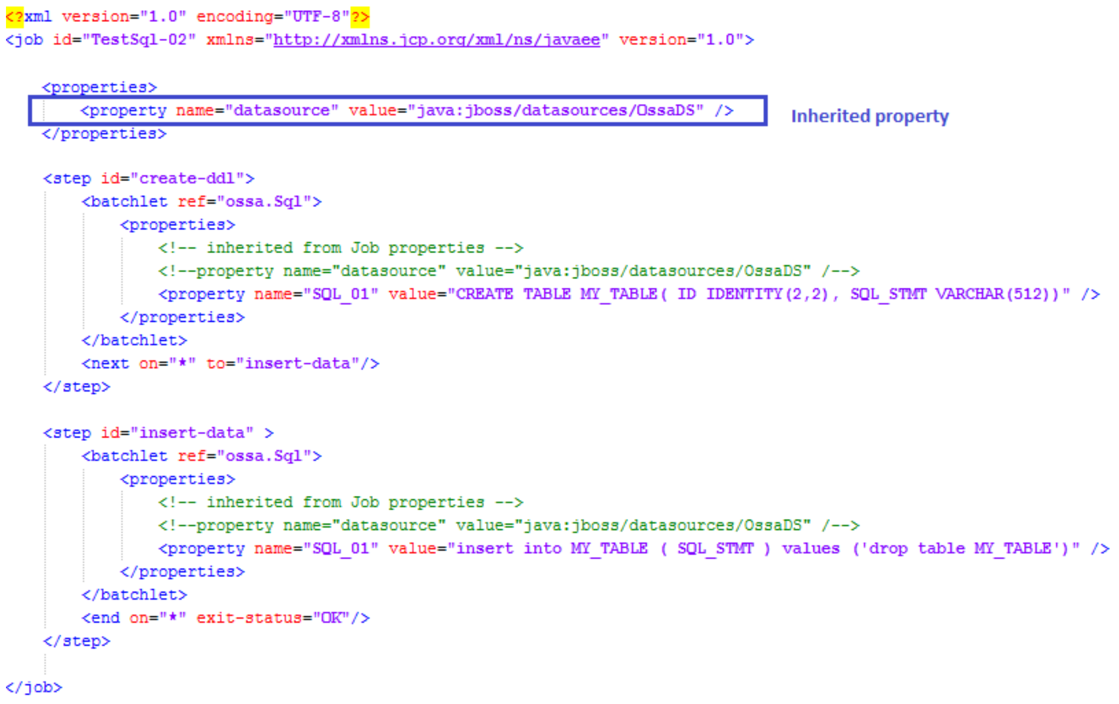
For more information on JSL and JSR-352 specification, please refer to https://jcp.org/aboutJava/communityprocess/final/jsr352/index.html)

2.1.1 Properties inheritance

The JSR-352 standard allows integrators to define properties at different level in a Job XML definition.
HP OSS Analytics batch engine proposes an inheritance mechanism for properties defined at Job, Step and Batchlet level.

For all provided HP OSS Analytics Foundation batchlet, if a property is not defined at the batchlet level, at runtime, the system will try to find the property value at the Step level, the Job level and finally as input Job parameters level.

Let take a concrete example : the ossa.Sql batchlet need a 'datasource' property. If the integration process has several ossa.Sql steps, it's not helpful to duplicate this 'datasource' property in all steps. This is a good candidate for property inheritance usage. You can define at the Job level a 'datasource' property that will be inherited by all ossa.Sql steps.



```
<?xml version="1.0" encoding="UTF-8"?>
<job id="TestSql-02" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">

    <properties>
        <property name="datasource" value="java:jboss/datasources/OssaDS" />
    </properties>

    <step id="create-ddl">
        <batchlet ref="ossa.Sql">
            <properties>
                <!-- inherited from Job properties -->
                <!--property name="datasource" value="java:jboss/datasources/OssaDS" /-->
                <property name="SQL_01" value="CREATE TABLE MY_TABLE( ID IDENTITY(2,2), SQL_STMT VARCHAR(512))" />
            </properties>
        </batchlet>
        <next on="*" to="insert-data"/>
    </step>

    <step id="insert-data" >
        <batchlet ref="ossa.Sql">
            <properties>
                <!-- inherited from Job properties -->
                <!--property name="datasource" value="java:jboss/datasources/OssaDS" /-->
                <property name="SQL_01" value="insert into MY_TABLE ( SQL_STMT ) values ('drop table MY_TABLE')" />
            </properties>
        </batchlet>
        <end on="*" exit-status="OK"/>
    </step>

</job>
```

Figure 2 - Batchlet properties inheritance sample

In this example, the Job has two ossa.Sql steps. Each step expects to have a 'datasource' property defined but here they are not. The required property value is inherited from the Job property 'datasource'. The value is shared by the two steps.

2.1.2 Externalized properties

HP OSS Analytics Batch Engine comes with the possibility to externalize in a file a property value.

It can be very useful when batchlet property values are quite long or if the integrator want to preserve indentation.

Keep in mind that jobs are defined in Xml file. The Xml standard defines that the whitespace characters are not preserved in a node attribute.

Also remember that you cannot use > or < or & or " characters. If you need it, you have to use the html character for that like &gt; &lt; &amp; &quote;

See the following example where we define a javascript that will be referenced in a job using a batchlet *ossa.javascript*:

```
<property name="script" value="
    log.info('testing switch statement ' );
    /* testing if switch case statement are working with strings */
    var result = 'not-def';
    result;
    " />
```

The script property value attribute is defined on several lines with indentation. When the system reads the value, it will see and get something like this:

```
<property name="script" value=" log.info('testing switch statement ' ); /* testing if switch case statement are working with strings */ var result = 'not-def'; result; " />
```

That's why, when you are writing a java script directly in the Xml value attribute, you cannot use the comment character // because the carriage return character won't exist anymore during execution.

Note that the same issue exists with SQL comments '--' .

If you need to preserve indentation or simply want to separate concerns, all HP OSS Analytics batchlets support the externalization of the value in a separate file.

Values can be externalized into file by using the following patern within the job xml file: [[path/file]]

In this example, the java script is externalized to a js file. It use the JSR-352 placeholding to retrieve the package name of the repository entry. The js file location is relative from the repository base folder.

```
<step id="javascript" >
    <batchlet ref="ossa.javascript">
        <properties>
            <property name="script" value="[[#{jobParameters['packageName']}/testcasesScripting/TestScripting-02.js]]" />
        </properties>
    </batchlet>
    <end on="OK" exit-status="OK"/>
    <fail on="*" exit-status="KO"/>
</step>
```

In this way, you have no indentation limitation or forbidden characters and the integration flow logic is separated from the 'business' process implementation done here in javascript.

2.1.3 Transient and Persistent context

Two different contexts are available for integrators. It allows user data manipulation or sharing between steps or process.

2.1.3.1 Job transient user data

This context is shared by all steps of a job execution. It's created at batch startup time and destroyed when batch is stopped, abandoned or completed.

The job transient user data is a Key-Value bag (Map<String,Object>) where integrators can put or get any kind of objects with a given name.

This object is directly added to the templating context with the name 'data'. You can access the values within your batch job xml with: `${data.myValueName}`

2.1.3.2 Step persistent data

The StepContext allow integrator to store some objects in the database, in the STEP_EXECUTION table.

2.1.4 Properties templating and placeholding

With HP OSS Analytics Batch Engine, most of the batchlet properties can be templated thanks to the FreeMarker engine.

2.1.4.1 Freemarker overview

FreeMarker is a "template engine"; a generic tool to generate text output (anything from HTML to autogenerated source code) based on templates. It's a Java package, a class library for Java programmers. It's not an application for end-users in itself, but something that programmers can embed into their products.

