OSS Analytics Foundation

Integration Guide

Version 1.1.4

Edition 1.0 - December 2016

Hewlett Packard
Enterprise

Notices

Legal notice

Printed in the US

Trademarks

Contents

List of tables

List of figures

Preface

About this guide

This guide describes how to install, configure, administrate and troubleshoot the HPE OSS Analytics Foundation software component.

Software component name: HPE OSS Analytics Foundation
Software component version: 1.1.4
Kit name: `ossa-server-1.1.4-MP.noarch.rpm`

Audience

This installation and configuration guide is for anyone who is responsible for configuring or integrating the HPE OSS Analytics Foundation.

Software versions

The terms Unix and Linux are used as a generic reference to the operating system, unless otherwise specified. The software versions referred to in this document are as follows:

Table 1: Software versions

Product version	Supported operating systems
HPE OSS Analytics Foundation version 1.1.4	Red Hat Enterprise Linux Server release 6.8
HPE Vertica version 7.2.3	Red Hat Enterprise Linux Server release 6.8

Typographical Conventions

`Courier` Font:

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

Italic Text:

- Filenames, 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

The following documents contain useful reference information:

HPE OSS Analytics Foundation Release Notes
HPE OSS Analytics Foundation Installation, Configuration and Administration Guide

Support

Please visit our HPE Software Support Online Web site at https://softwaresupport.hpe.com for contact information, and details about HPE Software products, services, and support.

The Software support area of the web site includes the following:

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

Chapter 1
Product overview

1.1 OSS Analytics Foundation introduction

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

1.2 OSSA Server architecture

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

Chapter 2
OSSA Data Mart REST API and Metadata

OSSA Server provides a Data Mart REST API to query database data. Database data should be organized as a data-mart, following a standard star schema (https://en.wikipedia.org/wiki/Star_schema). This is the standard and most efficient way of storing your analytics data, having specific facts and dimensions tables.
The OSSA batch system provides you some facilities (transformation functions) in order to match this standard analytics schema.
But, in case your solution relies on data types which may vary, and your schema can not match this standard way of organizing data because you handle generic facts and dimensions, you could consider using the generic/schema-less datamart.

Some examples of both types of schemas and associated metadata are given in this chapter.

What is "OSSA metadata" ?
Metadata provides an abstraction layer that allows one to compose simple URL to query database data.

2.1 Example of database schema for a standard data mart

In order to explain the concepts related to data mart schema, metadata and REST API, we take as example a simple data mart schema named "SCH1".
DIM_D1 is a table defining a natural dimension.
FCT_15MN, FCT_HOURLY, ... defined fact tables for all desired time granularities.
F1, F2 are fact columns, DDC1/2/3/4 are degenerated dimension columns (semantics is dimension data, but data is present as fact table column and externalized as a natural dimension data).

```
CREATE SCHEMA SCH1;

DROP TABLE IF EXISTS SCH1.DIM D1;
CREATE TABLE SCH1.DIM D1 (ID INTEGER, DC1 INTEGER, DC2 VARCHAR, DC3 FLOAT, DC4 TIMESTAMP);

CREATE TABLE SCH1.FCT_15MN (TIME TIMESTAMP, F1 INTEGER, F2 FLOAT, DIM_D1_ID INTEGER, DDC1 INTEGER, DDC2
       VARCHAR);
CREATE TABLE SCH1.FCT_HOURLY (TIME TIMESTAMP, F1 INTEGER, F2 FLOAT, DIM_D1_ID INTEGER, DDC1 INTEGER,
       DDC2 VARCHAR);
CREATE TABLE SCH1.FCT_DAILY (TIME TIMESTAMP, F1 INTEGER, F2 FLOAT, DIM_D1_ID INTEGER, DDC1 INTEGER,
       DDC2 VARCHAR);
CREATE TABLE SCH1.FCT_WEEKLY (TIME TIMESTAMP, F1 INTEGER, F2 FLOAT, DIM_D1_ID INTEGER, DDC1 INTEGER,
       DDC2 VARCHAR);
CREATE TABLE SCH1.FCT MONTHLY (TIME TIMESTAMP, F1 INTEGER, F2 FLOAT, DIM D1 ID INTEGER, DDC1 INTEGER,
       DDC2 VARCHAR);
```

2.2 Example of database schema for a generic/schema-less datamart

Unlike the way in section 2.1 to define dimensions and fact columns in several tables by categories, there's a way to define all dimensions in a unique table and all fact columns in a unique table: this way is called generic/schema-less data mart.
The GDIM is an example for generic dimension table, and the GFCT is an example for generic fact table.

```
CREATE SCHEMA SCH1;

DROP TABLE IF EXISTS SCH1.GDIM;
DROP TABLE IF EXISTS SCH1.GFCT;
```

```
CREATE TABLE SCH1.GDIM (
        ID INTEGER NOT NULL,
        TYPE VARCHAR(100) NOT NULL,
        ATTR_NAME VARCHAR(100) NOT NULL,
        VAL FLOAT FLOAT,
        VAL STR VARCHAR(1000),
        VAL TSTAMP Timestamp,
        VAL_INT INTEGER
);

ALTER TABLE SCH1.GDIM ADD CONSTRAINT PK GDIM PRIMARY KEY (ID, ATTR NAME, TYPE);

CREATE TABLE SCH1.GFCT (
        TSTAMP Timestamp NOT NULL,
        OBJID1 VARCHAR(100) NOT NULL,
        OBJID2 VARCHAR(100) NOT NULL,
        TYPE VARCHAR(100) NOT NULL,
        ATTR NAME VARCHAR(100) NOT NULL,
        VAL_FLOAT FLOAT,
        VAL_STR VARCHAR(1000),
        VAL_TSTAMP Timestamp,
        VAL INT INTEGER
);

ALTER TABLE SCH1.GFCT ADD CONSTRAINT PK_GFCT PRIMARY KEY (TSTAMP, ATTR_NAME, TYPE, OBJID1, OBJID2);
```

Full key in GDIM table is the composite of TYPE and ID. The TYPE column indicates different categories of dimensions, and the ID column is the identifier for each category of dimensions. Take the DIM_D1 table in the section 2.1 for example. For each record in the DIM_D1 table, it can be equivalently saved in GDIM table with 4 records of the same TYPE and ID value. The TYPE column of GDIM table can use the value "D1". The DC1 column of DIM_D1 table can be saved in VAL_INT column of GDIM table. The DC2 column of DIM_D1 table can be saved in VAL_STR column of GDIM table. The DC3 column of DIM_D1 table can be saved in VAL_FLOAT column of GDIM table. The DC4 column of DIM_D1 table can be saved in VAL_TSTAMP column of GDIM table.

The GFCT table saves data in a similar way as the GDIM table. The difference are that GFCT table has a TSTAMP column of timestamp type to indicate the timestamp of the data, and the identifier for GFCT can be the composite of several columns. In the above example, the full key in GFCT table is the composite of TSTAMP, TYPE, OBJID1 and OBJID2. Take the FCT_15MN table in the section 2.1 for example. For each record in the FCT_15MN table, it can be equivalently saved in the GFCT table with 5 records of the same TYPE, TSTAMP, OBJID1 and OBJID2 value. The TYPE column of GFCT table can use the value "FCT_15MN". The time column of FCT_15MN table can be saved in the TSTAMP of GFCT table. Other columns of FCT_15MN table can be save in the VAL_XXX columns in GFCT table depending on the data types of the columns. As the FCT_15MN has no specific primary key, in the GFCT table, OBJID1 can be written with a sequence value for each record in the FCT_15MN table, and OBJID2 can be filled with NULL.

2.3 Package

```
<?xml version="1.0" encoding="UTF-8"?>
<OSSAPackage id="pkg1" displayName="Example package 1"
             version="1.0" xsi:noNamespaceSchemaLocation="OSSA_packageDesc.xsd"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>This is an example package.
   It tries to illustrate various aspects of OSSA Server RESTAPI metadata.
   Any URL to query data in this package will start with:
   http://127.0.0.1:8080/ossa/packages/pkg1
</Description>
```

Here is the URL to list all packages:

http://127.0.0.1:8080/ossa/packages

The URL to get package definition for "pkg1" is:

http://127.0.0.1:8080/ossa/packages/pkg1

2.4 Natural Dimension

DIM_D1 is a natural dimension, its surrogate key column is column ID.

```
  <Dims>
    <Dim id="D1" displayName="Dimension D1" schemaName="SCH1" tableName="DIM_D1">
      <Description>Natural dimension D1</Description>
      <DimCol type="Integer"   displayName="Dimension DIM D1 Column1" colName="DC1"
        lowCardinalityFlag="false" />
      <DimCol type="String"    displayName="Dimension DIM D1 Column2" colName="DC2"
        lowCardinalityFlag="false" />
      <DimCol type="Float"     displayName="Dimension DIM_D1 Column3" colName="DC3"
        lowCardinalityFlag="false" />
      <DimCol type="Timestamp" displayName="Dimension DIM D1 Column4" colName="DC4"
        lowCardinalityFlag="false" />
      <SurrogateKey colName="ID" />
    </Dim>
  </Dims>
```

2.5 Degenerated Dimension

```
  <DegeneratedDims>
    <DegeneratedDim id="DD1" displayName="Degenerated dimension DD1">
      <Description>Degenerated dimension DD1</Description>
      <DimCol type="Integer"   displayName="DDim1 Column1" colName="DDC1" lowCardinalityFlag="false" />
      <DimCol type="String"    displayName="DDim1 Column2" colName="DDC2" lowCardinalityFlag="false" />
    </DegeneratedDim>
  </DegeneratedDims>
```

2.6 Generic Dimension for your generic/schema-less datamart

The natural dimension DIM_D1 can be defined as a generic dimension as follows. The differences are:

1. The table referred is changed to GDIM table.
2. The column type of SurrogateKey should be provided.
3. There's a new element Generic referred to another GenericDim element with the ID, which defines how to use the GDIM table.
4. The typeName attribute of Generic element is optional. It will be used as the TYPE value when looking up data in GDIM table. If it's not provided, the id attribute of Dim element will be used, instead.

```
  <Dims>
    <Dim id="D1" displayName="Dimension D1" schemaName="SCH1" tableName="GDIM">
      <Description>Natural dimension D1</Description>
      <DimCol type="Integer"   displayName="Dimension DIM D1 Column1" colName="DC1"
        lowCardinalityFlag="false" />
      <DimCol type="String"    displayName="Dimension DIM D1 Column2" colName="DC2"
        lowCardinalityFlag="false" />
      <DimCol type="Float"     displayName="Dimension DIM_D1 Column3" colName="DC3"
        lowCardinalityFlag="false" />
      <DimCol type="Timestamp" displayName="Dimension DIM D1 Column4" colName="DC4"
        lowCardinalityFlag="false" />
      <SurrogateKey colName="ID" colType="Integer" />
      <Generic genDimIdRef="myGenDim" typeName="D1" />
    </Dim>
  </Dims>
… …
  <GenericDims>
      <GenericDim id="myGenDim" typeColName="TYPE" attrColName="ATTR_NAME" valStrColName="VAL_STR"
        valIntColName="VAL_INT" valFloatColName="VAL_FLOAT" valTstampColName="VAL_TSTAMP"/>
  </GenericDims>
```

2.7 Facts

```
<Facts>
  <Fact id = "F1" displayName="Fact F1" type="Integer" unit="#" />
  <Fact id = "F2" displayName="Fact F2" type="Float" unit="" />
  <Fact id = "F3" displayName="Fact F3 (synthetic based of F1, f2)" type="Float" unit="" />
</Facts>
```

2.8 Fact Pattern

The fact pattern is composed by all the previous items, fact calculation columns, degenerated dimension, the data sources fact tables with all the time granularities supported and all the natural dimensions referenced by each fact table.

F3 is a synthetic fact calculation column, i.e. a column calculated on demand as soon as it is requested in URL.

```
<FactPattern id="FP1">
   <FactCalculation factIdRef="F1" colName="F1">
     <MultiRowSQLExpression>sum(F1)</MultiRowSQLExpression>
   </FactCalculation>

   <FactCalculation factIdRef="F2" colName="F2">
     <MultiRowSQLExpression>sum(F2)</MultiRowSQLExpression>
   </FactCalculation>

   <FactCalculation factIdRef="F3">
     <RowSQLExpression>F2/F1</RowSQLExpression>
     <MultiRowSQLExpression>sum(F2)/sum(F1)</MultiRowSQLExpression>
   </FactCalculation>

   <DegeneratedDim ddimIdRef="DD1"/>

   <DataSources>
     <DataSource schema="SCH1" table="FCT 15MN" timestampCol="TIME" timePeriodDuration="15" />
     <DataSource schema="SCH1" table="FCT HOURLY" timestampCol="TIME" timePeriodDuration="60" />
     <DataSource schema="SCH1" table="FCT DAILY" timestampCol="TIME" timePeriodDuration="1440" />
     <DataSource schema="SCH1" table="FCT_WEEKLY" timestampCol="TIME" timePeriodDuration="10080" />
     <DataSource schema="SCH1" table="FCT_MONTHLY" timestampCol="TIME" timePeriodDuration="43200" />
   </DataSources>

   <Joins>
     <Join joinType="InnerJoin" dimIdRef="D1" fkColInFactTable="DIM_D1_ID" />
   </Joins>
 </FactPattern>
```

2.9 Fact Pattern for Generic Fact for your generic/schema-less datamart

For generic facts, the definition of Fact and FactCalculation elements is the same as normal facts. The difference is the definition of the DataSource elements. In such case, only a data source with one timePeriodDuration is supported. Several data sources with different timePeriodDuration can't be defined. Besides, the DataSource element has a new child element Generic referred to another GenericFactPattern element which defines how to use the GFCT table. Here, the typeColValue attribute of Generic element is optional and will be used as the TYPE value when looking up data in GFCT table. If it's not provided, the id attribute of FactPattern element should match the value of TYPE column in the generic Fact table.

```
<FactPattern id="FP1">
   <FactCalculation factIdRef="F1" colName="F1">
     <MultiRowSQLExpression>sum(F1)</MultiRowSQLExpression>
   </FactCalculation>

   <FactCalculation factIdRef="F2" colName="F2">
     <MultiRowSQLExpression>sum(F2)</MultiRowSQLExpression>
   </FactCalculation>

   <FactCalculation factIdRef="F3">
     <RowSQLExpression>F2/F1</RowSQLExpression>
```

```
        <MultiRowSQLExpression>sum(F2)/sum(F1)</MultiRowSQLExpression>
      </FactCalculation>

      <DegeneratedDim ddimIdRef="DD1"/>

      <DataSources>
        <DataSource timestampCol="TSTAMP" table="GFCT" schema="SCH1" timePeriodDuration="15">
               <Generic genFactPatternIdRef="myGenFactPattern" typeColValue="FP1" />
        </DataSource>
      </DataSources>

      <Joins>
        <Join joinType="InnerJoin" dimIdRef="D1" fkColInFactTable="DIM_D1_ID" />
      </Joins>
    </FactPattern>
… …
   <GenericFactPatterns>
       <GenericFactPattern id="myGenFactPattern" typeColName="TYPE" attrColName="ATTR NAME"
       valStrColName="VAL_STR" valIntColName="VAL_INT" valFloatColName="VAL_FLOAT"
       valTstampColName="VAL_TSTAMP">
               <ObjIdColNames>
                      <ObjIdColName>OBJID1</ObjIdColName>
                      <ObjIdColName>OBJID2</ObjIdColName>
               </ObjIdColNames>
       </GenericFactPattern>
   </GenericFactPatterns>
```

2.10 URL examples and select queries

2.10.1 DIM values

Get dimension column values for DIM_D1.DC2

```
curl http://127.0.0.1:8080/ossa/packages/pkg1/dims/DC2/distinct
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/dims/DC2/distinct",
  "queryStr" : "SELECT SCH1.DIM_D1.DC2 AS DC2\nFROM SCH1.DIM_D1\nGROUP BY DC2",
  "status" : "OK",
  "headers" : [ "DC2" ],
  "values" : [ [ "V2" ], [ "V3" ], [ "V1" ] ]
}
```

Get dimension column values for DIM_D1.DC2. Filtering on DIM_D1.DC1
(2 <= DIM_D1.DC1 < 3)

```
curl \
http://127.0.0.1:8080/ossa/packages/pkg1/dims/DC2/distinct?wheredim=and,DC1,ge,2,DC1,lt,3
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/dims/DC2/distinct?wheredim=and,DC1,ge,2,DC1,lt,3",
  "queryStr" : "SELECT SCH1.DIM D1.DC2 AS DC2\nFROM SCH1.DIM D1\nWHERE ( (  DC1 >= 2 ) AND (  DC1 < 3 )
       )
                GROUP BY DC2",
  "status" : "OK",
  "headers" : [ "DC2" ],
  "values" : [ [ "V2" ] ]
}
```

2.10.2 Degenerated DIM values

Get degenerated dimension values for column DDC2.
Note that in case of degenerated dimension, a fact table has to be queried, it is always the table with largest granularity that is targeted. Here, it is FCT_MONTHLY.

```
curl http://127.0.0.1:8080/ossa/packages/pkg1/dims/DDC2/distinct
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/dims/DDC2/distinct",
  "queryStr" : "SELECT SCH1.FCT_MONTHLY.DDC2 AS DDC2\nFROM SCH1.FCT_MONTHLY\nGROUP BY DDC2",
  "status" : "OK",
  "headers" : [ "DDC2" ],
```

```
  "values" : [ [ "DDC1V9" ], [ "DDC1V3" ], [ "DDC1V4" ], [ "DDC1V10" ], [ "DDC1V1" ], [ "DDC1V5" ], [
       "DDC1V6" ], [ "DDC1V2" ], [ "DDC1V8" ], [ "DDC1V7" ] ],
  "nbVals" : 10
}
```

2.10.3 Fact values

Get fact column F1 along with natural dimension column DC2, for given time window. Granularity given is 15mn, so query is targeted to table SCH1.FCT_15MN.

```
curl \
http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/dims/DC2/timewindow/20151027-000000/20151028-
000000/?granularity=15
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/dims/DC2/timewindow/20151027-
       000000/20151028-000000/?granularity=15",
  "queryStr" : "SELECT SCH1.FCT_15MN.TIME AS TIME, DIM_D1.DC2 AS DC2, sum(F1) AS F1 FROM SCH1.FCT_15MN
       INNER JOIN SCH1.DIM_D1 DIM_D1 ON  DIM_D1.ID = SCH1.FCT_15MN.DIM_D1_ID WHERE (SCH1.FCT_15MN.TIME
       >= TO TIMESTAMP('2015-10-27 00:00:00', 'YYYY-MM-DD HH24:MI:SS'))  AND (SCH1.FCT 15MN.TIME <
       TO TIMESTAMP('2015-10-28 00:00:00', 'YYYY-MM-DD HH24:MI:SS')) GROUP BY TIME, DC2 ORDER BY TIME
       ASC",
  "status" : "OK",
  "tstampColName" : "TIME",
  "headers" : [ "TIME", "DC2", "F1" ],
  "values" : [ [ 1445940000000, "V1", 10 ], [ 1445940900000, "V2", 20 ],
               [ 1445941800000, "V3", 30 ], [ 1445944500000, "V1", 40 ],
               [ 1445945400000, "V2", 30 ], [ 1445946300000, "V3", 20 ] ],
  "nbVals" : 6
}
```

