JobStarter

Contents

[Description 4](#_Toc534893193)

[Modules 4](#_Toc534893194)

[JobStarter 4](#_Toc534893195)

[Description 4](#_Toc534893196)

[Settings 4](#_Toc534893197)

[Segments 5](#_Toc534893198)

[<app-config> 5](#_Toc534893199)

[<bulk-size> 5](#_Toc534893200)

[<connection-timeout> 6](#_Toc534893201)

[<directories> 6](#_Toc534893202)

[<mail> 7](#_Toc534893203)

[<db-types> 7](#_Toc534893204)

[<connection-string> 7](#_Toc534893205)

[<ldap-server> 8](#_Toc534893206)

[Commands 9](#_Toc534893207)

[Parameters 13](#_Toc534893208)

[CallExtractSql 17](#_Toc534893209)

[Description 17](#_Toc534893210)

[Parameters 17](#_Toc534893211)

[Complete command Example 20](#_Toc534893212)

[CallExtractTable 20](#_Toc534893213)

[Description 20](#_Toc534893214)

[Parameters 20](#_Toc534893215)

[Complete command Example 23](#_Toc534893216)

[CallSqlLoader 23](#_Toc534893217)

[Logic 23](#_Toc534893218)

[CallStoredProcedure 24](#_Toc534893219)

[Parameters 24](#_Toc534893220)

[Complete command Example 26](#_Toc534893221)

# Description

JobStarter is a command window based ETL tool created for extracting information from different sources based on:

1. Queries (SQL) contained in a file.
2. Metadata. The client used for the connection must have read from data dictionary privileges.

This tool support multiples databases connection, such as: Oracle, DB2, Teradata, SQL Server, MS Access. In the future, more databases support can be added, using the interface “IDBConnection”.

One benefit offered by this tool is the ability of encrypt the password used by the Database connection, adding another level of security. Connections and password modifications can be done by the respective interface passing the necessaries parameters, we will cover this in further session.

This applications use xml configuration files adding a high level of customization. If you need to pass a new parameter to a module, just defined it in the “Commands.xml” file (covered later on this document).

# Modules

## JobStarter

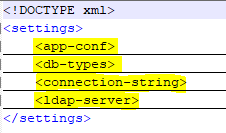
### Description

This is the Main Module containing the executable logic. Parameters are passed to this module and desired logic will be executed.

### Settings

The setting file for this module can be found under “**Settings**” folder (Application Path -> settings -> settings.xml). The settings file is in XML format and contain 4 sections or segments:

Figure 1



#### Segments

##### <app-config>

Figure 2



###### <bulk-size>

Default value 50,000 (records)

This parameter defined the number of records pulled from the database on the RecordSet at a time. The value is a combination of network speed, database type, volume of information extracting. We found that 50K is the best number, and we get the best runtime/performance.

###### <connection-timeout>

Default value 900,000 (milliseconds)

This parameter is useful when you are running a query for long period of time. Any extracting operation should finalize before this timeout, otherwise you will get a connection timeout and triggering an error reporting process.

###### <directories>

Here is defined the directories used by the application. [app\_directory] is a reserved word and has to be in-between “[]”, this indicated to the app to include in the path the working directory where the JobStarter.jar app is running:

1. Outpu-dir

Where plain files generated from the source extraction will be placed.

1. Log-dir

Each run generate a LOG file, here you can define the path for these files

1. Archive-dir

After loading a file into the database, you have the option to specify if archiving those files is desired. Here you can define the path for the archiving.

1. Input-dir

The input folder is used by the LOAD process for loading plain files into the database.

1. Work-dir

Once the LOAD process identify a file or files to be loaded, these are copied into the WORK directory (path provided here) + and folder named after the Process ID.

1. Lib-dir

Where all modules created for this application are copied. All JARs file not including database connection drivers, these are copied into the DRIVERS folder.

1. Script-dir

All scripts created for extraction need to be placed in this folder. The application will look for any xxx.sql file matching the parameter used.