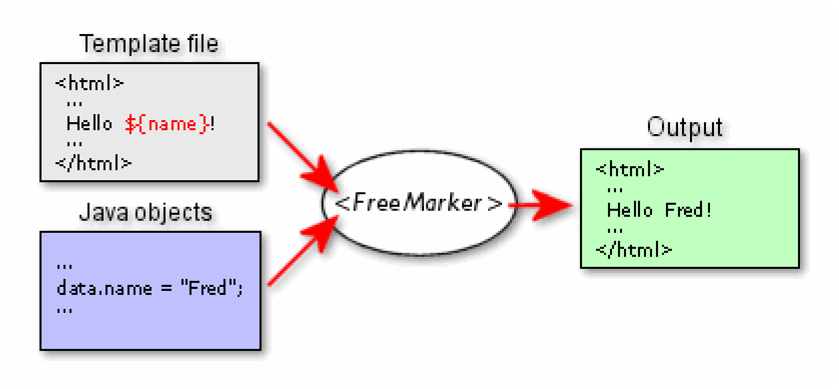


Figure 3 - Freemarker template engine overview

2.1.4.2 Templating context

When processing the freemarker template for a batchlet property, several objects are available as contextual objects and can be directly used by their names :

VARIABLE	DESCRIPTION	TYPE
log	The Logger for the underlying step batchlet	org.jboss.logging.Logger
job	The JobContext object https://docs.oracle.com/javaee/7/api/javax/batch/runtime/context/JobContext.html	javax.batch.runtime.context.JobContext
jobProps	Properties defined at Job level	java.util.Properties
step	The StepContext object https://docs.oracle.com/javaee/7/api/javax/batch/runtime/context/StepContext.html	javax.batch.runtime.context.StepContext
stepProps	Properties defined at the Step level	java.util.Properties
env	System env properties (java.lang.System.getenv())	Map<String,String>
props	System properties (java.lang.System.getProperties())	java.util.Properties
params	Job input parameters	java.util.Properties
data	The Job transient data	Map<String,Object>
[step-id]	The batchlet itself	Extends OssaBatchlet
batch	The OssaF Batch restApi proxy	com.hp.ossa.batch.restapi.BatchRestApi

Table 4 - Templating context

2.1.4.3 Placeholding

The JSR-352 placeholding for properties is done at the batch creation time. This means that you cannot use it to play with transient or persistent data to inject values between steps.

The Freemarker templating is done just before the usage of a property. With this, you can inject loaded values directly between job steps.

In order to understand the situation, considerate the following example:

```
<batchlet ref="ossa.Sql">
    <properties>
        <property name="SQL_01" value="insert into MyTable( COL_01 ) values ('some data')" />
        <property name="SQL_02" value="DATA(generatedId) SELECT LAST_INSERT_ID()" />
        <property name="SQL_03" value="insert into AnOtherTable (NEW_COL_01_ID) values ( ${data.generatedId} ) " />
    </properties>
</batchlet>
```

The first statement insert a row in MyTable

The second statement retrieves the generated identifier and save it as transient user data with the name 'generatedId'

In the last statement, we use the freemarker placeholding ${data.generatedId} value to do something else… here an other insert.

This is only possible because properties are templated at usage time.

2.1.4.4 Templating

The Freemarker templating can also be used to generate different property values depending of the context.

Imagine you are in a Sql batchlet, with an externalized Sql file and depending on a property value, you want to change the Sql statement to run :

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="TestDynamicSql" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">

    <properties>
        <property name="targetType" value="user" />
    </properties>

    <step id="do-something">
        <batchlet ref="ossa.Sql">
            <properties>
                <property name="SQL_01" value="[[sql/dynamic.sql]]" />
            </properties>
        </batchlet>
        <next on="*" to="next-step"/>
    </step>

    <step id="next-step">
    ...
```

Figure 4 - Generate dynamic property value with template

Here, depending of the 'targetProperty' value, we need to have different OrderBy part on the executed sql statement. We can use templating facilities for that like this :

```
-- demonstrate how to generate dynamic sql statement
-- built accoring to a job property named 'targetType'
SELECT
    COL_01, COL_02, COL_03
FROM
    MyTabble
ORDER BY
<#if jobProps['targetType'] == 'USER'>
     COL_01 ASC
<#else>
     COL_02 ASC, COL_03 DESC, COL_01 AS
</#if>
```

Dynamic sql part

Freemarker comes with a lot of functions to manipulate data. It proposes everything you need for conditional processing or formatting.

Please refer to the official Freemarker documentation for more details about capacities.

http://freemarker.org/index.html

2.2 OSSA Batchlet Library

The OSS Analytics Foundation embeds a OSS Analytics Batch Library which can be used for transformation purposes. This can serve to customize your OSS Analytics solution.

Several kind of batches are defined in this library; they are presented below.

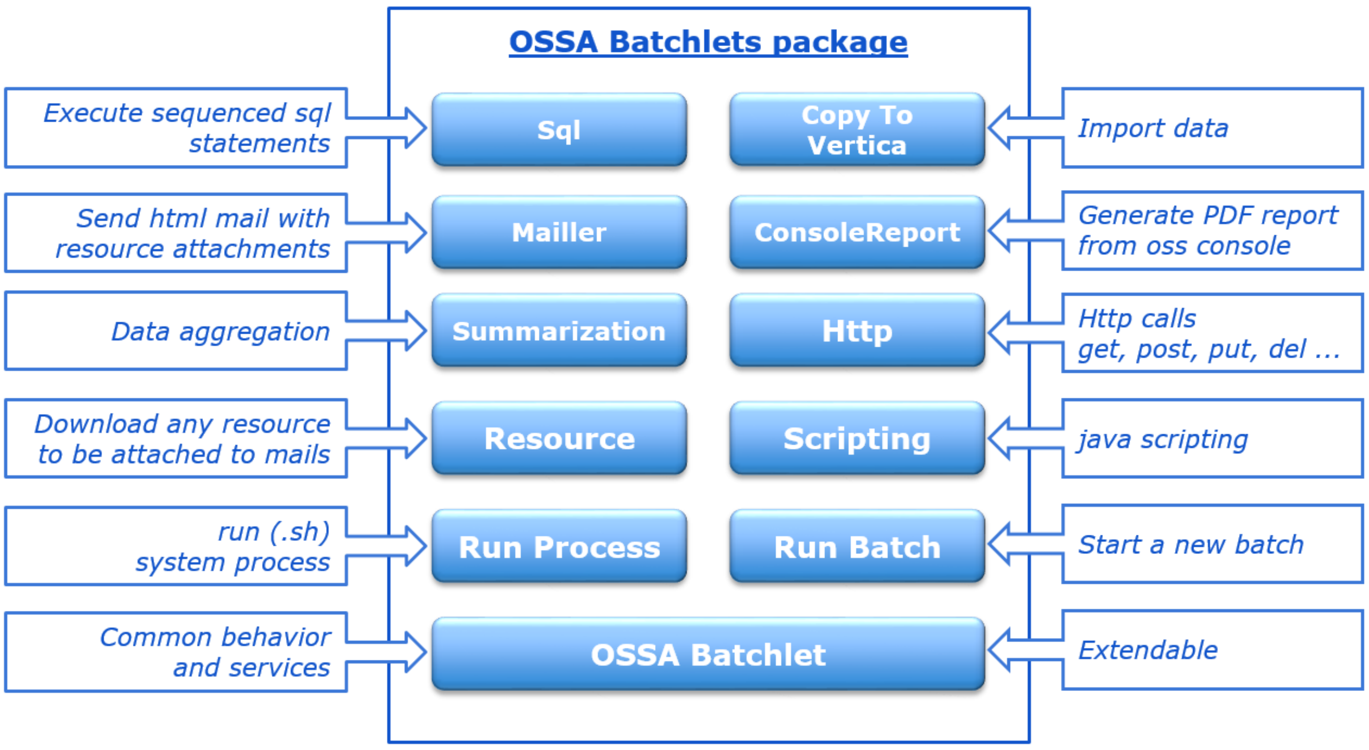


Figure 5 - HP Analytics Batchlets package

2.2.1 SQL Batchlet

2.2.1.1 Overview

The ossa.Sql batchlet allow user to run standard Sql statements on generic datasource.

Sql batchlet helps integrators to execute sequential and transactional DB operations.

With JSR-352, the transaction boundaries is a step. Every statements executed in a Sql batchlet step is committed at the end of the step.

The ossa.Sql batchlet supports cancel action. If the Stop method is called on the job running a Sql batchlet, the current statement is cancelled by the Vertica DB engine.

Sql batchlet allow user to mix Sql statement execution and data import in the same transaction. The CopyToVertica batchlet can be embedded into one of the SQL_XX properties.

The SqlBatchlet provided 99 optional properties named from SQL_01 to SQL_99. These statements are executed in sequence from 01 to 99. No continuity numbering is required.

SQL_XX usage:
Simple Sql processing. Any statement types are allowed as soon as the database accept it.

A special property named SQL_RETURN helps integrator to define the step Exit Status thanks to a Sql query. It's useful to drive the job execution flow.

Only the first object of the first column of the resultSet is used as the step ExitStatus

Ossa Foundation implements on top of SQL several useful services for integration purpose:

- EXECUTE *sql to be executed*

  If the statement is starting by EXECUTE, the end of the command will be considered as an SQL statement that should be executed.
  The provided SQL statement is supposed to produce SQL. The produced SQL statements are applied in sequence

- VCOPY *config-key*

  If the statement is starting by VCOPY, it will be treated as a ossa.CopyToVertica step.
  You simply have to provide the property key to use to find copy settings.
  (please refer to the ossa.CopyToVertica batchlet description)

- DATA(key) *sql statement*

  DATA allows user to store the query result in the job data context.
  Data is accessible thanks to the name pattern [step-id].[key]
  Data are transient. They are not stored to the DB but accessible from all job steps.

If the query returns one row and one column, the resultset object is stored directly

If the query returns one or multiple rows, it stores a list of key-value <columnName,object>

- STORE(key) *sql statement*

STORE allow user to persist in the database a query result.
The resultset value is attached to the step. It's stored in the step_execution batch tables.
It also store the query result as a job context data under the name [step-id].[key]

2.2.1.2 ossa.Sql batchlet interace

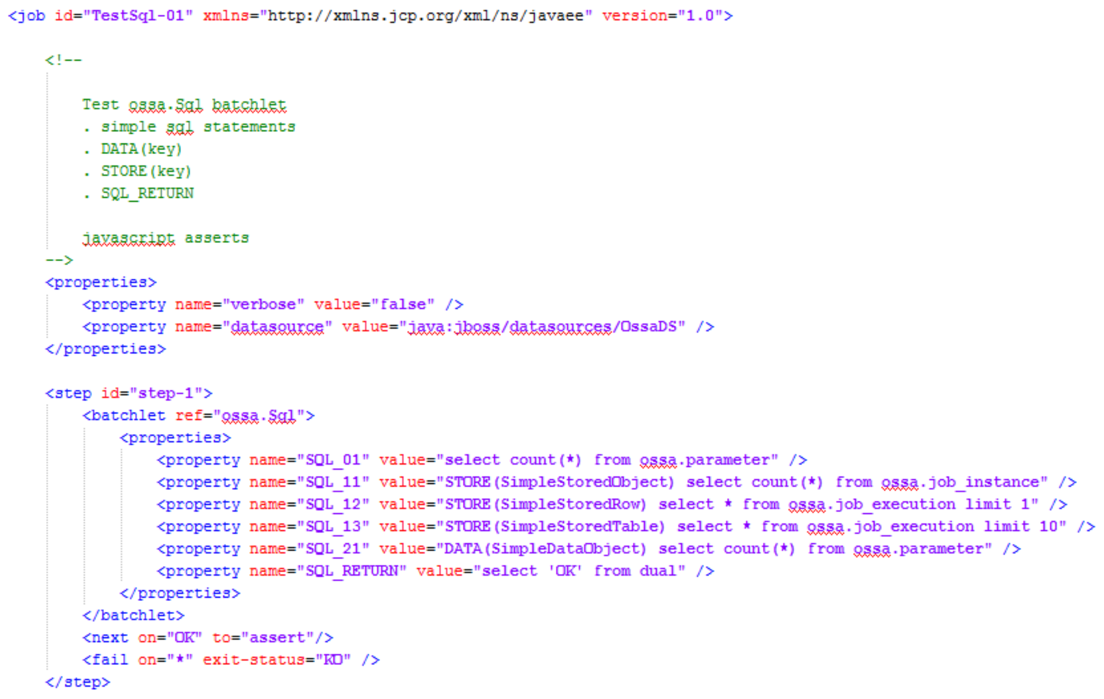
Support				
	Ref	ossa.Sql		
	Logger	com.hp.ossa.batch.batchlet.jdbc.SqlBatchlet		
	Cancellable	Yes		
	Templated	Yes		
Properties				
	datasource	JNDI url of datasource	req	String
	SQL_01	Sql to be executed in sequence	opt	String
	SQL_02		opt	String
	…		…	
	SQL_99		opt	String
	SQL_RETURN	Sql executed to get the step exit status. Used by next command in job xml	opt	String
Exit Status				
	COMPLETED	If no problems and no SQL_RETURN defined		
	[AnyUserValue]	If SQL_RETURN is defined, the first column data of the first row if return as step output		
Output Data				
	[AnyName]	In case of DATA(key) or STORE(key) usage	Object List[Map<Col,Val>]	

Table 5 - ossa.Sql batchlet interace

2.2.1.3 Sample usage

This sample has been extracted from:

/opt/ossa/repo-ossa/com.hp.ossa.test.batchlet/testcasesSql



