Hi,

The option of building (exporting) job is used to export the job script.  
Export item option is used to export the item files, including jobs, metadatas, contexts, routines.  
If the job requirement is to run your job on different systems,  you can export the job script(.bat/.sh)  
Please have a look at the KB article about:[TalendHelpCenter:Exporting+a+Job+script+and+executing+it+outside+of+Talend+Studio](https://help.talend.com/display/KB/Exporting+a+Job+script+and+executing+it+outside+of+Talend+Studio?_ga=1.262141325.1436495098.1462245134)  
Best regards  
Sabrina

!Relational.ISNULL(Facilities\_extract.DOB)?TalendDate.parseDate("yyyy-MM-dd HH:mm:ss.SSS",Facilities\_extract.DOB): TalendDate.parseDate("yyyy-MM-dd","1900:01:01")

!Relational.ISNULL(Facilities\_extract.LastTimeRefreshed )?TalendDate.parseDate("yyyy-MM-dd HH:mm:ss.SSS",Facilities\_extract.LastTimeRefreshed ): TalendDate.parseDate("yyyy-MM-dd","1900:01:01")

!Relational.ISNULL(Facilities\_extract.DateCreated )?Facilities\_extract.DateCreated :TalendDate.getCurrentDate()

!Relational.ISNULL(Facilities\_extract.DateModified )?Facilities\_extract.DateModified :TalendDate.getCurrentDate()

--

"DECLARE @RC int

EXECUTE @RC = [dbo].[spHubCustInfoDataLoadQuery] "

To execute stored procedure in talend

Facilities\_extract.CustomerWCISID == null || Facilities\_extract.CustomerWCISID.length() == 0?"":Facilities\_extract.CustomerWCISID

Facilities\_extract.IntegrationCode == null || Facilities\_extract.IntegrationCode.length() == 0?"":Facilities\_extract.IntegrationCode

*PPK stands for PuTTY Private Key Header*

Files created by PuTTYgen are known as PPK files. PPK files are PuTTY Private Key

WinSCP is a freeware windows client for the SCP (secure copy protocol), a way to transfer files across the network using the ssh (secure shell) encrypted protocol.

## [Split Rows to Columns](https://umeshrakhe.wordpress.com/2015/01/09/split-rows-to-columns/) is possible in talend

**tFixedFlowInput**allows you to generate fixed flow from many input ways.

From the three options, select the mode that you want to use.

* **Use Single Table :**Enter the data that you want to generate in the relevant value field.
* **Use Inline Table :**Add the row(s) that you want to generate.
* **Use Inline Content :**Enter the data that you want to generate, separated by the separators that you have already defined in the **Row** and **Field Separator** fields.

**how to run talend job from command prompt**

<jobName>\_run.bat

 .bat is for windows and .sh is for unix.

To change the context variable at run time, add it after the original command so:

“<jobName>\_run.bat –context\_param input=<not standard>”.

Many people will want to schedule their talend jobs as recurring tasks to run every day, week, etc. In unix, this can be done with a cron job, and in Windows we can use the Windows Scheduler application. Find this application and run it.

**tFixedFlowInput** generates as many lines and columns as you want using the context variables.

**tFixedFlowInput**allows you to generate fixed flow from internal variables.

sh -x codename\_runsh

[‎6/‎16/‎2016 12:22 PM] Dhore, Milan (CONT):

there is new one method which will check first it there any kind of escape character

String[] arr = { "+","\\:","\n","\t","\r","&","\b","\f","\'", "-", "&&", "||", "!", "(", ")", "{", "}",

                "[", "]", "^", "\"", "~", "\*", "?", ":", "\\", "AND", "OR" };

A smoke test is basically just a sanity check to see if the software functions on the most basic level.

If your smoke test fails, it means there is no point in running your other functional tests. Smoke Testing is performed after software build to **ascertain that the critical functionalities of the program is working fine**

It is usually used as a time-saving step before the more thorough regression/integration/acceptance tests, since there is no point in running the full testsuite, if the thing catches fire anyway.

<http://www.guru99.com/smoke-sanity-testing.html>

**Regression testing** is a type of software **testing** that verifies that software previously developed and tested still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc.

Regression testing is testing existing software applications to make sure that a change or addition hasn’t broken any existing functionality. Its purpose is to catch bugs that may have been accidentally introduced into a new build or release candidate, and to ensure that previously eradicated bugs continue to stay dead.  By re-running testing scenarios that were originally scripted when known problems were first fixed, you can make sure that any new changes to an application haven’t resulted in a regression, or caused components that formerly worked to fail.  Such tests can be performed manually on small projects, but in most cases repeating a suite of tests each time an update is made is too time-consuming and complicated to consider, so an [automated testing tool](https://smartbear.com/product/testcomplete/overview/) is typically required.

Regression test is a test that is performed to make sure that previously working functionality still works, after changes elsewhere in the system.

**1) Corrective Regression Testing:** Corrective regression testing can be used when there is no change in the specifications and test cases can be reused.

**2) Progressive Regression Testing:** Progressive regression testing is used when the modifications are done in the specifications and new test cases are designed.

**3) Retest-All Strategy:** The retest-all strategy is very tedious and time consuming because here we reuse all test which results in the execution of unnecessary test cases. When any small modification or change is done to the application then this strategy is not useful.