Same query with granularity 60mn, so generated query is targeted to table SCH1.FCT_HOURLY.

```
curl \
       http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/dims/DC2/timewindow/20151027-000000/20151028-
       000000?granularity=60
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/dims/DC2/timewindow/20151027-
       000000/20151028-000000?granularity=60",
  "queryStr" : "SELECT SCH1.FCT HOURLY.TIME AS TIME, DIM D1.DC2 AS DC2, sum(F1) AS F1 FROM
       SCH1.FCT_HOURLY INNER JOIN SCH1.DIM_D1 DIM_D1 ON  DIM_D1.ID = SCH1.FCT_HOURLY.DIM_D1_ID WHERE
       (SCH1.FCT_HOURLY.TIME >= TO_TIMESTAMP('2015-10-27 00:00:00', 'YYYY-MM-DD HH24:MI:SS'))  AND
       (SCH1.FCT_HOURLY.TIME < TO_TIMESTAMP('2015-10-28 00:00:00', 'YYYY-MM-DD HH24:MI:SS')) GROUP BY
       TIME, DC2 ORDER BY TIME ASC",
  "status" : "OK",
  "tstampColName" : "TIME",
  "headers" : [ "TIME", "DC2", "F1" ],
  "values" : [ [ 1445925600000, "V3", 20 ], [ 1445929200000, "V2", 30 ], [ 1445932800000, "V1", 40 ], [
       1445943600000, "V3", 30 ], [ 1445947200000, "V2", 20 ], [ 1445950800000, "V1", 10 ] ],
  "nbVals" : 6
}
```

A more complete example with more fact column, including synthetic column F3, and OR condition on natural dimension column DC2.

```
curl 'http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/F2/F3/dims/DC2/timewindow/20151027-
       000000/20151028-000000?wheredim=or,DC2,lt,2,DC2,gt,3&granularity=60'
{
  "url" : "http://127.0.0.1:8080/ossa/packages/pkg1/facts/F1/F2/F3/dims/DC2/timewindow/20151027-
       000000/20151028-000000?wheredim=or,DC2,lt,2,DC2,gt,3&granularity=60",
  "queryStr" : "SELECT SCH1.FCT HOURLY.TIME AS TIME, DIM D1.DC2 AS DC2, sum(F1) AS F1, sum(F2) AS F2,
       sum(F2)/sum(F1) AS F3 FROM SCH1.FCT_HOURLY INNER JOIN SCH1.DIM_D1 DIM_D1 ON  DIM_D1.ID =
       SCH1.FCT_HOURLY.DIM_D1_ID WHERE (SCH1.FCT_HOURLY.TIME >= TO_TIMESTAMP('2015-10-27 00:00:00',
       'YYYY-MM-DD HH24:MI:SS'))  AND (SCH1.FCT_HOURLY.TIME < TO_TIMESTAMP('2015-10-28 00:00:00',
       'YYYY-MM-DD HH24:MI:SS') AND  (  DIM D1.DC2 < '2' ) OR (  DIM D1.DC2 > '3' ) ) GROUP BY TIME,
       DC2 ORDER BY TIME ASC",
  "status" : "OK",
  "tstampColName" : "TIME",
  "headers" : [ "TIME", "DC2", "F1", "F2", "F3" ],
  "values" : [ [ 1445925600000, "V3", 20, 80.0, 4.0 ], [ 1445929200000, "V2", 30, 70.0,
       2.33333333333333 ],
            [ 1445932800000, "V1", 40, 60.0, 1.5 ], [ 1445943600000, "V3", 30, 30.0, 1.0 ],
            [ 1445947200000, "V2", 20, 20.0, 1.0 ], [ 1445950800000, "V1", 10, 10.0, 1.0 ] ],
  "nbVals" : 6
}
```

Chapter 3
OSSA Batch System

3.1 Setup Scheduled Batch Jobs

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

A batch job is a valid JSR-352 XML file which defines its processing (for more information please refer to section 3.2 Batch job definition features).

CAUTION: Within this .xml file describing the batch job processing, the **`job id` field** must be in the form:

```
job id="<batchPackage>/<batchName>"
```

The batch package name will then be used when loading the batch, and running it.

Moreover, for each batch you want load on the system, you must create a dedicated JSON file responsible for declaring the batch file and its schedule.

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

Here also, in this .json file, **the job must be identified by its package: `package` and `jobXml` fields must be used.**
Example:

```
"packageName" : "mypackage",
"jobXml" : "mybatch.xml"
```

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

The *BATCH_xxx.json* file contains the reference to the batch job Xml file and the description of the execution schedule required.

Here are details about each parameters for a batch job configuration defined in a *BATCH_xxx Json* files:

Table 2: Batch configuration parameters

Parameter	Type	Description
jobXml	String	This is the repository parameter entry name that contains the job xml definition. Note that the batch Xml definition should be stored in the same package than its json setup file.
jobParameters	Map<String,String>	A Key-Value Json object defining job input parameters
adminState	*Locked* | *Unlocked*	*Locked* : the batch is not scheduled *Unlocked*: the batch is scheduled according to the value of *batchSchedule* parameter
batchSchedule	java.ejb.Schedule	A Json object defining the batch scheduler. See section batch scheduler for more details

NOTE: 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 behavior, you can set the *concurrentFlag* parameter value to *true* within the *jobParameters*.
Here is an example of *Batch_xxx.json*

```
{
 "jobXml" : "…",
 "jobParameters" : {
       "concurrentFlag" : "true"
  }
 "adminState" : "…",
"batchSchedule" : { …}
}
```

In that case, multiple executions of the job can be run in parallel.

3.1.1 Batch identifier (job id)

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*.

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

Example: in ***summJob****.xml* file:

```
<job id="SUMMJOBTEST/summJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
```

3.1.2 Batch system scheduler

The batch system scheduler is built on top of standard JEE 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:

Table 3: Calendar based scheduler attributes

Parameter	Type	Description
jobXmlPath	String	This is the repository parameter entry name that contains the job xml definition. Note that the batch Xml definition should be stored in the same package than its json setup file.
jobParameters	Map<String,String>	A Key-Value Json object defining job input parameters
adminState	*Locked* | *Unlocked*	*Locked* : the batch is not scheduled *Unlocked*: the batch is scheduled according to the value of *batchSchedule* parameter
batchSchedule	java.ejb.Schedule	A Json object defining the batch scheduler. See section batch scheduler for more details

3.1.3 Default Batch Job parameters

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

Table 4: Default batch job input parameters details

Input parameter	Type	Description
Host	Text	The host name of the batch runner for the job execution
Port	Int	The host http port of the batch runner for the job execution
runner-uuid	Text	Unique identifier of the batch runner. This UUID is unique for each started HP OSS Analytics Foundation framework.
packageName	Text	The repository package name where the job definition is coming from

Once the job is started, you can see these input parameter values directly in the OSSA Server Admin console in the Batch Monitor screen by clicking on the 'Status' button of the job execution.

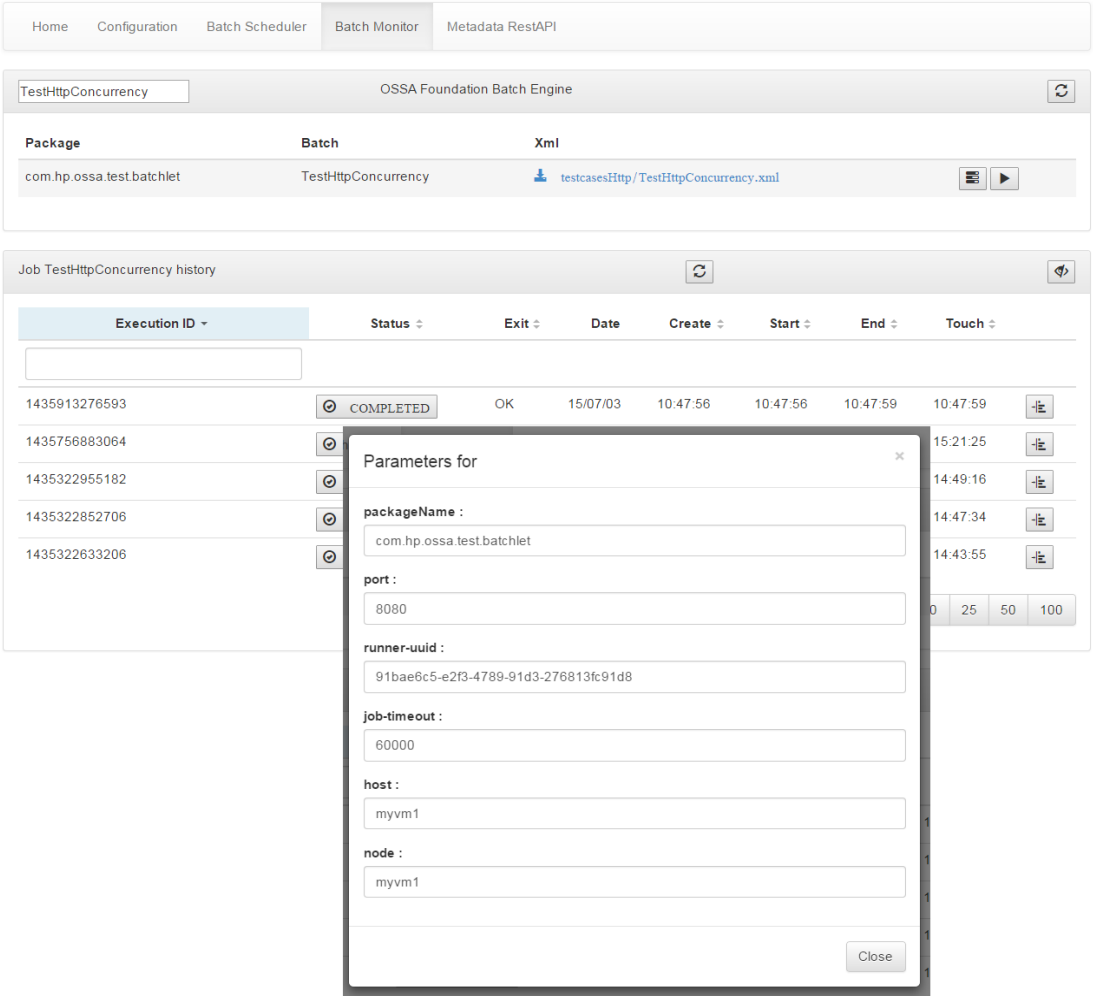


Figure 1: Default job input parameter from OSSA Server Admin Console

3.2 Batch job definition features

Job processing is defined in XML following the standard Job Specification Language (JSL) 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

CAUTION: Your batch job definition (XML file) will potentially contain some commands (in javascript, SQL ...) but remember that this definition file follows the XML standard, so:
- any "insignificant" whitespace characters are not preserved
- you cannot use > or < or & or " characters. If you need them, you have to use the corresponding html character:
  > < & &quote;

3.2.1 Job XML Substitution

As part of the JSL (Job specification language, relying on JSR-352), one interesting feature is the "substitution". Job XML supports substitution as part of any attribute value.