```xml
<job id="TestSql-01" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">

    <!--

        Test ossa.Sql batchlet
        . simple sql statements
        . DATA(key)
        . STORE(key)
        . SQL_RETURN

        javascript asserts
    -->
    <properties>
        <property name="verbose" value="false" />
        <property name="datasource" value="java:jboss/datasources/OssaDS" />
    </properties>

    <step id="step-1">
        <batchlet ref="ossa.Sql">
            <properties>
                <property name="SQL_01" value="select count(*) from ossa.parameter" />
                <property name="SQL_11" value="STORE(SimpleStoredObject) select count(*) from ossa.job_instance" />
                <property name="SQL_12" value="STORE(SimpleStoredRow) select * from ossa.job_execution limit 1" />
                <property name="SQL_13" value="STORE(SimpleStoredTable) select * from ossa.job_execution limit 10" />
                <property name="SQL_21" value="DATA(SimpleDataObject) select count(*) from ossa.parameter" />
                <property name="SQL_RETURN" value="select 'OK' from dual" />
            </properties>
        </batchlet>
        <next on="OK" to="assert"/>
        <fail on="*" exit-status="KO" />
    </step>
```

Figure 6 - ossa.Sql batchlet sample usage

2.2.2 CopyToVertica

2.2.2.1 Overview

The ossa.CopyToVertica batchlet is a simple encapsulation of the the Vertica COPY sql statement.

This SQL function provided by Vertica allow user to load datafile in the database.

For more details about COPY Vertica statement, please refer to the official Vertica Sql Reference documentation.

For Vertica 6.1 : http://my.vertica.com/docs/6.1.x/HTML/index.htm#1668.htm

The ossa.CopyToVertica batchlet propose, on top of this sql utility, functions to manage input files. The integrator can specify the archiving policy for data file import.

OSSAF supports:

REMOVE policy: where loaded files are simply removed.

ARCHIVE policy: where loaded files are archived to defined folder.

NO policy: nothing is done. The file stay in place.

The ossa.CopyToVertica batchlet supports cancel action. If the Stop method is called on the job running a Sql batchlet, the current statement is cancelled by the Vertica DB engine.

The integrator is responsible to provide the Copy Sql statement. The file location is manage by placeholding of the *:input* token.

2.2.2.2 ossa.CopyToVertica batchlet interface

Support				
	Ref	ossa.CopyToVertica		
	Logger	com.hp.ossa.batch.batchlet.jdbc.CopyToVerticaBatchlet		
	Cancellable	Yes		
	Templated	Yes		
Properties				
	datasource	JNDI url of datasource	req	String
	asBaseDir	Relative base directory to find files from the application server	req	String
	dbBaseDir	Relative base directory to find files from the database	req	String
	importStatement	Vertica copy sql statement :input will be replaced by the batchlet with selected file name	req	String
	inputFilter	This regexp should match the file name in the input directory	req	RegExp
	inputArchingPolicy	REMOVE : delete imported file	req	String
		ARCHIVE : file will be moved to the archive directory when loaded		
		NO : do nothing		
	inputDir	Directory where the system should find files to be imported	Req	String
	errorDir	Directory where the files are moved in case of error	req	String
	archiveDir	Directory where the files are moved when loaded	req	String
	rejectDir	Directory where the files are moved when rejected data	req	String
	failOnError	Default : true Define if the batchlet should stop on error in case of sql exception.	opt	boolean
Exit Status				
	COMPLETED	If no problems		
	FAILED	If one of the sql statements has failed		
Output Data				

Table 6 - ossa.CopyToVertica batchlet interface

2.2.2.3 Sample usage with placeholder