1. Ctl-dir

90% of the time CTLs are built on runtime by CallSqlLoader.jar module, using destination metadata. But if needed it, custom CTL file can be created and has to be place in this folder.

###### <mail>

1. Mail-param

All parameter related to the SMTP server

1. Mail-list

List of all receipting mails

##### <db-types>

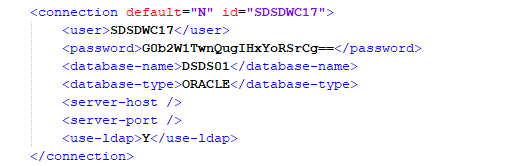
Here is defined all supported databases:

1. Class: name of the class used
2. Driver: the driver published by the fabricant
3. Id: used later on the connection-string configuration

##### <connection-string>

Here is configured necessary parameters to connect to a desired database. Any modification or new connection has to be done through the interface calling the respective parameter.

Figure 3



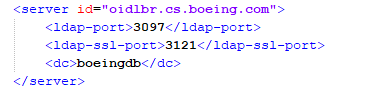
1. ID: the unique identifier for the connection. Used by jobs
2. Default (Y/N): this property indicate if the connection is default and used if no connection id is specified in a job call.
3. User: database user used by the connection
4. Password: encrypted password used by the connection
5. Database-name: The name of the database
6. Database-type: Id specified and created in previous section (<db-types>)
7. Use-ldap (Y/N): this parameter is valid for Oracle database connection. In case of Boeing, an LDAP server is used for user authentication.
8. Server-host: if no LDAP server authentication is used, the you might have to indicate the server address
9. Server-port: same as prior point, is no LDAP server is used, server-host and port must be specified.

##### <ldap-server>

Boeing use LDAP server for database user authentication. This information was obtained from TOAD connection tool. For any other information related contact:

Moeller (US), Kenneth M [Kenneth.M.Moeller@boeing.com](mailto:Kenneth.M.Moeller@boeing.com)

Figure 4



1. Id: LDAP host server name
2. Ldap-port: none ssl port
3. Ldap-ssl-port: SSL port
4. Dc: the domain controlled

### Commands

| **ID** | **Parent** | **Description** | **Name\*** | **Type\*** | **Class\*** | **Value\*** | **Jar\*** | **Required\*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -r |  | Parameter indicating the RUN action | OBJECT | Call | CallerImpl | next | N/A | N/A |
| r | -r | Run stream job configured in the database. | OBJECT | LIB | Not Implemented |  |  |  |
| s | -r | Run Stored Procedure using a Jar module in LIB directory  E.g.  -rs <procedureName1,procedureName2,…> --c connectionName JOBNAME=’Name given to the job’ | OBJECT | LIB | com.boeing.jobstarter.services.CallStoredProcedure | next | CallStoredProcedure.jar | N/A |
| f | -r | Run anonymous block contained in a file. | OBJECT | LIB | Not Implemented |  |  |  |
| l | -r | Run load module (SQL Loader) for loading files into oracle database | OBJECT | LIB | com.boeing.jobstarter.services.CallSqlLoader | none | CallSqlLoader.jar | JOBNAME,  CONNECTION |
| x | -r | Extract |  |  |  |  |  |  |
| t | x | Extract and generate plain file, from sources based on table name and metadata on the source.  e.g.  -rxt <tablename1,tableName2,…> --c connectionName --sd schemaDestName –JOBNAME=’Name given to the job’ | OBJECT | LIB | com.boeing.jobstarter.services.CallExtractTable | next | CallExtractTable.jar | JOBNAME,  CONNECTION, SCHEMADEST |
| w | x | Extract from Web Service | OBJECT | LIB | Not Implemented |  |  |  |
| q | x | Extract and generate plain file, based on a file containing a SQL command  e.g.  -rxt SQLFIleName.sql --c connectionName --sd schemaDestName --JOBNAME=’Name given to the job’ | OBJECT | LIB | com.boeing.jobstarter.services.CallExtractSql | next | CallExtractSql.jar | JOBNAME,  CONNECTION, SCHEMADEST |
| c | -r | Connection Configuration module |  |  |  |  |  |  |
| l | c | List all connections | LIST | call | CallConnectionConfig | none | N/A | N/A |
| a | c | Add new connection | ADD | Call | CallConnectionConfig | none | N/A | N/A |
| d | c | Delete a connection | DELETE | Call | CallConnectionConfig | none | N/A | N/A |
| m | c | Modify a connection | MODIFY | Call | CallConnectionConfig | none | N/A | N/A |
| t | c | Test a connection | TEST | Call | CallConnectionConfig | none | N/A | N/A |
| d | -r | Create a table on the destination schema base on a table name passed as parameter on the source database  e.g.  -rd tableName1, tableName2… --c connectionName --sd schemaDestName | OBJECT | Call | CallDDL | next | N/A | N/A |