A substitution expression may reference a job parameter or a job property or a system property by specifying the name of the parameter or property through a substitution expression operator.

```
#{jobProperties['<propertyName>']}
#{jobParameters['<paramName>']}
#{systemProperties['<systPropName>']}
```

Here is a simple example of substitution expression for a job property:

```
<property name="myfilename" value="myfile.txt" />
<property name="tmpfilename" value="#{jobProperties['myfilename']}.tmp" />
```

CAUTION: Substitution expressions are processed at job start.
So, they cannot deal with "dynamic" values. For such dynamic handling, please refer to OSSAF templating feature in the next sections.

3.2.2 Properties inheritance

The JSR-352 standard allows integrators to define properties at different level in a Job XML definition.
Inheritance is possible for properties defined at Job, Step and Batchlet level.

At runtime, if a property is not defined at the batchlet level, the JEE batch runtime environment will try to find the property value at the Step level, the Job level and finally as input Job parameters level.

Let's take a concrete example : the *ossa.Sql* batchlet need a *'datasource'* property. If the integration process has several *ossa.Sql* steps, it's not needed 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
<?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>
```

Inherited property

Figure 2: Batchlet properties inheritance example

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'*. Its value is used by the two steps.

3.2.3 Externalized property values

Batch property values can be externalized to a file. This can be very useful when batchlet property values are quite long or if the integrator want to preserve indentation of the property value.

Let's take an example to understand the benefit of such externalized properties.
Here we define a property which value corresponds to a java script (that could be referenced in 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. But, when the system reads the value, it will get it on one line.

This can be problematic to have such processing; when handling comments for example.

When you are writing a java script directly in the XML value attribute, **you must not use the comment character //** because the carriage return character will not be taken into account during execution. Note that the **same issue exists with SQL comments '--'**.

So, if you need to preserve indentation or simply want to separate concerns, you can use this feature '**externalized properties**', which is supported by all OSSA batchlets.

Within a job xml file, the values can be externalized in a file by using the following syntax:

```
Value="[[file]]"
```

In the following example, the javascript is externalized to a *TestScripting-02.js* file.

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="TestScripting-02" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
  <!-- Testing javascript batchlet -->
  <properties>
    <property name="verbose" value="false" />
  </properties>

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

In this way, you have no indentation limitation nor forbidden characters and the integration flow logic is separated from the 'business' process implementation defined in java script.

3.2.4 Transient and persistent context

Two different contexts are available during the execution of a batch job and allows data manipulation or sharing between steps or processes:

- transient context
- persistent context

3.2.4.1 Transient context

The transient context is shared by all steps of a job execution. It is created at job startup time and destroyed when the job is stopped, abandoned or completed.

This context contains **job transient user data** as a List of *Map<Key,Value>* where integrators can put or get any kind of objects with a corresponding name. This object is directly added to the templating context with the name '**`data`**'.
You can access the value within your batch job xml with: **`${data['<stepName>.myParam']}`**
You can also handle it in your java scripts with:

- `data.put('myParam','myValue');` and
- `data.get('<stepName>.myParam');`

From Sql batchlets transient user data can be accessed using: `DATA(key) sql statement`
For more details, please refer to the related *ossa.javscript*, *ossa.sql batchlets* descriptions below.

3.2.4.2 Persistent context

A job can also handle data from a persistent context (objects are stored in the database, in the *STEP_EXECUTION* table). With OSSA Batch system you can access it through the usage of thejava script ***step*** variable (*step.getPersistentUserData( ), step.setPersistentUserData(…)*).
Step variable usage is defined in further sections.

3.2.5 Properties templating

As already seen, JSR-352 JSL supports substitution as part of any property value. But this substitution is done at batch creation time, so this cannot deal with dynamic values, transient or persistent data for injecting values between steps.

That is why the OSSA Batch system provides a '**properties templating**' capability.

3.2.5.1 Simple properties templating

The template evaluation is done just before the usage of the property. With this, you can inject loaded values directly between job steps. Here is an 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 saves it as transient user data with the name *'generatedId'*.
In the last statement, another insert, we use the templating feature with the placeholder *${data.generatedId}*.

This is only possible because **templated properties are evaluated** at runtime.

3.2.5.2 Conditional processing with properties templating

The templating can also be used to generate different property values depending on 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 3: Generate dynamic property value with template

Here, depending on the *'targetType'* property value, we need to have different *OrderBy* part on the executed SQL statement. We can use templating facilities for that:

```
-- 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

NOTE: FreeMarker comes with a lot of functions to manipulate data. It proposes many facilities for conditional processing or formatting. Please refer to the official FreeMarker documentation for more details about capacities. http://freemarker.incubator.apache.org/index.html

3.2.5.3 Templating context

Using the properties templating one benefits from native contextual objects available in templates.

Table 5: contextual objects available in templates

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 OSSA batch bean service	com.hp.ossa.batch.BatchIfBase

3.3 OSSA Batchlet Library

The OSS Analytics Foundation provides an OSSA Batchlet library which can be used for transformation purposes. This can serve to customize your OSS Analytics solution.

Several kind of batchlets are defined in this library; some of them are presented below.

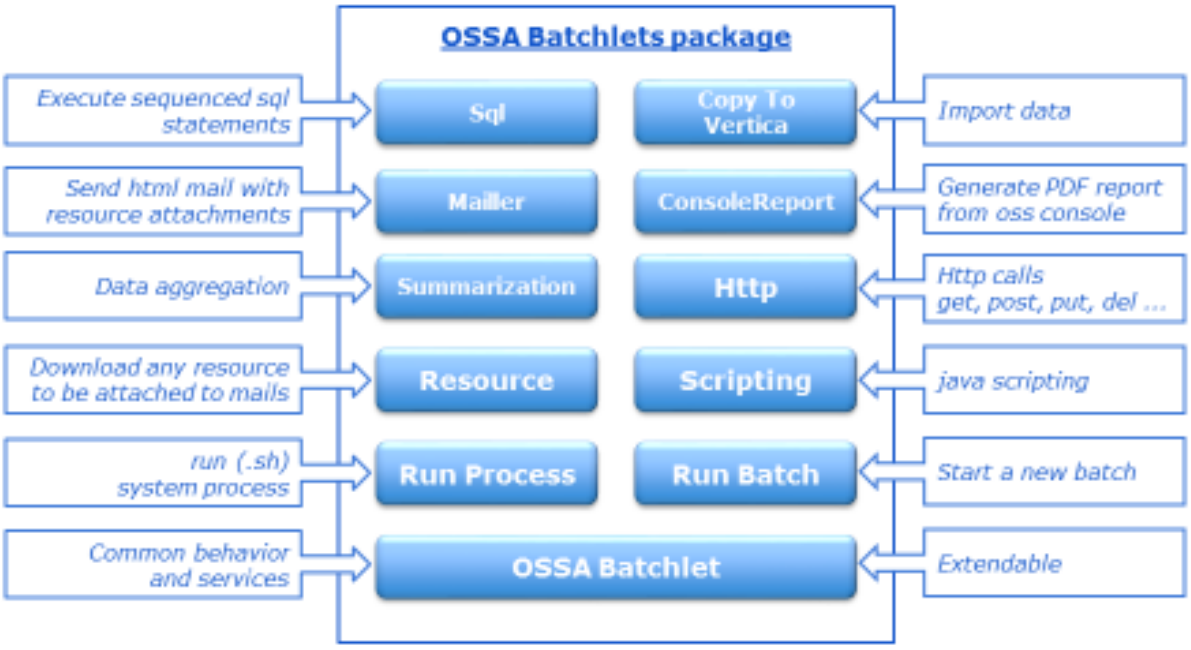


Figure 4: OSSA Batchlet Library

You will also find in the next sections, the description of the *Transformation* batchlet and *SqlCDC* batchlet.