```
<step id="cleanupStep">
    <batchlet ref="ossa.CopyToVertica">
        <properties>
            <property   name="inputFilter"
                        value="^(nfva_ems2_naming_2.csv)$" />

            <property   name="importStatement"
                        value="
                            COPY R_CSV_ems2_NAMING
                            (
                                RECORD_TIMESTAMP FORMAT 'YYYY-MM-DD.HH:MI:SS.US',
                                IS_DELETION,
                                ENTITY_ID,
                                ARTIFACT_ID
                            )
                            FROM
                                #{jobProperties['import_mode']}
                                '#{jobProperties['baseDir']}/#{jobProperties['inputDir']}/:input'
                                #{jobProperties['import_node']}
                                DELIMITER ',' TRAILING NULLCOLS SKIP 1 DIRECT
                                abort on error
                                REJECTED DATA '#{jobProperties['baseDir']}/#{jobProperties['rejectDir']}/:input'
                                EXCEPTIONS    '#{jobProperties['baseDir']}/#{jobProperties['errorDir']}/:input'
                                NO COMMIT
            " />
        </properties>
    </batchlet>
</step>
```

<u>Inherited job properties</u>

```
<!-- Datasource at job level -->
<properties>
    <!-- Define datasource -->
    <property name="datasource" value="java:jboss/datasources/OssaDS" />
    <!-- Defines if the verbose mode is enabled -->
    <property name="verbose"    value="false" />
    <!-- defines if the step should fail in case of error -->
    <property name="failOnError" value="true" />
    <!-- defines the archive policy mode  ARCHIVE, REMOVE, NO -->
    <property name="inputArchivingPolicy" value="ARCHIVE" />
    <!-- defines the application server mounting point to access files -->
    <property name="asBaseDir" value="D:/temp/nfva" />
    <!-- defines the database server mounting point to access files-->
    <property name="dbBaseDir" value="/data2/NFVA" />
    <!-- defines the input subdirectory folder -->
    <property name="inputDir" value="ems2" />
    <!-- defines the error subdirectory folder -->
    <property name="errorDir" value="exceptions" />
    <!-- defines the archive subdirectory folder -->
    <property name="archiveDir" value="archives" />
    <!-- defines the rejection subdirectory folder -->
    <property name="rejectDir" value="rejections" />

    <property   name="baseDir"
                value="#{jobProperties['asBaseDir']}" />
    <property   name="import_mode"
                value="LOCAL" />
    <property   name="import_node"
                value="" />
</properties>
```

2.2.3 ConsoleReport

2.2.3.1 ossa.ConsoleReport overview

When Oss Foundation is deployed with the OssConsole application, it's possible to make it generated reports that can be integrated in other integration steps like a 'mailing" step for example.

In order to generate a report, integrators need to define:

1. authentication parameters

As the OssConsole is a secured application, the ConsoleReport batchlet supports 2 types of authentication:

Either you can use a specific user and password, that should be previously authorized to generate the wanted report. This option is only working if OssConsole has been configured to use internal authentification provider.

Either you can specific Auth2 Token value. This token should be a valid token recognized and accepted by the OssConsole application to generate a report.

For more details about how to generate and get a valid OSS Console Token, please refer to the ***Security Section - JSon webtoken*** *in the HP Unified OSS Console installation guide.*

---

2. Report specification

The 2 mains parameters to generate an Oss Console report are:

- The report data Uri : is the same uri the user navigates to see its data in OssConsole Application
- The ossConsoleReportUri: is the generator service uri of the OssConsole.

Moreover, you can define several specific report generation options like paper size, orientation, and margin.

2.2.3.2 ossa.ConsoleReport batchlet interace

Support				
	Ref	ossa.ConsoleReport		
	Logger	com.hp.ossa.batch.batchlet.jdbc.ConsoleReportBatchlet		
	Cancellable	No		
	Templated	Yes		
Properties				
	baseFolder	Output base folder where downloaded report are stored	req	String
	url	Url of the resource to be downloaded	req	String
	file	Name of the file where the resource will be stored	req	String
	ossConsoleUrl	Url of Oss Console application	req	String
	ossUsername	Login for the console	opt	String
	ossPassword	Password for the console	opt	String
	ossToken	Security token is no user and password	opt	String
	ossConsoleLoginUri	Login uri	req	String
	ossConsoleReportUri	Report generator uri	req	String
	uri	Report uri	req	String
	orientation	portrait or landscape	req	Enum
	format	A4, A3…	req	Enum
	margin	Eg. 10	req	Integer
	viewportHeight	Eg. 1200	req	Integer
	viewportWitdh	Eg. 1900	req	Integer
Exit Status				
	COMPLETED	If no problems		
	FAILED	If download failed		
Output Properties				
	res	Downloaded java file		
	link	http uri link to get the file		
	source	http link of oss console report		

Table 7 - ossa.ConsoleReport batchlet interace

2.2.3.3 Sample usage with User / Password

```
<!-- OSS Console report download -->
<step id="downloadReport" next="sendByMail">
    <batchlet ref="ossa.ConsoleReport">
        <properties>
            <!-- setup resourceBatchlet -->
            <property name="verbose" value="true" />
            <property name="baseFolder" value="d:/temp/mailler" />
            <property name="file" value="WS-FAS-NETWORK-MGMT-HEALTH.pdf" />

            <!-- setup ossConsoleResourceBatchlet -->
            <property name="ossConsoleUrl" value="http://dubaiv1.gre.hp.com:3000" />
            <property name="ossConsoleLoginUri" value="/auth/local/login" />
            <property name="ossConsoleReportUri" value="/V1.0/report" />

            <!-- Simulate user login -->
            <property name="ossUsername" value="admin" />
            <property name="ossPassword" value="admin" />

            <!-- Report generation settings -->
            <property name="uri" value="/workspaces/WS-FAS-NETWORK-MGMT-HEALTH" />
            <property name="orientation" value="landscape" />
            <property name="format" value="A4" />
            <property name="margin" value="10" />
            <property name="viewportHeight" value="1200" />
            <property name="viewportWidth" value="1900" />

        </properties>
    </batchlet>
</step>
```

2.2.3.4 Sample usage with Token

```
<!-- OSS Console report download -->
<step id="download" next="send">
    <batchlet ref="ossa.ConsoleReport">
        <properties>
            <!-- setup resourceBatchlet -->
            <property name="verbose" value="true" />
            <property name="baseFolder" value="/opt/ossa/batch-jobs/mailler" />
            <property name="file" value="WS-FAS-NETWORK-MGMT-HEALTH.pdf" />

            <!-- setup ossConsoleResourceBatchlet -->
            <property name="ossConsoleUrl" value="http://dubaiv1.gre.hp.com:3000" />
            <property name="ossConsoleReportUri" value="/V1.0/report" />
            <!--Valid OSS Console Report Toke n           -->
            <property name="ossToken" value="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE0MzQ0N
                                            zMwNTIsImV4cCI6MTQ2NjAwOTA3NywiaWQiOiJhZG1pbiIsIm5hbWU
                                            iOiJBZG1pbmlzdHJhdG9yIiwicm9sZXMiOlsiUGxhdGZvcm0gQWRta
                                            W5pc3RyYXRvciIsIlVzZXIgQWRtaW5pc3RyYXRvciIsIlBhY2thZ2U
                                            gRGVzaWduZXIiLCJPcGVyYXRvcl9MMSIsIk9wZXJhdG9yX0wyIiwiT
                                            3BlcmF0b3JfTDMiLCJBdXRob3JpemVkX09wZXJhdG9yX0Zvcl9Mb2M
                                            iLCJSZXBvcnRfRXhwb3J0ZXIiLCJHdWVzdCJdfQ.09w1dUG22CeTTl
                                            VpRrWxZw-gIhWXmcyONa0zPcoSeEU" />

            <!-- Report generation settings -->
            <property name="uri" value="/workspaces/WS-FAS-NETWORK-MGMT-HEALTH" />
            <property name="orientation" value="landscape" />
            <property name="format" value="A4" />
            <property name="margin" value="10" />
            <property name="viewportHeight" value="1200" />
            <property name="viewportWidth" value="1900" />