**ID**: Parameter code. Can be one letter or multiple base on desire logic to execute.

E.g. Let say we want to execute the SQLLoader module. We need to send the respective parameters

-r => Run

l => Load

The complete parameter command should looks like

-rl --p <file patter1>=<destination table1>,<file patter2>=<destination table2>,… --c <connection id> --REFRESH=<TRUNCATE|DELETE> (if DELETE was chosen, then DELETEBY parameter must be specify) DELETEBY=”column1=value1 and column2=value2…“

**Name**: Command name

**Type**: specify if the desired action is a call to a core logic on JobStarter module or a call to a custom jar (module) added as enhancement

Call: will call a class or CORE logic in the JobStarter application

E.g. the Connection configuration is part of the Core and the way to invoke this functionality is declaring the action as Call because it, implements the ICall interface.

LIB: Will invoke an external JAR included in the LIB folder. This functionality will generate an instance of the invoked class and execute it. Any LIB class to be call, has to extend from Callable Abstract class.

**Class**: the class name to be invoke. In the case of LIB should be included the entire classPath and for Call type only the class name

**Value**: If there is a value to be read, indicate in which position the value is coming. To any given value, containing spaces, it should be enclosed by “…”

None: no value to read

Next: read from next position parameters.

Read: read the parameter from the same position after the expected “=”

E.g. --PARAM=value or --PARAM=”value with spaces”

**Jar**: for Command type LIB, jar file name must be specified.

**Required**: indicate which parameters are required by the command.

### Parameters

It’s important to keep in mind that java recognize SPACE as list delimiter. When we are calling the app and passing parameters one space means a new parameter, so if we intend to pass a String to the app or any other value with SPACES on it, we have to enclose the value between “”.

E.g. --JOBNAME=This is a job Example.

If this case java will identify as parameters (1)--JOBNAME=This (2)is (3)a (4)job (5)Example.

The correct way to do this is:

--JOBNAME=”This is a job Example.”