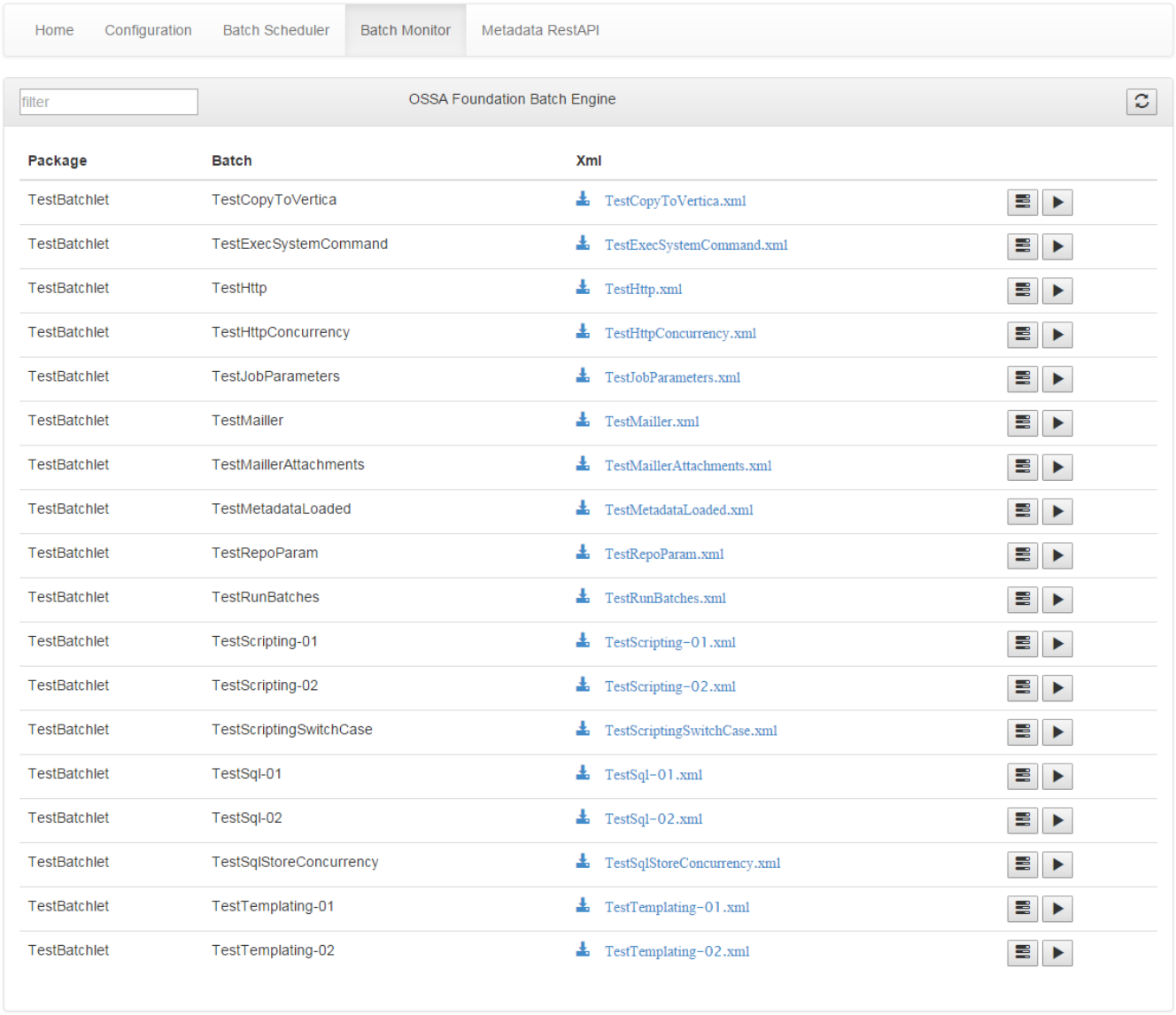
3.3.1 Batch job examples

All OSSA batchlets that are presented here have some usage examples in the OSSA installation directory. Those job examples are located here: *${OSSA_HOME}/repo-ossa/test-batchlet/*

If you want to load all the job examples into the OSSA Batch system, you can execute:

```
${OSSA_HOME}/bin/ossa-repo.sh loadDirectory TestBatchlet ${OSSA_HOME}/repo-ossa/test-batchlet
${OSSA_HOME}/bin/ossa-repo.sh reload
${OSSA_HOME}/bin/ossa-batch.sh reload
```

Once loaded, you can see in the OSSA Admin Console, the package *TestBatchlet* containing all the job examples available for manual execution (no job is scheduled by default in this package).
Note that the job *'test_batchlets.xml'* allows one to start all the other jobs.



Package	Batch	Xml
TestBatchlet	TestCopyToVertica	TestCopyToVertica.xml
TestBatchlet	TestExecSystemCommand	TestExecSystemCommand.xml
TestBatchlet	TestHttp	TestHttp.xml
TestBatchlet	TestHttpConcurrency	TestHttpConcurrency.xml
TestBatchlet	TestJobParameters	TestJobParameters.xml
TestBatchlet	TestMailler	TestMailler.xml
TestBatchlet	TestMaillerAttachments	TestMaillerAttachments.xml
TestBatchlet	TestMetadataLoaded	TestMetadataLoaded.xml
TestBatchlet	TestRepoParam	TestRepoParam.xml
TestBatchlet	TestRunBatches	TestRunBatches.xml
TestBatchlet	TestScripting-01	TestScripting-01.xml
TestBatchlet	TestScripting-02	TestScripting-02.xml
TestBatchlet	TestScriptingSwitchCase	TestScriptingSwitchCase.xml
TestBatchlet	TestSql-01	TestSql-01.xml
TestBatchlet	TestSql-02	TestSql-02.xml
TestBatchlet	TestSqlStoreConcurrency	TestSqlStoreConcurrency.xml
TestBatchlet	TestTemplating-01	TestTemplating-01.xml
TestBatchlet	TestTemplating-02	TestTemplating-02.xml

Figure 5: OSSA Batchlet library loaded into OSSA repository

The following sections describe in details each OSSA batchlet.

3.3.2 Common properties

Before listing all the existing OSSA batchlets, here are some description of properties that can be used in any OSSA batchlets.

3.3.2.1 ignoreExceptions

When *ignoreException* property is set to true, in case of exception during the batchlet processing, the processing is not stopped, exit status is not set to ERROR, but only a warning message is pushed in the logs.

Example:

```
<property name="ignoreExceptions" value="true" />
```

3.3.3 SQL Batchlet

3.3.3.1 Overview

The *ossa.Sql* batchlet helps integrators to execute transactional DB operations. In a job execution step a set of SQL statements can be executed using a given datasource.

- The ossa.Sql batchlet provides 99 optional properties named from SQL_01 to SQL_99.
  SQL_XX usage: a simple SQL processing (any statement types are allowed as soon as the database accepts it).
  These statements are executed in sequence from 01 to 99 (no continuous numbering is required).
- All statements executed in a SQL batchlet step are committed at the end of the step.
- A special property named *SQL_RETURN* allows to define the step Exit Status thanks to a SQL query. This can be useful to drive the job execution flow.
  Note that only the first object of the first column of the result set is used as the step ExitStatus.
- 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.
- Sql batchlet allows user to mix SQL statement execution and data import in the same transaction. The CopyToVertica facility can be called in one of the SQL_XX properties.

Within the SQL statements, the *ossa.Sql* batchlet provides additional services:

- `EXECUTE <sql to be executed>`
  The SQL statement is supposed to produce a set of strings that will be considered as new SQL statements to be executed in their turn.
- `VCOPY <property-key>`
  If the statement starts with `VCOPY`, it will be treated as a *"ossa.CopyToVertica"* step. You simply have to provide the property key that defines the copy settings. (see the *ossa.CopyToVertica* batchlet description)
- `DATA(key) <sql statement>`
  `DATA` allows one to store the query result in the job transient data context. Then, the data is accessible thanks to the placeholder `${data.<step-id>.<key>}` (as already seen, data stored in transient context is accessible from all job steps).
  Note that if the query returns one row and one column, the result set object is stored directly, whereas, if the query returns one or multiple rows, it stores a list of key-value `<columnName, value>`
- `STORE(key) <sql statement>`
  `STORE` allows one to store the query result in the job persistent data context. The result set value is attached to the step. It is stored in the STEP_EXECUTION batch tables. It also stores the query result as a job context data which is accessible thanks to the placeholder `${data.<step-id>.<key>}`

3.3.3.2 ossa.Sql batchlet interface

Table 6: ossa.Sql batchlet interface

Features	
Batchlet ref	**`ossa.Sql`**
Logger	`com.hp.ossa.batch.batchlet.jdbc.SqlBatchlet`
Cancellable	Yes
Templating	Yes

Properties	Description	Required or optional	Type
`datasource`	JNDI url of datasource	Required	String
`SQL_01`	SQL statement	Optional	String
`SQL_02`	"	Optional	String
…	"	…	…
`SQL_99`	"	Optional	String
`SQL_RETURN`	SQL executed for pushing a step exit status (will be used by `next` command in job xml)	Optional	String

Exit status	
`COMPLETED`	If no errors during execution of the batchlet and no `SQL_RETURN` defined
`<value>`	If `SQL_RETURN` is defined, the exit status value will be the value of the first column data of the first row

Data pushed into job transient or persistent context	Description	Type
In case of `DATA(key)` or `STORE(key)` usage	user specific data	`String or List[Map<Col,Value>]`

3.3.3.3 Example

This example is an extract of */opt/ossa/repo-ossa/test-batchlet/Sql/TestSql-01.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="TestBatchlet/TestSql-01" xmlns="http://xmlns.jcp.org/xml/ns/javaee"    version="1.0">
        <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>
        <step id="assert" > …
```

3.3.4 SqlCDC Batchlet

3.3.4.1 Overview

The *ossa.SqlCDC* batchlet is a SQL derived batchlet which provides the ***Change Data Capture*** feature.

Change Data Capture design allows to determine (and track) the data that has changed on source table, so that action can be taken using the new data only.
The principle is that, each time the batchlet is executed, the new or updated data from the source table are automatically detected, and the SQL processing occurs only on that modified data.

Like the SQL batchlet, this SqlCDC batchlet allows you to execute several SQL statements in sequence.

The other features of *ossa.Sql* batchlet (refer to *ossa.Sql* batchlet section) are also available in this *ossa.SqlCDC* batchlet.

3.3.4.2 ossa.SqlCDC batchlet interface

Table 7: ossa.SqlCDC batchlet interface

Features	
Batchlet ref	**`ossa.SqlCDC`**
Logger	`com.hp.ossa.batch.batchlet.jdbc.SqlCDCBatchlet`
Cancellable	Yes
Templating	Yes

Properties	Description	Required or optional	Type
`datasource`	JNDI url of datasource	Required	String
`SQL_01`	SQL statement	Optional	String
`SQL_02`	"	Optional	String
…	"	…	…
`SQL_99`	"	Optional	String
`SQL_RETURN`	SQL executed for pushing a step exit status (will be used by `next` command in job xml)	Optional	String
`src_table`	the name of the source table on which the CDC will be applied	Required	String
`src_timeColumn`	This property must be defined only if the limitation of source data is needed (if the `src_maxHoursHandled` is set). In that case, the value of `src_timeColumn` must be the name of the column of the source table which determine the timestamp of the source values.	Optional	String
`src_maxHoursHandled`	This property must be defined only if you want to limit the amount of data to be taken from the source table (in case of large history of data from the source table, and in case you are not interested by all the data, this will allow you to retrieve only the most recent data). Value of this property corresponds to the maximum number of hours that will be handled from the source data. If 0, no limit on the number of hours to be handled. If X different than 0, the processing will not take data older than X hours in the past (in term of data `src_time_column`).	Optional	Integer

Exit status	
`COMPLETED`	If no errors during execution of the batchlet and no `SQL_RETURN` defined
`<value>`	If `SQL_RETURN` is defined, the exit status value will be the value of the first column data of the first row

Data pushed into job transient or persistent context	Description	Type
In case of `DATA(key)` or `STORE(key)` usage	user specific data	`String or List[Map<Col,Value>]`

Note that, in case one of the SQL statement fails, the whole batchlet execution is considered as failed and obviously the CDC tracker will not progress; meaning that the next time the batchlet will be run, the same data will be automatically retrieved (plus potentially new data) in order to be processed by the batchlet (so that you will not miss any data from the source table).

Using this batchlet, you will automatically benefit of the variable CDCFILTER: which gives you the filtering of new rows from the source tables.
When your SQL request requires CDC, you just have to put:

```
… FROM … WHERE ${data['CDCFILTER']}
```

3.3.4.3 Example

SqlCDC batchlet usage:

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="SqlCDCJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
  <properties>
     <property name="datasource" value="java:jboss/datasources/OssaDS" />
  </properties>
  <step id="testSqlCDC">
    <batchlet ref="ossa.CDCSql">
      <properties>
         <property name="src_table" value="SRCTABLE" />
         <property name="SQL_01" value="INSERT INTO DESTTABLE(DATATIME, VALUE) SELECT SRC_DATATIME, VALUE FROM
SRCTABLE WHERE ${data['CDCFILTER']}"/>
      </properties>
    </batchlet>
  </step>
</job>
```

3.3.5 Transformation batchlet

3.3.5.1 Overview

The aim of the *transformBatchlet* is to transform data from a source table to a destination table with specific transformation functions.