        </properties>
    </batchlet>
</step>
```

2.2.4 Summarization

2.2.4.1 Summarization overview

The OSS Analytics Batch Library provides a data summarization functionality.

This transformation is applicable for multidimensional schema, having a star model.

The aim is to aggregate data from a datamart ‘fact’ table.

The transformation produces data in an output table where values from the original table are <u>aggregated by selected dimensions and selected time granularities</u> (hourly, daily, weekly, monthly). This is why it is called summarization.

When scheduling summarization batch job, new and updated data from the original table is detected thanks to the CDC column (**C**hange **D**ata **C**apture), and the processing occurs in order to update accordingly the summarized table.

2.2.4.2 Summarization batchlet interface

SUPPORT

	Logger	**com.hp.ossa.batch.batchlet.summ**
	Cancellable	Yes
	Templated	No

PROPERTIES

	datasource	JNDI url of datasource	req	
	src_table	the name of the original fact table	req	
	dest_table	the name of the table where the aggregated data will be put	req	
	src_time_column	the column of the source table which determine the timestamp of the fact values (this column will be used for applying the time aggregation)	req	
	dest_time_slice	time slice on which the aggregation must be performed. It can be: '**X**MIN': for bunch of minutes aggregation. **x** could be 1,2,5,10,15,20 or 30 'HH24': for hourly aggregation 'DD': for daily aggregation, 'DAY' or 'IW' : for weekly aggregation starting on Sunday or Monday 'MM': for monthly aggregation	req	
	dest_timeColumn	the timestamp column within the summarized table which will be filled with the timestamp of the time slice (the timestamp of the start of the timeslice)	req	
	src_dimensions	the list of columns of the original fact table defining the dimensions upon which the aggregation must be performed	req	
	dest_sumAggregations	the list of mappings (separated by '//'): <column name in the summarized table> = <the aggregation function done on the fact columns (as a SQL expression)>	req	
	src_maxCalculationPeriod	max number of periods of data to be taken into account when handling backlog use cases. If 0, no limit on the number of periods to be calculated. If X different than 0, do not take data older than X <time slice> periods in the past into	req	

		consideration for the summarization		
	src_CDCcolumn	the column which must be used to detect the new data on the source table (an update timestamp generally)	req	
	src_CDCtype	the type of the CDC column (only timestamp is supported in V1)	req	
	src_CDCdeltaWindow	(value in minutes) If you ensure that your source table src_CDCcolumn has incremental timestamp values, set the value to 0. Potentially, for multithreaded applications populating the source table, where you are not sure that src_CDCcolumn values are incremental, you can define a delta window of 1 minute for retrieval of data; incremental summarization will then take data from source table having src_CDCcolumn > 'latest CDC timestamp took by previous summarization' – 1 mn		
	dest_CDCcolumn	column name within the destination table that will identify a new or updated row	req	
EXIT STATUS				
	COMPLETED	If no problems		
	FAILED	If summarization failed		

Table 8 - Summarization batchlet interface

2.2.4.3 Sample usage

In red, the configuration parameters of the summarization, as described above

```
<properties>
    <property name="datasource-fas" value="java:jboss/datasources/OssaFaultDS" />
</properties>

    <flow id="summarizeMGT_H">
        <step id="summPropertiesMGT_H">
        <next on="COMPLETED" to="summListCalcPeriodsMGT_H"/>
            <batchlet ref="summGetPropertiesBatchlet">
                <properties>
                    <property name="summarizationName" value="SUMM_HOURLY_MANAGEMENT" />
                    <property name="src_table" value="FCT_FAULT" />
                    <property name="dest_table" value="SUMM_HOURLY_MANAGEMENT" />
                    <property name="src_timeColumn" value="ORIGINALEVENTTIME" />
                    <property name="dest_timeSlice" value="'HH24'" />
                    <property name="dest_timeColumn" value="TIME" />
                    <property name="src_dimensions"
                        value="OPERATIONCONTEXTID,ACKUSERID,CLOSEUSERID,HANDLEUSERID,RELEASEUSERID,TERMUSERID,OUTAGEFLAG" />
                    <property name="dest_summAggregations"
                        value="SUMMARIZED_COUNT=count(1) // ALARM_OBJECTS_COUNTER_SUM=SUM(CASE WHEN alarmclass = 0 THEN 1 ELSE 0 END)" />
                    <property name="src_maxCalculationPeriod" value="2232" />
                    <property name="dest_retentionPeriod" value="2232" />
                    <property name="src_CDCcolumn" value="UPDATE_TIMESTAMP" />
                    <property name="src_CDCtype" value="TIMESTAMP" />
                    <property name="src_CDCdeltaWindow" value="1" />
                    <property name="dest_CDCcolumn" value="UPDATE_TIMESTAMP" />
                </properties>
            </batchlet>
        </step>
        <step id="summListCalcPeriodsMGT_H">
            <next on="COMPLETED" to="summMGT_H"/>
            <batchlet ref="summListCalcPeriodsBatchlet">
            <properties>
                <property name="datasource" value="#{jobProperties['datasource-fas']}" />
            </properties>
                </batchlet>
        </step>
        <step id="summMGT_H">
            <next on="COMPLETED" to="summFinalizeMGT_H"/>
            <batchlet ref="summBatchlet">
                <properties>
                    <property name="datasource" value="#{jobProperties['datasource-fas']}" />
                    <property name="calcPeriodId" value="#{partitionPlan['calcPeriodId']}" />
                </properties>
            </batchlet>
            <partition>
                <mapper ref="SummPartitionMapper">
                    <properties>
                        <property name="nbThreads" value="10" />
                    </properties>
                </mapper>
            </partition>
        </step>
        <step id="summFinalizeMGT_H">
            <batchlet ref="summFinalizeBatchlet" />
        </step>
    </flow>
```

2.2.5 Mailer

2.2.5.1 ossa.Mail batchlet overview

The ossa.mail batchlet is built on top of java.mail Api.

It allows integrators to send an Html templated mail in their integration flows.

Mail receivers can be defined thanks to the ‘to’, ‘cc’, ‘bcc’ properties. You can add several receivers by using the coma separator.

The property “from” allow integrators to define how is sending the mail. It should be accepted by your underlying smtp server.

The smtp server is defined at installation time when you configure the batch engine.

You have to define, in the setup file /opt/ossa/ossa.conf, properties OSSA_MAIL_SERVER and OSSA_MAIL_PORT. It will be configured automatically in the Wildfly application server configuration file.

The content of the mail should be Html content. As the batchlet properties are templated, you can externalize in a FTL script the content and use FreeMarker functions in order to generate dynamic content.

The “attachment” property can be use. It defines the list of previous step-id supposed to provide a resource to attach. For instance, a ConsoleReport step can be use before the Mail step. The downloaded report can be added as attachment to the mail to send.

2.2.5.2 ossa.Mail bachlet interface

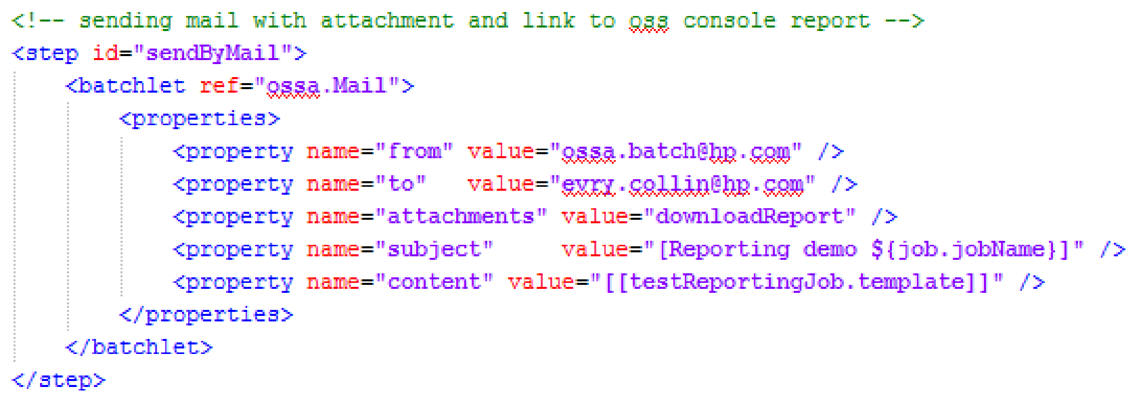
SUPPORT				
	Ref	ossa.Mail		
	Logger	com.hp.ossa.batch.batchlet.resource.MaillerBatchlet		
	Cancellable	No		
	Templated	Yes		
PROPERTIES				
	baseFolder	Output base folder	req	
	from	Mail sender address	req	
	to	List of mail receivers (,)	req	
	cc	List of mail cc receivers (,)	req	
	bcc	List of mail bcc receivers (,)	req	
	attachments	List of attachments. Use the step id of resource step to be attached to the mail	opt	
	subject	Mail subject	req	
	content	Mail content	req	
EXIT STATUS				
	COMPLETED	If no problems		
	FAILED	If download failed		

Table 9 - ossa.Mail bachlet interface

2.2.5.3 Sample usage

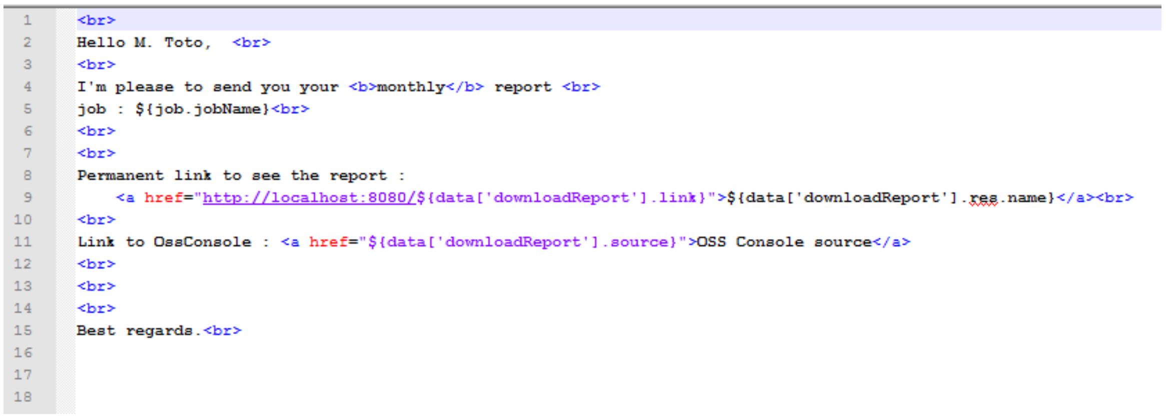
The mail content is produced by executing a freemarker template where the user can use environment data to produce the html mail content.

Mail step (With external template file)