**4) Selective Strategy:** In selective strategy we use a subset of the existing test cases to cut down the retesting effort and cost. If any changes are done to the program entities, e.g. functions, variables etc., then a test unit must be rerun. Here the difficult part is to find out the dependencies between a test case and the program entities it covers.

* Regression test is required even when a very small change is done in the code because this small modification can bring unexpected issues in the existing functionality.

ClientConfiguration clientConfig = new ClientConfiguration();

clientConfig.setProtocol(Protocol.HTTPS);

clientConfig.setProxyHost(ip\_address);

clientConfig.setProxyPort(port number);

clientConfig.setProxyUsername(user id on network);

clientConfig.setProxyPassword(password on network);

All Buckets /cof-commercialbank-qa-application-data/portfoliotracker/Dev/ACBS

In case of sudo su - it is a login shell, so /etc/profile, .profile and .bashrc are executed and you will find yourself in root's home directory with root's environment.

su - The command su is used to switch to another user (**s** witch **u** ser), but you can also switch to the root user by invoking the command with no parameter.

I understand what su - does

* logs you into root with the root environment
* (as opposed to su which logs you into root with your environment)

And I understand what sudo does

* you are root for one command

----------------------------------------------

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a **tLogCatcher** in order for the log data collected to be encapsulated and passed on to the output defined.

Provides a priority-rated message to the next component.

A **Junk Dimension** is a **dimension** table consisting of attributes that do not belong in the fact table or in any of the existing **dimension** tables. The nature of these attributes is usually text or various flags, e.g. non-generic comments or just simple yes/no or true/false indicators.

<http://www.1keydata.com/datawarehousing/junk-dimension.html>

The junk dimension is simply a structure that provides a convenient place to store the junk attributes. It is just a collection of random transactional codes, flags and/or text attributes that are unrelated to any particular dimension.

Late Arriving Dimension

**Data** scrubbing, also called **data cleansing**, is the process of amending or removing**data** in a database that is incorrect, incomplete, improperly formatted, or duplicated

STORED PROCEDURES

Isdate

Isnumeric in postgresql

select ('122.32325' ~ '^[0-9]+\.?[0-9]\*$');

CASE WHEN textregexeq(MP.PrimaryMPE,'^[[:digit:]]+(\.[[:digit:]]+)?$')= 't' then THEN CAST(MP.PrimaryMPE AS DECIMAL(19,2))

                        ELSE CAST(0 AS DECIMAL(19,2)) END AS PrimaryMPE,

oh

are you aware of etl job log tables

use this query

SELECT moment, pid, root\_pid, father\_pid, project, job, context, priority,

type, origin, message, code

FROM pt\_intg.etl\_job\_log order by moment desc limit 100

SELECT \*

FROM pt\_intg.etl\_rowcount\_log order by moment desc limit 100

first will tell u job progress

and second will tell u rowcount

ALTER TABLE cr\_pty\_type\_dim

OWNER TO etluser;

GRANT ALL ON TABLE cr\_pty\_type\_dim TO rds\_superuser WITH GRANT OPTION;

GRANT ALL ON TABLE cr\_pty\_type\_dim TO adsdb\_ownr;

GRANT SELECT, UPDATE, INSERT, TRUNCATE, DELETE ON TABLE ads\_rpt.cr\_pty\_type\_dim TO ads\_dml\_role;

GRANT SELECT ON TABLE cr\_pty\_type\_dim TO ads\_select\_role;

GRANT SELECT ON TABLE cr\_pty\_type\_dim TO dev\_role;

GRANT ALL ON TABLE cr\_pty\_type\_dim TO ads\_ownr\_role;

**scripts**

sh /opt/ACBS/build/cml\_ads\_credit\_party\_type\_dim/cml\_ads\_credit\_party\_type\_dim/cml\_ads\_credit\_party\_type\_dim\_run.sh >>/opt/ACBS/log/cml\_ads\_credit\_party\_type\_dim\_$(date +%Y%m%d).log 2>>/opt/ACBS/log/cml\_ads\_credit\_party\_type\_dim\_$(date +%Y%m%d).err

**Differnce between Context and Global Map:--**  
  
These are global areas used to store data that can be used by all components within a Talend job.  
Context variables are predefined prior to job execution in a context group whereas Global Map variables are created on the fly at any point within a job.

Below is a list of the differences between **tJoin** and **tMap**:

1. **tMap** can have many output links compared to **tJoin**, which only has main and reject links.
2. **tMap** supports writing expressions on the columns while providing the joining condition, whereas **tJoin** only supports exact matches between the keys.
3. **tMap** supports the storage of the look-up data on disk.
4. **tMap** allows you to reload the look-up data for each main record.
5. **tMap** supports more types of join model, includes unique join, first join, and all join, whereas **tJoin** only supports unique join. This article [(The differences between Unique match, First match and All matches)](https://help.talend.com/#/search/all?query=The+differences+between+Unique+match%252C+First+match+and+All+matches&filters=EnrichVersion%253D%25226.3%2522) discusses the differences between the different join models.
6. **tMap** allows you to link multiple look-up flows to it, and supports loading multiple look-up flows in parallel. **tJoin** only accepts one look-up flow.
7. **tMap** supports the 'die on error' option.

From the above list of differences, you can see that **tMap** is a powerful component that can handle more complicated cases compared to **tJoin**, which is basic. **tMap** generates more code while compiling, and it may take more space and time to load the data in the memory during the job execution than **tJoin**. Generally speaking, you should use **tJoin** if it is able to meet your requirements; otherwise, use **tMap**.