Each time the '*transform*' is executed, the new or updated data from the source table are automatically detected, and the transformation processing occurs in order to accordingly insert (no update !) new data into the destination table.
(the CDC principle described on previous section is applied for this batchlet also)

3.3.5.2 transformBatchlet interface

Table 8: transformBatchlet interface

Features	
Batchlet ref	**`transformBatchlet`**

Logger	com.hp.ossa.batch.batchlet.transform.transformBatchlet
Cancellable	Yes
Templating	No

Properties	Description	Required or optional	Type
datasource	JNDI url of datasource	Required	String
transformationName	the name you want to give to your transformation	Required	String
src_table	the name of the source table	Required	String
dest_table	the name of the destination table where the transformed data will be put	Required	String
dest_transformations	Definition of the data transformations for each column of the destination table. It is defined as a list of mappings (separated by '//'): <column name in the dest summarized table> = < function to apply on the source column (as a SQL expression)>	Required	String
src_timeColumn	This property must be defined only if the limitation of source data is needed (if the src_maxHoursHandled is set). In that case, the value of src_timeColumn must be the name of the column of the source table which determine the timestamp of the source values.	Optional	String
src_maxHoursHandled	This property must be defined only if you want to limit the amount of data to be taken from the source table (in case of large history of data from the source table, and in case you are not interested by all the data, this will allow you to retrieve only the most recent data). Value of this property corresponds to the maximum number of hours that will be handled from the source data. If 0, no limit on the number of hours to be handled. If X different than 0, the processing will not take data older than X hours in the past (in term of data src_time_column).	Optional	Integer
verticaTmMaxAcceptedDurMins	(advanced) This parameter is part of the transformation "limiter" feature: it is related to the Vertica activity check done before doing transformation. Its goal is to avoid flooding Vertica with too much new data. This consists in checking if Vertica Tuple Mover operation is running on the destination table, and since how much time. This parameter defines the maximum number of minutes that we accept the Vertica Tuple Mover to run (default value is 5 minutes). If Vertica Tuple Mover is running since less than verticaTmMaxAcceptedDurMins, the transformation is executed. Else, it means that Vertica is still too busy and thus, the transformation	Optional	Integer

	is shifted by few seconds until tuple mover operations are completed: the check is then performed again before the transformation.		

Exit status	
COMPLETED	If no errors during execution of the batchlet
FAILED	If transformation failed

Data pushed into job transient or persistent context	Description	Type
None		

3.3.5.3 Example

Transformation batchlet usage:

Imagine that you have a source table STG_SC_METRICS with the following columns:

```
SRC_DATATIME            Timestamp
SRC_INTEGER             integer
SRC_VARCHAR             VARCHAR
SRC_STR_TIMESTAMP       VARCHAR
```

and a destination table FCT_SC_METRICS:

```
DEST_DATATIME           Timestamp
DEST_INTEGER            integer
DEST_VARCHAR            VARCHAR
DEST_TIMESTAMP          TIMESTAMP
```

The batchlet which transforms your source columns to destination columns will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="transformJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
  <properties>
    <property name="datasource" value="java:jboss/datasources/OssaDS" />
  </properties>


  <step id="myTransform">
    <batchlet ref="transformBatchlet">
      <properties>
        <property name="transformationName" value="MY_TRANSFORM" />
        <property name="src_table" value="STG_SC_METRICS" />
        <property name="dest_table" value="FCT_SC_METRICS" />
        <property name="dest_transformations" value="DEST_DATATIME=SRC_DATATIME // DEST_INTEGER=SRC_INTEGER //
DEST_VARCHAR = SRC_VARCHAR // DEST_TIMESTAMP=TO_TIMESTAMP(SRC_STR_TIMESTAMP, 'YYYY-MM-DD HH24:MI:SS')"/>
        <property name="src_timeColumn" value="SRC_DATATIME" />
        <property name="src_maxHoursHandled" value="744" /> <!-- 1 month -->
      </properties>
    </batchlet>
  </step>
</job>
```

3.3.6 CopyToVertica Batchlet

3.3.6.1 Overview

The *ossa.CopyToVertica* batchlet is a simple encapsulation of the Vertica COPY SQL statement. This SQL function provided by Vertica allows user to **load data files in the database**. For more details about COPY Vertica statement, please refer to the official Vertica SQL Reference documentation (https://my.vertica.com/docs/7.2.x/HTML/index.htm).

- The *ossa.CopyToVertica* batchlet proposes, on top of the standard SQL utility, functions to manage input files.
- The integrator can specify the archiving policy for data file import. OSSAF supports:
  - REMOVE policy: loaded files are simply removed
  - ARCHIVE policy: loaded files are archived to a defined folder
  - NO policy: nothing is done. The input files stay in place.
- The *ossa.CopyToVertica* batchlet supports cancel action. If the Stop method is called on the job running a *ossa.CopyToVertica* batchlet, the current statement is cancelled.
- The integrator is responsible to provide the Copy SQL statement.
- *:input* placeholder is used for pointing to the input file.

3.3.6.2 ossa.CopyToVertica batchlet interface

Table 9: ossa.CopyToVertica batchlet interface

Features	
Batchlet ref	**`ossa.CopyToVertica`**
Logger	`com.hp.ossa.batch.batchlet.jdbc.CopyToVerticaBatchlet`
Cancellable	Yes
Templating	Yes

Properties	Description	Required or optional	Type
`datasource`	JNDI url of datasource	Required	String
`asBaseDir`	Base directory (of the OSSAF server) from which files are looked for	Required	String
`importStatement`	The Vertica copy SQL statement In this statement, the '*:input*' placeholder will be replaced by the filtered input files names	Required	String
`inputFilter`	A regular expression used for matching the input file names from the input directory	Required	RegExp
`inputArchivingPolicy`	REMOVE: delete the imported file ARCHIVE: when loaded, the imported file will be moved to the archive directory NO: nothing to do, imported file is kept	Required	String
`inputDir`	Directory (relative to base directory) where the input files are looked for	Required	String
`errorDir`	Directory (relative to base directory) where the files are moved in case of error	Required	String
`archiveDir`	Directory (relative to base directory) where the files are moved when loaded (if ARCHIVE policy is set)	Required	String
`rejectDir`	Directory (relative to base directory) where the files are moved when there are rejected data	Required	String

failOnError	(default value=true) defines if the batchlet must stop on error in case of SQL exception.	Optional	Boolean

Exit status	
COMPLETED	If no errors during execution of the batchlet
FAILED	If one of the SQL statement has failed

Data pushed into job transient or persistent context	Description	Type
None		

3.3.6.3 Example

This example is an extract of: */opt/ossa/repo-ossa/test-batchlet/Sql/TestCopyToVertica.xml*

```
    <batchlet ref="ossa.CopyToVertica">
      <properties>
        <property name="inputFilter" value="^(data.csv)$" />
        <property name="importStatement" value="
COPY ${data['random-table-name']}
(
        FIELD1,
        FIELD2
)
FROM ${jobProps['import_mode']} '${data['tmp-basedir']}/${jobProps['inputDir']}/:input' ${jobProps['import_node']} DELIMITER ',' TRAILING NULLCOLS SKIP 1 DIRECT
        abort on error
        REJECTED DATA '${data['tmp-basedir']}/${jobProps['rejectDir']}/:input'
        EXCEPTIONS    '${data['tmp-basedir']}/${jobProps['errorDir']}/:input'
        NO COMMIT
        "/>
      </properties>
    </batchlet>
```

Figure 6: ossa.CopyToVertica batchlet example

3.3.7 Summarization batchlet

3.3.7.1 Summarization batchlet overview

The OSS Analytics Batch Library provides a data summarization functionality. This transformation is applicable for multidimensional model, designed as a star schema.

The aim of the batchlet is to aggregate data from a datamart 'fact' table.
The transformation produces data in an output table in which values from the original table are aggregated by selected dimensions and selected time granularities (hourly, daily, weekly, monthly) with given aggregation functions. This is why the transformation is called summarization.

Each time the summarization is executed, the new or updated data from the source table are detected, and the aggregation processing occurs in order to update accordingly the summarized table.

3.3.7.2 summBatchlet interface

Table 10: summBatchlet interface

Features	
Batchlet ref	**summBatchlet**
Logger	com.hp.ossa.batch.batchlet.summ.summBatchlet
Cancellable	Yes
Templating	No

Properties	Description	Required or optional	Type
datasource	JNDI url of datasource	Required	String
summarizationName	the name you want to give to your transformation	Required	String
src_table	the name of the source fact table	Required	String
dest_table	the name of the destination table where the aggregated values will be put	Required	String
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)	Required	String
dest_time_slice	time slice on which the aggregation must be performed. It can be: 'XMIN': for bunch of minutes aggregation. (X can 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 ('DAY') or Monday ('IW') 'MM': for **monthly** aggregation	Required	String
dest_timeColumn	the timestamp column within the destination summarized table which will be filled with the timestamp of the time slice (the timestamp of the start of the timeslice)	Required	String
src_dimensions	the comma separated list of columns of the original fact table defining the dimensions upon which the aggregation must be performed	Required	String
dest_sumAggregations	Definition of the aggregation that must be performed. defined as a list of mappings (separated by '//'): <column name in the dest summarized table> = <the aggregation function done on the source columns (as a SQL expression)>	Required	String
src_maxCalculationPeriod	maximum number of periods of source 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, summarization will not take data older than X <time slice> periods in the past (in term of data time) into consideration for the summarization	Required	Integer
verticaTmMaxAcceptedDurMins	(advanced) This parameter is part of the summarization "limiter"	Optional	Integer

	feature: it is related to the Vertica activity check done before doing summarization. Its goal is to avoid flooding Vertica with too much new data. This consists in checking if Vertica Tuple Mover operation is running on the destination table, and since how much time. This parameter defines the maximum number of minutes that we accept the Vertica Tuple Mover to run (default value is 5 minutes). If Vertica Tuple Mover is running since less than `verticaTmMaxAcceptedDurMins`, the summarization is executed. Else, it means that Vertica is still too busy and thus, the summarization is shifted by few seconds until tuple mover operations are completed: the check is then performed again before the summarization.		

Exit status	
`COMPLETED`	If no errors during execution of the batchlet
`FAILED`	If summarization failed

Data pushed into job transient or persistent context	Description	Type
`None`		

3.3.7.3 Example