```
<!-- sending mail with attachment and link to oss console report -->
<step id="sendByMail">
    <batchlet ref="ossa.Mail">
        <properties>
            <property name="from" value="ossa.batch@hp.com" />
            <property name="to"   value="evry.collin@hp.com" />
            <property name="attachments" value="downloadReport" />
            <property name="subject"     value="[Reporting demo ${job.jobName}]" />
            <property name="content" value="[[testReportingJob.template]]" />
        </properties>
    </batchlet>
</step>
```

Templated html mail content with dynamic data



```
<br>
Hello M. Toto,  <br>
<br>
I'm please to send you your <b>monthly</b> report <br>
job : ${job.jobName}<br>
<br>
<br>
Permanent link to see the report :
    <a href="http://localhost:8080/${data['downloadReport'].link}">${data['downloadReport'].res.name}</a><br>
<br>
Link to OssConsole : <a href="${data['downloadReport'].source}">OSS Console source</a>
<br>
<br>
<br>
Best regards.<br>



```

2.2.6 Http

2.2.6.1 ossa.http batchlet overview

The Http batchlet allow integrator to send every kind of Http request in their integration process.

The Http batchlet is built on to of apache.httpcomponents thanks to the httpclient api.

The ‘method’ property allow integrator to send all Http query type: GET, POST, PATCH, PUT, DELETE, HEAD, OPTIONS, TRACE.

The ‘headers’ property allow integrator to manage all Http header request parameter. Headers property is a Key,Val list.

The ‘url’ property defines the request target.

The 'content' property defines the content of the request to send. It’s template but is not mandatory.

The Http batchlet expect to receive an Http 200 response code. In this case, and if the response content is not null, it is stored by default as a job transient data with the key [step-id].result.
You can make this result persistent (stored in DB) by using the ‘store’ property. It defines the name of the persisted step data.

The Http batchlet can be cancelled by stopping the job.

The Http response code is return as the Exit Status for the step

2.2.6.2 ossa.http batchlet interface

SUPPORT				
	Ref	ossa.http		
	Logger	com.hp.ossa.batch.batchlet.http.HttpBatchlet		
	Cancellable	Yes		
	Templated	Yes		
PROPERTIES				
	method	Standard http method. GET|POST|PATCH|PUT|DELETE|HEAD OPTIONS|TRACE	req	Enum
	url	Request target url	req	Url
	headers	Http request headers	opt	K=V,
	content	Request content	opt	String
	store	Id of the data to store in the DB	opt	String
EXIT STATUS				
	[Http Response Status]	Http response code. 200, 404, …		
OUTPUT PROPERTIES				
	[step-id].result	Response content if exists	String	

Table 10 - ossa.http batchlet interface

2.2.6.3 Sample usage

In this example, we will call on of the OSSA RestApi entry point in order to get the server Uuid. We store in the DB, at the step level, the server response to be used later by another step or integration process

```
<step id="http-get" >
    <batchlet ref="ossa.http">
        <properties>
            <property name="method"     value="GET" />
            <property name="url"        value="http://${props['jboss.bind.address']}:${props['jboss.http.port']}/ossa/batch/uuid" />
            <property name="headers"    value="head1=val1, head2=val2" />
            <property name="store"      value="uuid" />
        </properties>
    </batchlet>
    <next on="200" to="assert"/>
    <fail on="*" exit-status="KO"/>
</step>
```

2.2.7 Resource

2.2.7.1 ossa.Resource batchlet overview

The ossa.Resource batchlet is a step abstraction that is supposed to provide a resource to other step. For example, the attachments of Mail batchlet should be resource step. The ConsoleReport batchlet extends the Resource batchlet. That's why, integrators can attach a downloaded report to a mail.

The Resource batchlet need first to be define a base folder where resources will be stored.

The default behaviour of Resource batchlet is to download the content from a given url (property url) and to store it in the base folder under the 'file' property file.

The exit status is COMPLETED if everything gone right.

2.2.7.2 ossa.Resource batchlet interface

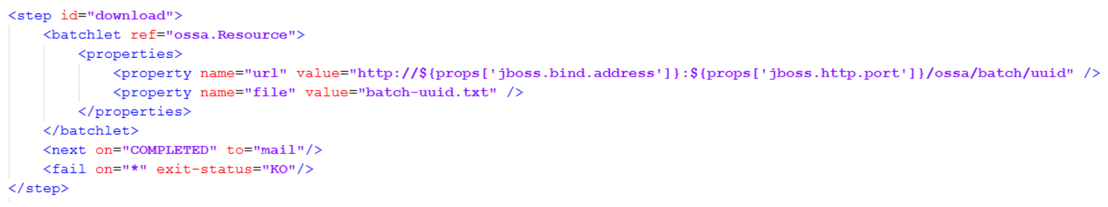
SUPPORT				
	Ref	ossa.Resource		
	Logger	com.hp.ossa.batch.batchlet.resource.ResourceBatchlet		
	Cancellable	No		
	Templated	Yes		
PROPERTIES				
	baseFolder	Directory where downloaded resource are stored as files.	req	String
	url	Resource url to be downloaded	req	Url
	file	Output file name		
EXIT STATUS				
	COMPLETED			
OUTPUT PROPERTIES				
	source	The full url where resource has been downloaded		
	link	The link where the downloaded resource can be retrieved		String
	res	The downloaded file		File

Table 11 - ossa.Resource batchlet interface

2.2.7.3 Sample usage

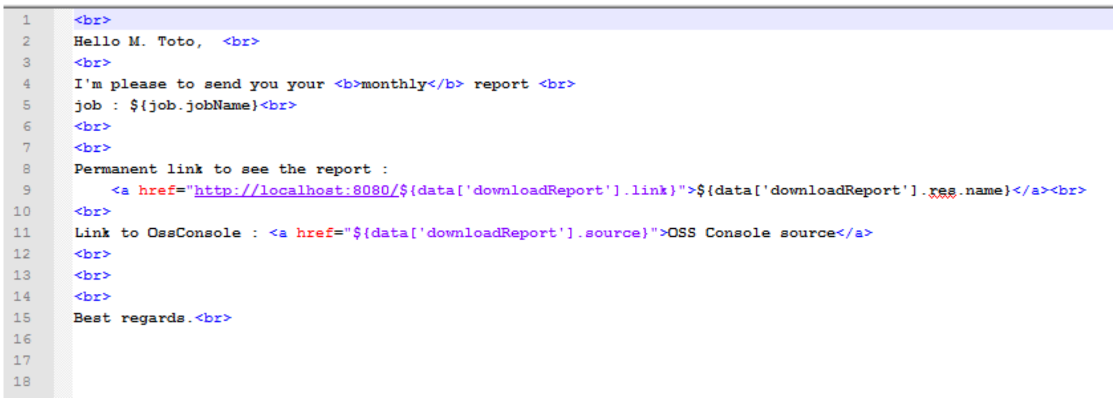
Here we will see again the Mail batchlet example.

In the first step, we download a file as an ossa.Resource:



```xml
<step id="download">
    <batchlet ref="ossa.Resource">
        <properties>
            <property name="url" value="http://${props['jboss.bind.address']}:${props['jboss.http.port']}/ossa/batch/uuid" />
            <property name="file" value="batch-uuid.txt" />
        </properties>
    </batchlet>
    <next on="COMPLETED" to="mail"/>
    <fail on="*" exit-status="KO"/>
</step>
```

Next we can use batchlet output properties. Here, we build a mail content with a link to get the resource:



```html
<br>
Hello M. Toto,  <br>
<br>
I'm please to send you your <b>monthly</b> report <br>
job : ${job.jobName}<br>
<br>
<br>
Permanent link to see the report :
    <a href="http://localhost:8080/${data['downloadReport'].link}">${data['downloadReport'].res.name}</a><br>
<br>
Link to OssConsole : <a href="${data['downloadReport'].source}">OSS Console source</a>
<br>
<br>
<br>
Best regards.<br>



```

2.2.8 Java scripting

2.2.8.1 ossa.javascript batchlet overview

The javascript batchlet allow integrator to implement any logic/treatment for a batch step thanks to a java scripting language.

It's build on top of Rhino java scripting engine.

For more details about Rhino Java scripting, please refer to the official documentation :
https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino

---

The aim of any batch step is to:
- do something
- provide an exit status that is used to drive the batch process.

This is meaning that the provided java script should return a value that can be used to drive the flow.

Rhino allow integrator to do whatever they want to do as soon as it can be implemented in Java. All API available in the JDK can be used directly.

Main java scripting usage cases:
- Initialisation step: before starting an integration process, you can setup variable, create files, load configuration from properties, generate random values ....
- Assertion step : in order to verify that everything gone right
- Call user defined functions

It accepts only one templated property named 'script'.

Execution context:

The execution context is exactly the same than the freemarker templating context. Please refer to section 2.4.4 for more details. You can use directly any of these variable available in the context:

VARIABLE	DESCRIPTION	TYPE
log	The Logger for JavaScript batchlet	org.jboss.logging.Logger
job	The JobContext object https://docs.oracle.com/javaee/7/api/javax/batch/runtime/context/JobContext.html	javax.batch.runtime.context.JobContext
jobProps	Properties defined at Job level	java.util.Properties
step	The StepContext object https://docs.oracle.com/javaee/7/api/javax/batch/runtime/context/StepContext.html	javax.batch.runtime.context.StepContext
stepProps	Properties defined at the Step level	java.util.Properties
env	System env properties (java.lang.System.getenv())	Map<String,String>
props	System properties (java.lang.System.getProperties())	java.util.Properties
params	Job input parameters	java.util.Properties
data	The Job transient data	Map<String,Object>
[step-id]	The batchlet itself	JavascriptBatchlet
batch	The OssaF Batch restApi proxy	com.hp.ossa.batch.restapi.BatchRestApi

2.2.8.2 ossa.javascript batchlet interface

SUPPORT				
	Ref	ossa.javascript		
	Logger	com.hp.ossa.batch.batchlet.script.JavascriptBatchlet		
	Cancellable	No		
	Templated	Yes		
PROPERTIES				
	script	Java script	req	String
EXIT STATUS				
	[AnyUserValue]	Value returned by the script.		

Table 12 - ossa.javascript batchlet interface

2.2.8.3 Sample usage

In this first example, we are using java scripting batchlet to generate folders and csv data in order to import in a later step this csv file thanks to the CopyToVertica batchlet. As you can see, you can access to the full java standard API (JDK 1.7 in June 2015)

```
<step id="init" >
    <batchlet ref="ossa.javascript">
        <properties>
            <property
                name="script"
                value="
                /* generate random table name */
                data.put('random-table-name', 'TMP_'+java.lang.System.currentTimeMillis() );
                /* generate csv data */
                var path = java.nio.file.Files.createTempDirectory('tmp-test-');
                var workingDir = path.toFile();
                workingDir.deleteOnExit();

                log.info('Created temp working folder '+workingDir.getAbsolutePath());

                /* create folder structure */
                data.put('tmp-basedir', workingDir.getAbsolutePath() );
                var inputDir = new java.io.File(workingDir, 'input');
                inputDir.mkdirs();
                new java.io.File(workingDir, 'error').mkdirs();
                new java.io.File(workingDir, 'archives').mkdirs();
                new java.io.File(workingDir, 'rejections').mkdirs();

                /* generate csv data file */
                var data = new java.io.File( inputDir, 'data.csv');
                var csv = 'data,data\ndata,data\ndata,data\ndata,data\ndata,'
                         +'data\ndata,data\ndata,data\ndata,data\ndata,data\ndata,data\ndata,data\n';
                var writer = new java.io.FileWriter(data);
                writer.write(csv);
                writer.close();
                log.info('Created data file '+data.getAbsolutePath());

                'INIT-DONE';
                "/>
        </properties>
    </batchlet>
    <next on="INIT-DONE" to="createDdl"/>
    <fail on="*" exit-status="KO" />
</step>
```

In this second example, we are using the javascript batchlet as an assertion step in order to verify that all steps previously executed finished with an ‘OK’ status. If yes, the step is OK, if not, the step is KO. This example is extracted from the test_batchlets.xml job that is a testsuite executing several testcase jobs.
(This is also a usage sample of the Batch restAPI proxy service provided in the context)

```
<step id="end">
    <batchlet ref="ossa.javascript">
        <properties>
            <property
                name="script"
                value="
                    var ok = true;
                    var steps = batch.getStepExecutions( job.executionId ).toArray();
                    for( var idx in steps ) {
                        var step = steps[idx];
                        if( step.stepName != 'end' ) {
                            if( step.exitStatus!='OK' ) {
                                ok = false;
                                log.info( step + ' not OK exit status : '+step.exitStatus);
                            }
                        }
                    }
                    if( ok ) { 'OK'; }
                    else 'KO';
                "/>
        </properties>
    </batchlet>
    <end on="OK" exit-status="OK"/>
    <fail on="*" exit-status="KO" />
</step>
```

2.2.9 Run Batch

2.2.9.1 ossa.batch batchlet overview

The batch batchlet allow integrators to run any new batch instance as a batch integration step.

The ExitStatus of the run batch is returned as the current step status.

The batch is executed synchronously. This batchlet is waiting for the run batch to be completed before continuing.

The batch batchlet accepts 3 parameters:
- the batch name to be run
- the batch input parameters as a Map<String, String> object
- the run batch timeout

2.2.9.2 ossa.batch batchlet interface

SUPPORT				
	Ref	ossa.batch		
	Logger	com.hp.ossa.batch.batchlet.utils.RunBatchBatchlet		
	Cancellable	No		
	Templated	Yes		
PROPERTIES				
	batch	Batch name to be executed synchroniously	req	String
	params	Batch input parameters	opt	Map<String,String>
	timeout	Batch timeout in ms. -1 means no timeout	opt	long
EXIT STATUS				
	[BatchExitStatus]	Value returned by the batch.		

Table 13 - ossa.batch batchlet interface

2.2.9.3 Sample usage

This example is extracted from :

/opt/ossa/repo-ossa/com.hp.ossa.test.batchlet/test_batchlets.xml