| **ID** | **Description** | **Name** | **Value** | **Other Values** |
| --- | --- | --- | --- | --- |
| --c | Primary connection used for the apps. | CONNECTION | Next |  |
| --p | Parameter or list of parameters (split by comma) | PARAMETER | Next |  |
| --sd | Schema destination. Where the extracted files will be loaded. This parameter is fundamental because it is included in the generated file name. Then the load process will look for this name in the file name pattern and load the info in the respective schema. | SCHEMADEST | Next |  |
| --NOHEADER | Indicate in the extract phase, if the file will include the header or not (By default header is generated). In the Load phase will indicate if the file contain a header and needs to be skipped. | NOHEADER | none |  |
| --SKIP | This parameter is used to indicate to the SQLLoader how many lines are desired to be skipped. By default the skip value depend of the NOHEADER parameter, if was indicated as header is included, then the skip value going to be 1, other way will be 0 | SKIP | Read |  |
| --ZIP | Indicate if the file will be zipped after the generation. By default no zip functionality | ZIP | none |  |
| --ARCHIVE | Indicate if after the generation or load, the archive functionality is desired. This process will copy and zip the file/work directory, into the archive folder | ARCHIVE | None |  |
| --NODELETE | This parameter indicate if after the generation, load or any process, the resulting files will be deleted. If this parameter is no indicated, the default behavior is to delete all | NODELETE | None |  |
| --COPY | Indicate if the generated file will be copy over any given path | COPY | Read |  |
| --MOVE | Indicate is the generated file will be moved to different path | MOVE | Read |  |
| --REPORTBY | Indicate how do you want to report Failure/Success | REPORTBY | read | MAIL,COPY,DATABASE |
| --REPORT | Indicate if you want to report any failure, success or both when job ran | REPORT | read | SUCCESS or ERROR or BOTH |
| --JOBNAME | The job name for reporting purpose | JOBNAME | read |  |
| --LOADLAST | This parameter indicate when multiples files from the same table were generated, if the application can only load the latest version. | LOADLAST | None |  |
| --USEHEADER | Indicate if the header contained in the file can be used as column name for the dynamic CTL construction. Instate of using the METADATA or custom CTL file | USEHEADER | None |  |
| --DATEFORMAT | This is used by the load process to indicate how the date is formatted | DATEFORMAT | Read |  |
| --REFRESH | Indicate how the destination table will be refreshed before loading the new data. Possible values are TRUNCATE or DELETE.  If truncate is specified, then the entire table will be erased  If delete is specified, then the table will be delete before load base on the conditions specified in the parameter DELETEBY | REFRESH | Read | TRUNCATE or DELETE |
| --DELETEBY | Provide the conditions under a table will be deleted before loading new values. Condition must be provided, same as any WHERE clause but without the word WHERE.  E.g.  --DELETEBY=”column1=value1 and column2=value2 …” | DELETEBY | Read |  |
| --ADDLOADDATE | In the load process, include the sysdate to specified column  E.g.  --ADDLOADDATE=columDate1 | ADDLOADDATE | Read |  |
| --CTL | I customized CTL file is required for the load process, this must be specified through this parameter. Notice that the CTL file must exists in the CTL folder | CTL | Read |  |

**Value**: Indicate in which position the value is coming for the specified parameter. To any given value, containing spaces, it should be enclosed by “…”

None: no value to read

Next: read from next position from the list of parameters.

Read: read the parameter from the same position after the expected “=”

E.g. --PARAM=value or --PARAM=”value with spaces”

## CallExtractSql

### Description

This module extract from any source information based on input files in “[app\_directory]\scripts\” folder, this file must contain a SQL command.

Once the information is extracted, the module proceed to generate a plain file split by comma “,”, with the extension “.csv”.

Any file generated is placed in the “output” folder (Path obtained from “settings.xml” file and <output-dir> segment.

This module receive as parameter one or a list of SQL file names and proceed to extract the information form the source using the provided connection ID. Information is extracted and generated one SQL file at the time. The resulting plain file will be named using the fallowing pattern:

[Schema Destination]\_[SQL File Name]\_[yyymmddhhmmss].csv

For more actions after file generation, please refer to the session Parameters

### Parameters

**Command call**: [-rxq] this is the command required for invoking the execution of the module

**List of SQL files:** [File\_Name\_1.sql,File\_name\_2.sql…] can be one or many files names. All file named here must exists in the script folder “[app\_directory]\scripts\” or any other location defined on the <script-dir> tag in the settings.xml file.

**--p:** If the SQL command contain a Bind parameter (?)

E.g.

Select column1, column2, column3 From Table1 where Table1.columnX = ?

You may pass the value using --p parameter. And you can have as many as Bind parameters are defined inside the SQL commands and files.

If you want to use the same value in multiples SQL commands/Files then you must to use only the column name.

*Scenario No. 1*

File1.sql

Select column1, column2, column3 From Table1 where Table1.columnX = ?

File2.sql

Select column1, column2, column3 From Table1 where Table2.columnX = ?

If the value for columnX is the same in both SQL commands, you may define the parameter as:

--p columnX=’value’ or columnX=12345

*Scenario No. 2*

File1.sql

Select column1, column2, column3 From Table1 where Table1.columnX = ?

File2.sql

Select column1, column2, column3 From Table1 where Table2.columnX = ?