```
<properties>
  <property name="datasource" value="java:jboss/datasources/OssaFaultDS" />
  <property name="verticaTmMaxAcceptedDurMins" value="2" />
</properties>

    <step id="summMGT_H">
      <batchlet ref="summBatchlet">
        <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) // SIMILAR_ALARM_COUNTER_SUM=SUM(CASE WHEN alarmclass = 1 THEN 1 ELSE 0 END) // ALARM_COUNTER_SUM=COUNT(1) // CRITICAL_OCCURRENCES_SUM=sum(CRITICALOCCURRENCES) // MAJOR_OCCURRENCES_SUM=sum(MAJOROCCURRENCES) // MINOR_OCCURRENCES_SUM=sum(MINOROCCURRENCES) // WARNING_OCCURRENCES_SUM=sum(WARNINGOCCURRENCES) // INDETERMINATE_OCCURRENCES_SUM=sum(INDETERMINATEOCCURRENCES) // CLEAR_OCCURRENCES_SUM=sum(CLEAROCCURRENCES) // CLEAR_COUNTER_SUM=sum(CASE WHEN clearflag =1 THEN 1 ELSE 0 END) // ESCALATED_COUNTER_SUM=sum(CASE WHEN escalatedalarmflag =1 THEN 1 ELSE 0 END) // HANDLED_COUNTER_SUM=sum(CASE WHEN handleflag =1 THEN 1 ELSE 0 END) // CLOSED_COUNTER_SUM=sum(CASE WHEN closeflag =1 THEN 1 ELSE 0 END) // ACK_COUNTER_SUM=sum(CASE WHEN ackflag =1 THEN 1 ELSE 0 END) // TERMINATED_FLAG_SUM=sum(CASE WHEN termflag =1 THEN 1 ELSE 0 END) // ACK_BY_COUNT=count(ackuserid) // CLOSED_BY_COUNT=count(closeuserid) // TERMINATED_BY_COUNT=count(termuserid) // RELEASED_BY_COUNT=count(releaseuserid) // HANDLED_BY_COUNT=count(handleuserid) // CLEAR_DURATION_SUM=sum(CLEARDURATION) // HANDLE_DURATION_SUM=sum(HANDLEDURATION) // CLOSE_DURATION_SUM=sum(CLOSEDURATION) // ACK_DURATION_SUM=sum(ACKDURATION) // TERM_DURATION_SUM=sum(TERMDURATION) // COLLECTION_DURATION_SUM=sum(COLLECTIONDURATION) // IN_FAULT_DURATION_SUM=sum(INFAULTDURATION) // IN_MANAGEMENT_DURATION_SUM=sum(INMGMTDURATION) // COLLECTION_COUNTER_SUM=sum(CASE WHEN COLLECTIONDURATION is not null THEN 1 ELSE 0 END) // IN_FAULT_COUNTER_SUM=sum(CASE WHEN INFAULTDURATION is not null THEN 1 ELSE 0 END) // IN_MANAGEMENT_COUNTER_SUM=sum(CASE WHEN INMGMTDURATION is not null THEN 1 ELSE 0 END) // CLEAR_DURATION_AVG=avg(CLEARDURATION) // HANDLE_DURATION_AVG=avg(HANDLEDURATION) // CLOSE_DURATION_AVG=avg(CLOSEDURATION) // ACK_DURATION_AVG=avg(ACKDURATION) // TERM_DURATION_AVG=avg(TERMDURATION) // COLLECTION_DURATION_AVG=avg(COLLECTIONDURATION) // IN_FAULT_DURATION_AVG=avg(INFAULTDURATION) // IN_MANAGEMENT_DURATION_AVG=avg(INMGMTDURATION)" />
          <property name="src_maxCalculationPeriod" value="2232" /> <!-- = 93 days = 3 months-->
        </properties>
      </batchlet>
    </step>
```

Figure 7: summBatchlet example

3.3.8 ConsoleReport Batchlet

3.3.8.1 Overview

When one uses OSSA Server and HPE Unified OSS Console, it is possible to associate to a job some report generation (as e.g. PDF document); then subsequent job step can use the generated report and send it as email attachment. In order to generate a report, integrators need to define:

1. Authentication parameters

Because the OSS Console is a secured application, the *ConsoleReport* batchlet supports 2 types of authentication:

- either you use a specific user and password, which authorization allows to generate the requested report (this option is only working if OSS Console has been configured to use internal authentication provider)
- either you can use specific Auth2 token value. This token should be a valid token recognized and accepted by the OSS Console application to generate a report.
  For more details about how to generate and get a valid OSS Console Token, please refer to the *HPE UOC Installation Guide* in chapter Security Guide, section Authentication, paragraph *Generate JSON Web Token*.

2. Report specification

The 2 main parameters to generate an OSS Console report are:

- The report data Uri : this is the same uri than the one which is displayed in the browser when the UOC user navigate to the requested report
- The *ossConsoleReportUri*: this is the OSS Console report generator service URI.

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

3.3.8.2 ossa.ConsoleReport batchlet interface

Table 11: ossa.ConsoleReport batchlet interface

Features	
Batchlet ref	**`ossa.ConsoleReport`**
Logger	`com.hp.ossa.batch.batchlet.resource.ConsoleReportBatchlet`
Cancellable	No
Templating	Yes

Properties	Description	Required or optional	Type
`baseFolder`	Output base folder where downloaded report are stored	Required	String
`url`	URL of the resource to be downloaded	Required	String
`File`	Name of the file where the resource will be stored	Required	String
`ossConsoleUrl`	URL of OSS Console application	Required	String
`ossUsername`	OSS Console login	Optional	String
`ossPassword`	OSS Console password	Optional	String
`ossToken`	Security token (in case no user/password is used)	Optional	String

ossConsoleLoginUri	Login uri	Required	String
ossConsoleReportUri	OSS Console Report generator service uri	Required	String
Uri	Report uri	Required	String
orientation	portrait or landscape	Required	Enum
format	A4, A3...	Required	Enum
margin	Margin in the report generated. Example: 10	Required	Integer
viewportHeight	Example: 1200	Required	Integer
viewportWitdh	Example: 1900	Required	Integer

Exit status	
COMPLETED	If no errors during execution of the batchlet
FAILED	If generation or download of the report fails

Data pushed into job transient or persistent context	Description	Type
res	The downloaded file	Java File object (*Java.io.File*)
link	Part of URI link for retrieving the file that was generated by the batchlet. Example: /ossa/resource?path=/tmp/job-XXX/mydata.txt One must prefix with Error! Hyperlink reference not valid."http://<OSSAFserver>:<port>/" for having the full uri for accessing the file	String
source	OSS Console report http link that was used to get the report	String

3.3.8.3 Example

```
<!-- 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="  /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>
```

Figure 8: ossa.ConsoleReport batchlet example with user/password

```
<!-- 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>
```

Figure 9: ossa.ConsoleReport batchlet example with token

3.3.9 Mail batchlet

3.3.9.1 ossa.Mail batchlet overview

The *ossa.mail* batchlet allows to **send an Html templated mail** in their integration flows.

- Mail recipients can be defined thanks to the *'to'*, *'cc'*, *'bcc'* properties. You can add several recipients by using the comma separator.
- The property *'from'* allows to define who is sending the mail (obviously, it should be accepted by your underlying SMTP server)
- The SMTP server is defined during OSS Analytics Foundation configuration.
  (please refer to parameters OSSA_MAIL_SERVER and OSSA_MAIL_PORT in *HPE OSS Analytics Foundation - Installation Configuration and Administration Guide*)
- The content of the mail should be HTML content.
- As the properties of this batchlet can be templated, you can externalize in a file the content and use FreeMarker functions in order to generate dynamic content.
- The *"attachments"* property allows to manage the attachments of the mail. Its value must be filled with the list of previous job "step id" values supposed to provide a 'Resource' to attach. For instance, an *ossa.ConsoleReport* step can be defined before the *ossa.Mail* step: this downloaded report from the *ossa.ConsoleReport* step can then be added as attachment to the mail to send.

3.3.9.2 ossa.Mail batchlet interface

Table 12: ossa.Mail batchlet interface

Features	
Batchlet ref	**ossa.Mail**
Logger	com.hp.ossa.batch.batchlet.resource.MailerBatchlet
Cancellable	No
Templating	Yes

Properties	Description	Required or optional	Type
from	Mail sender address	Required	String
To	Comma separated list of mail recipients	Required	String
Cc	Comma separated list of mail cc recipients	Required	String
bcc	Comma separated list of mail bcc recipients	Required	String
attachments	List of attachments. You must use the job step id of a *resource* step in order to attach this resource to the mail.	Optional	Job step ids
subject	Mail subject	Required	String
content	Mail content	Required	String

Exit status	
COMPLETED	If no errors during execution of the batchlet
FAILED	If batchlet failed

Data pushed into job transient or persistent context	Description	Type
None		

3.3.9.3 Example

In this example the mail content is produced by processing the template where the user can use environment data to produce the html mail content. This example is an extract of: */opt/ossa/repo-ossa/test-batchlet/Mail/TestMaillerAttachments.xml*

Mail step (using external template file):