```
<step id="TestJobParameters" next="TestScripting-01">
    <batchlet ref="ossa.batch" >
        <properties>
            <property name="batch" value="TestJobParameters" />
            <property name="params" value="return=OK" />
        </properties>
    </batchlet>
</step>
```

2.2.10 Run System Process

2.2.10.1 ossa.run batchlet overview

The ossa.run batchlet allow integrator the run a system sh script as a job step.

This batchlet is cancellable. You can interrupt the process execution by stopping the job.

The exit status of the step is the process exit value. Usually an integer.

2.2.10.2 ossa.run batchlet interface

SUPPORT				
	Ref	ossa.run		
	Logger	com.hp.ossa.batch.batchlet.system.ProcessBatchlet		
	Cancellable	Yes		
	Templated	Yes		
PROPERTIES				
	command	Command to be executed by the underlying OS	req	String
	arguments	Command arguments. (Coma separator)	opt	[String]
	environment	Environnement variables for the process execution	opt	[String]
	baseDir	Base directory for process execution	opt	String
EXIT STATUS				
	[ProcessExitStatus]	Value returned by the system process.		
OUTPUT PROPERTIES				
	processOut	The process ouput stream		java.io.BufferedReader
	processErr	The process error stream		java.io.BufferedReader

Table 14 - ossa.run batchlet interface

2.2.10.3 Sample usage

```
<step id="exec" >
    <batchlet ref="ossa.run">
        <properties>
            <property name="command" value="/opt/ossa/bin/ossa_env.sh" />
        </properties>
    </batchlet>
    <next on="0" to="assert"/>
    <fail on="*" exit-status="KO"/>
</step>
```

2.3 Batch Job examples

HP OSS Analytics Foundation embeds some examples of OSSA batchlets. They are available here:

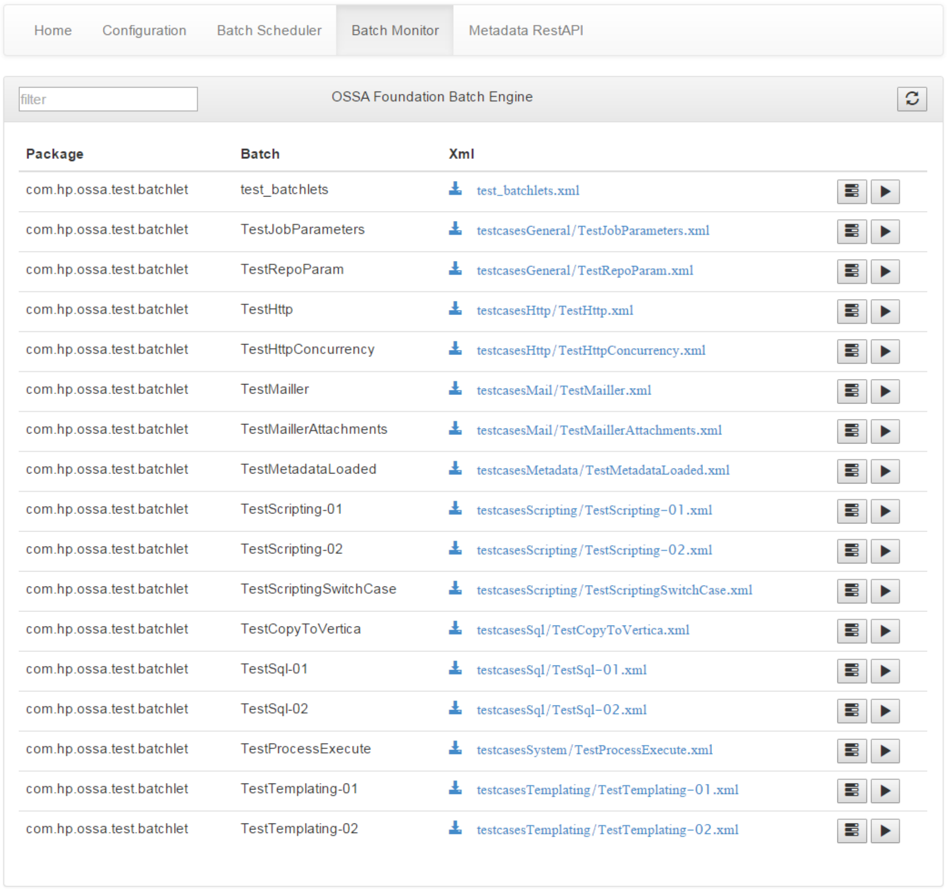
```
/opt/ossa/repo-ossa/com.hp.ossa.test.batchlet
```

There is one batch job (*test_batchlets.xml* / *BATCH_test_batchlets.json*) which allows to start all the other batch jobs defined in subdirectories.

If you want to load them into the OSSA Batch engine, execute:

ossa-repo.sh loadDirectory com.hp.ossa.test.batchlet /opt/ossa/repo-ossa/com.hp.ossa.test.batchlet

Once loaded, you should see the com.hp.ossa.test.batchlet batch available for manual execution (no job is scheduled by default in this package)



Package	Batch	Xml
com.hp.ossa.test.batchlet	test_batchlets	test_batchlets.xml
com.hp.ossa.test.batchlet	TestJobParameters	testcasesGeneral/TestJobParameters.xml
com.hp.ossa.test.batchlet	TestRepoParam	testcasesGeneral/TestRepoParam.xml
com.hp.ossa.test.batchlet	TestHttp	testcasesHttp/TestHttp.xml
com.hp.ossa.test.batchlet	TestHttpConcurrency	testcasesHttp/TestHttpConcurrency.xml
com.hp.ossa.test.batchlet	TestMailler	testcasesMail/TestMailler.xml
com.hp.ossa.test.batchlet	TestMaillerAttachments	testcasesMail/TestMaillerAttachments.xml
com.hp.ossa.test.batchlet	TestMetadataLoaded	testcasesMetadata/TestMetadataLoaded.xml
com.hp.ossa.test.batchlet	TestScripting-01	testcasesScripting/TestScripting-01.xml
com.hp.ossa.test.batchlet	TestScripting-02	testcasesScripting/TestScripting-02.xml
com.hp.ossa.test.batchlet	TestScriptingSwitchCase	testcasesScripting/TestScriptingSwitchCase.xml
com.hp.ossa.test.batchlet	TestCopyToVertica	testcasesSql/TestCopyToVertica.xml
com.hp.ossa.test.batchlet	TestSql-01	testcasesSql/TestSql-01.xml
com.hp.ossa.test.batchlet	TestSql-02	testcasesSql/TestSql-02.xml
com.hp.ossa.test.batchlet	TestProcessExecute	testcasesSystem/TestProcessExecute.xml
com.hp.ossa.test.batchlet	TestTemplating-01	testcasesTemplating/TestTemplating-01.xml
com.hp.ossa.test.batchlet	TestTemplating-02	testcasesTemplating/TestTemplating-02.xml

Figure 7 - Loaded test/demonstration package in repository

Test / Demonstration package content

DIRECTORY	JOB	DESCRIPTION
[ROOT]	test_batchlets.xml	This is the testsuite responsible to run all job testcase. It use the batch and javascript batchlets.
testcasesGeneral	TestJobParameters.xml	Testing job input parameters.
testcasesGeneral	TestRepoParam.xml	Test the configuration repository rest Api. It use a javascript test in order to generate fake data, and use http batchlet in order to call repository webservice functions.
testcasesHttp	TestHttp.xml	Testing usage of http batchlet. It use a javascript assertion step in order to verify response.
testcasesHttp	TestHttpConcurrency.xml	Testing multiple http request in parallel thanks to the Split batch service.
testcasesMail	TestMailler.xml	Send a simple mail.
testcasesMail	TestMaillerAttachments.xml	Send a mail with a downloaded resource as attachment with external content generation template (freemarker).
testcasesMetadata	TestMetadataLoaded.xml	Simply verify that metadata service is accessible to /ossa/packages url.
testcasesScripting	TestScripting-01.xml	Basic java scripting demonstration with embedded script.
testcasesScripting	TestScripting-02.xml	Basic java scripting demonstration with externalized script.
testcasesSql	TestSql-01.xml	Basic demonstration of sql batchlet functions with validation java script.
testcasesSql	TestSql-02.xml	More complex sql batchlet example
testcasesSystem	TestProcessExecute.xml	Basic demonstration of system process execution
testcasesTemplating	TestTemplating-01.xml	Basic templating usage rather than JSR-352 standard placeholding
testcasesTemplating	TestTemplating-02.xml	Externalized templating sample

Table 15 - Test package overview