If the value for columnX is different in both queries, then you have to specify the value desired for each SQL file, using the file name as a prefix.

--p file1.columnX=”value1”,file2.columnX=”value2”

You can include the table name if you want

--p file1.table1.columnX=”value1”,file2.table2.columnX=”value2”

**--c:** Connection ID to be used to connect to the source database. The ID can be obtained from the file “settings.xml” or running the command:

java --jar JobStarter --rcl

**--sd:** Schema destination ID. Specify the Oracle schema where eventually the file will be uploaded

This parameter is important and required. Is used for the file naming convention, and later on used by the load process to identify the file in the input folder.

**--MOVE:** This parameter is only required if after the file generation you want to move/copy it to different location.

This parameter and --COPY are useful when the extraction and load is happening in the same server. Because the load process only read from the IMPUT folder, you might want to copy/move the file to the input folder once it’s generated, in order to be picked-up by the load process.

**--COPY:** This parameter is only required if after the file generation you want to move/copy it to different location. A copy of the original file will remains in the output folder.

**--ZIP:** This parameter is not required and indicate if after the file generation you want to compress the file in ZIP format. By default no compression is happening

**--JOBNAME:** This parameter allows you to identify the job in any report after successful or failure completion of job execution. Will be included as part of the subject on email report or any other way of reporting desired.

**--REPORT:** In this parameter you can indicated when you want to be notified. If the process FAIL, SUCCESS or BOTH. You might choose FAIL or SUCCESS or BOTH

**--REPORTBY:** How do you want to be notified?

* MAIL: based on the value passed on parameter --REPORT you will get an email indicating the status or error details. All mail will be sent to the distribution list configured in settings.xml file
* COPY: if this value is specified, then a copy of the LOG file will be copied in the output folder

This functionality was introduced for the GIO81 port forwarding, and because the server is not connected to any internal Boeing resource like SMTP server. This functionality will allow the app to reports a failure to the load process. The load process will identify the error file in the input folder and report the error by email.

* DATABASE: This function in not implemented, but it will allow to save in any oracle table the error for further revision

**--ARCHIVE:** with this parameter you indicate if after the process ends, you want to archive any resulting file, like LOGS generated files, CTLs, etc. Files will be zipped and stored in a folder under the connection name in the <archive-dir> path, configured in the settings.xml file

### Complete command Example

java -Xmx1024M -jar JobStarter.jar -rxq etvs\_bca\_active.sql --c NWCS271A --sd DW737IC --MOVE="C:\Development\workspaces\default-configuration\JobStarter\target\classes\input" --ZIP --JOBNAME="Extract from MS SQL Server Database NWPACTIVE" --REPORT=FAIL --REPORTBY=MAIL

## CallExtractTable

### Description

This module extract from any source information based on tables/views metadata. Dynamically the module will create a SELECT statement base on the table/view metadata and extract the information.

This module receive as parameter one or a list of Tables/Views names and proceed to extract the information form the source using the provided connection ID. Information is extracted and generated one Table/View at the time.

Once the information is extracted, the module proceed to generate a plain file split by comma “,”, with the extension “.csv”.

Any file generated is placed in the “output” folder (Path obtained from “settings.xml” file and <output-dir> segment.

The resulting plain file will be named using the fallowing pattern:

[Schema Destination]\_[Table Name]\_[yyymmddhhmmss].csv

For more actions after file generation, please refer to the session Parameters

### Parameters

**Command call**: [-rxt] this is the command required for invoking the execution of the module

**List of Tables/View:** [Table\_Name\_1,Table\_name\_2…] can be one or many Table/View names. All tables should exists in the source and in the destination, and columns names and order should match.

**--p:** If any filter is desired for the extraction, then it must be specified here

E.g.

Select column1, column2, column3.. From Table1 where Table1.columnX = ?

You may pass the value using --p parameter. If no parameter are defined, then the module will extract the entire table. If parameter are specified then the dynamic query will be built with a WHERE clause including those parameters.

If you want to use the same value in multiples Tables/Viewa, then you must to use only the column name.