```
<step id="mail" >
  <batchlet ref="ossa.Mail">
    <properties>
      <property name="from"        value="ossa.batch@hpe.com"/>
      <property name="to"          value="user@hpe.com"/>
      <property name="attachments" value="download"/>
      <property name="subject"     value="Reporting demo"/>
      <property name="content"     value="[[TestMailer2.ftl]]"/>
    </properties>
  </batchlet>
</step>
```

Templated html mail content (*TestMailer2.ftl*) with dynamic data:

```
<br>
Hello, <br>
<br>
OSSAF job ${job.jobName} has sent this mail with an attachment.
<br>
<br>
Here is a permanent link to see the attachment:
  <a href="http://localhost:8080/${data['download'].link}"?>${data['download'].res.name}</a>

<br>
<br>
Regards
<br>
```

Figure 10: ossa.Mail batchlet example

3.3.10 HTTP batchlet

3.3.10.1 ossa.http batchlet overview

The *ossa.http* batchlet allows integrator to send any kind of HTTP request in their batch jobs.

- The property '*method*' allows to define the HTTP query type: `GET, POST, PATCH, PUT, DELETE, HEAD, OPTIONS, TRACE`.
- The property '*headers*' allows to manage the HTTP header of the request. Its value is a List of Key,Value.
- The property '*url*' defines the request target
- The optional property '*content*' defines the content of the request to send. It is a template.
- 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. '*store*' value will identify the name of the persisted step data.
- The HTTP response code is returned as the *Exit Status* for the step
- The *ossa.http* batchlet is cancelled when stopping the job.

3.3.10.2 ossa.http batchlet interface

Table 13: ossa.http batchlet interface

Features	
Batchlet ref	**`ossa.http`**
Logger	`com.hp.ossa.batch.batchlet.http.HttpBatchlet`
Cancellable	Yes
Templating	Yes

Properties	Description	Required or optional	Type
`method`	Type of request: `GET | POST | PATCH | PUT | DELETE | HEAD | OPTIONS | TRACE`	Required	String
`url`	Request target url	Required	String
`headers`	HTTP request header	Optional	List(<Key>, Value)
`content`	Request content	Optional	String
`Store`	Name of the column where the result of the request will be persisted.	Optional	String

	If not set, the result of the request will be available as transient data in `[step-id].result`		

Exit status	Description
`[HTTP Response Status]`	HTTP response code: 200, 204 ...

Data pushed into job transient or persistent context	Description	Type
`[step-id].result`	Transient data containing response content if exists	String
<value of `store` property>	Persistent data containing response content if exists	

3.3.10.3 Example

In this example, the *ossa.http* batchlet calls an OSSAF RestApi entry point in order to get the server Uuid. This unique identifier is then stored in the DB (because the `store` property is set).
The HTTP response can be used to define the next step.

This example is an extract of: */opt/ossa/repo-ossa/test-batchlet/Http/TestHttp.xml*

```
<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>
```

Figure 11: ossa.http batchlet example

3.3.11 Resource batchlet

3.3.11.1 ossa.Resource batchlet overview

The *ossa.Resource* batchlet is a step abstraction that is supposed to **provide a resource to other steps**. For example, the `attachments` property value of the *ossa.Mail* batchlet must reference an *ossa.Resource* step id. And so, a job can be defined for attaching a downloaded resource (result from *ossa.Resource*) to a mail.

- The *ossa.Resource* batchlet needs first to be define a base folder where resources will be stored.
- The default behavior 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 named by the value of the `file`.

3.3.11.2 ossa.Resource batchlet interface

Table 14: ossa.Resource batchlet interface

Features	

Batchlet ref	**ossa.Resource**
Logger	com.hp.ossa.batch.batchlet.resource.ResourceBatchlet
Cancellable	No
Templating	Yes

Properties	Description	Required or optional	Type
baseFolder	Directory where downloaded resource are stored as files	Required	String
url	Resource url to be downloaded	Required	String
File	Output file name (in the baseFolder)	Required	String

Exit status	
COMPLETED	If no errors during execution of the batchlet
FAILED	If batchlet failed

Data pushed into job transient or persistent context	Description	Type
source	The URL where resource has been downloaded	String
Link	Part of uri link for retrieving the file that was generated by the batchlet. Example: /ossa/resource?path=/tmp/job-XXX/mydata.txt One must prefix with Error! Hyperlink reference not valid.“http://<OSSAFserver>:<port>/” for having the full uri for accessing the file	String
Res	The downloaded file	Java File object (*Java.io.File*)

3.3.11.3 Example

In this example, the *ossa.mail* batchlet is used in order to send as an attachment an *ossa.Resource*.
This example is an extract of: */opt/ossa/repo-ossa/test-batchlet/Mail/TestMaillerAttachments.xml*
As a first step, a resource file (simple one for the example) is downloaded.
In the second step, this downloaded file is sent as an attachment to a mail.

```
<properties>
  <property name="baseFolder" value="${props['java.io.tmpdir']}"/>
</properties>

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

<step id="mail" >
  <batchlet ref="ossa.Mail">
    <properties>
      <property name="from"        value="ossa.batch@hpe.com"/>
      <property name="to"          value="test@hpe.com"/>
      <property name="attachments" value="download"/>
      <property name="subject"     value="Reporting demo"/>
      <property name="content"     value="[[TestMailer2.ftl]]"/>
    </properties>
  </batchlet>
</step>
```

Figure 12: ossa.Resource batchlet example

3.3.12 Javascript batchlet

3.3.12.1 ossa.javascript batchlet overview

The *ossa.javascript* batchlet allows integrator to implement any logic within a job step thanks to the javascript language. Moreover, note that the OSSA Server java script engine allows also to call any Java method from the standard Java API.

The main java scripting use cases could be:

- initialization step: before starting a processing, you can setup variables, create files, load configuration from properties ….
- assertion step: in order to verify the result of some previous step execution and drive the rest of the job flow
- processing step: call user defined functions

The *ossa.javascript* batchlet has only one property named '*script*'. This property can use templating.
Using this templating, one benefits from native contextual objects available in templates.

Table 15: contextual objects available in templates

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 OSSA Server batch system	com.hp.ossa.batch.BatchIfBase

3.3.12.2 ossa.javascript batchlet interface

Table 16: ossa.javascript batchlet interface

Features	
Batchlet ref	**`ossa.javascript`**
Logger	`com.hp.ossa.batch.batchlet.script.JavascriptBatchlet`
Cancellable	No
Templating	Yes

Properties	Description	Required or optional	Type
`script`	Java script	Required	String

Exit status	
`[Any user defined value]`	Value returned by the script

Data pushed into job transient or persistent context	Description	Type
`[Any user defined data]`		

3.3.12.3 Example

In this first example, extracted from */opt/ossa/repo-ossa/test-batchlet/Sql/TestCopyToVertica.xml*
one uses ossa.javascript batchlet for generating folders and csv data that will be then imported, in later steps, thanks to the CopyToVertica batchlet. You can see that you can access the full java standard API within the javascript.

```
<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, extracted from */opt/ossa/repo-ossa/test-batchlet/TestRunBatches.xml* (which is a test suite executing several test jobs) one uses the javascript batchlet as an assertion step in order to verify that all steps previously executed completed with an 'OK' status.
Note also the example of a usage the OSSA batch system thanks to the object *'batch'* natively available 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>
```

Figure 13: ossa.javascript batchlet example

3.3.13 Run batch batchlet

3.3.13.1 ossa.batch batchlet overview

The *ossa.batch* batchlet allow integrators to run any new batch instance within a batch.

- The ExitStatus of the batch executed is returned as the current step status.
- The batch is executed synchronously.
- The *ossa.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

3.3.13.2 ossa.batch batchlet interface

Table 17: ossa.batch batchlet interface

Features	
Batchlet ref	**`ossa.batch`**
Logger	`com.hp.ossa.batch.batchlet.utils.RunBatchBatchlet`
Cancellable	No
Templating	Yes

Properties	Description	Required or optional	Type
`Batch`	Name of the batch that will be executed	Required	String
`Params`	Batch input parameters	Optional	Map<String,String>
`Timeout`	Batch timeout in ms. (-1 means no timeout)	Optional	long

Exit status	
`[BatchExitStatus]`	Value returned by the batch executed

Data pushed into job transient or persistent context	Description	Type
`None`		

3.3.13.3 Example

This example is extracted from */opt/ossa/repo-ossa/test-batchlet/TestRunBatches.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>
```

Figure 14: ossa.javascript batchlet example

3.3.14 Run System process batchlet

3.3.14.1 ossa.Run batchlet overview

The *ossa.Run* batchlet allows integrator to run a system command 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.

3.3.14.2 ossa.Run batchlet interface

Table 18: ossa.Run batchlet interface

Features	
Batchlet ref	**`ossa.Run`**
Logger	`com.hp.ossa.batch.batchlet.system.ProcessBatchlet`
Cancellable	Yes
Templating	Yes

Properties	Description	Required or optional	Type
`command`	Command to be executed on the system	Required	String
`arguments`	Comma separated list of command arguments	Optional	String
`environment`	Comma separated list of environment variables for the process execution	Optional	String
`baseDir`	Base directory for the process execution	Optional	String

Exit status	
`[ProcessExitStatus]`	Value returned by the system process

Data pushed into job transient or persistent context	Description	Type
`processOut`	The process output stream	*java.io.BufferedReader*
`processErr`	The process error stream	*java.io.BufferedReader*

3.3.14.3 Example

This example is extracted from */opt/ossa/repo-ossa/test-batchlet/ExecSystemCommand/TestExecSystemCommand.xml*

```
<step id="exec">
  <batchlet ref="ossa.Run">
    <properties>
       <property name="command"  value="echo 1 > /tmp/TestExecSystemCommand.txt"/>
    </properties>
  </batchlet>
  <next on="0" to="assert"/>
  <fail on="*" exit-status="KO"/>
</step>
```

Figure 15: ossa.Run batchlet example