*Scenario No. 1*

Table1

Select column1, column2, column3 From Table1 where Table1.columnX = ?

Table2

Select column1, column2, column3 From Table1 where Table2.columnX = ?

If the value for columnX is the same in both SQL commands, you may define the parameter as:

--p columnX=’value’ or columnX=12345

*Scenario No. 2*

Table1

Select column1, column2, column3 From Table1 where Table1.columnX = ?

Table2

Select column1, column2, column3 From Table1 where Table2.columnX = ?

If the value for columnX is different in both queries, then you have to specify the value desired for each SQL file, using the file name as a prefix.

--p Table1.columnX=”value1”,Table2.columnX=”value2”

**--c:** Connection ID to be used to connect to the source database. The ID can be obtained from the file “settings.xml” or running the command:

java --jar JobStarter --rcl

**--sd:** Schema destination ID. Specify the Oracle schema where eventually the file will be uploaded

This parameter is important and required. Is used for the file naming convention, and later on used by the load process to identify the file in the input folder.

**--MOVE:** This parameter is only required if after the file generation you want to move/copy it to different location.

This parameter and --COPY are useful when the extraction and load is happening in the same server. Because the load process only read from the IMPUT folder, you might want to copy/move the file to the input folder once it’s generated, in order to be picked-up by the load process.

**--COPY:** This parameter is only required if after the file generation you want to move/copy it to different location. A copy of the original file will remains in the output folder.

**--ZIP:** This parameter is not required and indicate if after the file generation you want to compress the file in ZIP format. By default no compression is happening

**--JOBNAME:** This parameter allows you to identify the job in any report after successful or failure completion of job execution. Will be included as part of the subject on email report or any other way of reporting desired.

**--REPORT:** In this parameter you can indicated when you want to be notified. If the process FAIL, SUCCESS or BOTH. You might choose FAIL or SUCCESS or BOTH

**--REPORTBY:** How do you want to be notified?

* MAIL: based on the value passed on parameter --REPORT you will get an email indicating the status or error details. All mail will be sent to the distribution list configured in settings.xml file
* COPY: if this value is specified, then a copy of the LOG file will be copied in the output folder

This functionality was introduced for the GIO81 port forwarding, and because the server is not connected to any internal Boeing resource like SMTP server. This functionality will allow the app to reports a failure to the load process. The load process will identify the error file in the input folder and report the error by email.

* DATABASE: This function in not implemented, but it will allow to save in any oracle table the error for further revision

**--ARCHIVE:** with this parameter you indicate if after the process ends, you want to archive any resulting file, like LOGS generated files, CTLs, etc. Files will be zipped and stored in a folder under the connection name in the <archive-dir> path, configured in the settings.xml file.

### Complete command Example

java -Xmx1024M -jar JobStarter.jar -rxt bca\_active --p ac\_num=KC46 --c NWCS271A --sd DW737IC --MOVE="C:\Development\workspaces\default-configuration\JobStarter\target\classes\input" --ZIP --JOBNAME="Extract from MS SQL Server Database NWPACTIVE" --REPORT=FAIL --REPORTBY=MAIL

## CallSqlLoader

This module wrap the SQL Loader functionality and loads plain files into the database. The INPUT folder is the repository used by the module for loading into the database. INPUT folder path is defined in the settings.xml.

NOTE: INPUT folder is not created by any process, so if for any reason is deleted or a new installation is done, this folder must be created manually and the path entered in the settings.xml file.

### Logic

1. When the Cron job is started, the module will search in the INPUT folder for any file matching the pattern defined. E.g. SCHEMA\_\*unique\_file\_identifier\_\*
2. Will determinate if the idenfied file is zipped, based on the extension “.zip” and will proceed to unzip the file.
3. Then will create on the WORK directory a folder named as the Process ID. (Process id is shown at the beginning of the job execution and is given by the Operating System). This functionality ensure that no other process will miss up with the same file.
4. Depending of parameter --LOADLAST, if was specified, the process will load the latest file version (in case of multiples files for the same table and schema) if no parameter specified, the process will load all files one at the time, starting by the oldest. Repeating from the point 1.
5. SQL Loader use a Control files for loading plain files. This module has multiples option for CTL build
   1. Based on Metadata. The process will connect to oracle and extract the metadata from related Table, and will use it to build the ctl file.

NOTE: The order of the columns in the file must match the order of the table columns. Other way the process will fail, due to Column Size, Type or ending with wrong information loaded.

* 1. Based on Header included in the file. The process will use the file header to create the CTL

NOTE: Headers names must match table column names or the load process will fail

* 1. Based on provided CTL file. The process will use a CTL file existing in the CTL folder (path defined in settings.xml). To specify the use of a custom CTL you must use the parameter --CTL

1. Before load the file into the table, the process will refresh the destination table based on the value passed by parameter --REFRESH, if TRUNCATE was specified, the process will process to delete the entire table (information delete with TRUNCATE can’t be recovered). If DELETE was specified, then the process will use the condition specified in parameter --DELETEBY. If no DELETEBY was specified, the process will delete the entire table but using the DML command DELETE.

E.g.

(…) --REFRESH=DELETE --DELETEBY=column1=value1,column2=value2,…

1. If no RFRESH parameter is specified, then all records will be added to the destination table with the risk of getting duplicated records.
2. After all files were loaded, and if --ARCHIVE parameter was specified, the process will proceed with the archival of the folder in WORK\<PRODESS ID>. The folder will be zipped and copy/moved into the ARCHIVE directory.
3. If the input file has in the name the word ERROR, the process will trigger an error report, using the same parameters specified in current job --REPORTBY

## CallStoredProcedure

This module will execute any Oracle Stored Procedure and recover any output message created with DBMS\_OUTPUT.PRINT\_LINE.

### Parameters

**Command call**: [-rs] this is the command required for invoking the execution of the module

**List of Procedures:** [Procedure1,procedure2…] can be one or many procedure names.

**--p:** If procedures are expecting any parameter

E.g.

Procedure1(param1, param2, param3)

Command should look like

-rs Procedure1 --p param1=value1,param2=value2,param3=value3

You may pass the value using --p parameter and enclose it between “” if value has spaces on it.

If you want to use the same value in multiples procedure, then you must to use only the column name.

*Scenario No. 1*

Procedure1(param1, param2, param3)

Procedure2(param1, param2)

If the value for param1,2 and 3 is the same in both procedures, you may define the parameter as:

--p param1=”value 1”,param2=value2,param3=value3

If the value for param1,2,3 is different in both procedures, then you have to specify the value desired for each SQL file, using the file name as a prefix.

--p Procedure1.param1=”value 1”,Procedure1.param2=value2,param3=value3,Procedure2.param1,Procedure2.param2

**--c:** Connection ID to be used to connect to the source database. The ID can be obtained from the file “settings.xml” or running the command:

java --jar JobStarter --rcl

**--JOBNAME:** This parameter allows you to identify the job in any report after successful or failure completion of job execution. Will be included as part of the subject on email report or any other way of reporting desired.

**--REPORT:** In this parameter you can indicated when you want to be notified. If the process FAIL, SUCCESS or BOTH. You might choose FAIL or SUCCESS or BOTH

**--REPORTBY:** How do you want to be notified?

* MAIL: based on the value passed on parameter --REPORT you will get an email indicating the status or error details. All mail will be sent to the distribution list configured in settings.xml file
* COPY: if this value is specified, then a copy of the LOG file will be copied in the output folder

This functionality was introduced for the GIO81 port forwarding, and because the server is not connected to any internal Boeing resource like SMTP server. This functionality will allow the app to reports a failure to the load process. The load process will identify the error file in the input folder and report the error by email.

* DATABASE: This function in not implemented, but it will allow to save in any oracle table the error for further revision

### Complete command Example

java -Xmx1024M -jar JobStarter.jar –rs procedure\_refresh --p ac\_num=KC46 --c NWCS271A --JOBNAME="Procedure to refresh database" --REPORT=FAIL --REPORTBY=MAIL