

Talend Standards and Best Practices

[1. Overview 3](#_Toc468834585)

[I. Purpose of the document 3](#_Toc468834586)

[II. Intended Audience 3](#_Toc468834587)

[2. Naming Conventions 4](#_Toc468834588)

[I. Job Naming Conventions 4](#_Toc468834589)

[II. Joblet Naming Convention 4](#_Toc468834590)

[III. Sub job/Step Naming Convention 5](#_Toc468834591)

[IV. Context Variable Naming Convention 5](#_Toc468834592)

[V. File Naming Convention 5](#_Toc468834593)

[VI. Folder Naming Convention 5](#_Toc468834594)

[VII. Test Case Naming Convention 6](#_Toc468834595)

[3. Best Practices 7](#_Toc468834596)

[I. Talend Workspace Path Should Not Contain Any Spaces 7](#_Toc468834597)

[II. Every Job Should Have an Entry and Exit Point 7](#_Toc468834598)

[III. Error Handling and Logging 7](#_Toc468834599)

[IV. Always Perform Null Handling 8](#_Toc468834600)

[V. Create Repository Metadata for DB Connections 8](#_Toc468834601)

[VI. Use Repository Schema for Files/DB and DB Connections 8](#_Toc468834602)

[VII. Create DB Connection Using t<Vendor> Connection 8](#_Toc468834603)

[VIII. Always Close the Connection to DB using t<Vendor> Close Component 8](#_Toc468834604)

[IX. Create a Document Corresponding To Every Talend Job 8](#_Toc468834605)

[X. Provide Subjob Title for Every Subjob 9](#_Toc468834606)

[XI. Avoid Hard Coding In Talend Job Component 9](#_Toc468834607)

[XII. Create Context Groups in Repository 10](#_Toc468834608)

[XIII. Use Talend.properties and tContextLoad to Intialize Context Variables 10](#_Toc468834609)

[XIV. Create User Routines/Functions for Common Transformation and Validation 10](#_Toc468834610)

[XV. Develop Talend Job Iteratively 11](#_Toc468834611)

[XVI. Always Rename Main Flows in Talend Job to Meaningful Names 11](#_Toc468834612)

[XVII. Provide Relevant Documentation for each component 11](#_Toc468834613)

[4. Appendix 12](#_Toc468834614)

[Best Practices - Performance Optimization 12](#_Toc468834615)

[I. Remove Unnecessary Fields/Columns using tFilter Component 12](#_Toc468834616)

[II. Remove Unnecessary Data/Records using tFilter Component 12](#_Toc468834617)

[III. Use Select Query to Retrieve Data from DB 12](#_Toc468834618)

[IV. Store on Disk Option 12](#_Toc468834619)

[V. Parallelism 12](#_Toc468834620)

[VI. Use Talend ETL Components When Required 12](#_Toc468834621)

[5. References 13](#_Toc468834622)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Created By** |
| 05/11/2016 | 1.0 | Initial Draft Version | Bhawana Singh |
| 10/11/2016 | 1.1 | Added the naming convention to be followed for Test Jobs. | Bhawana Singh |
| 07/12/2016 | 1.2 | Added component naming ,documentation details | Bhawana Singh |

# Overview

## Purpose of the document

This document discusses guidelines and standard practices for the development of all Talend projects on the UHG Talend environments. It is highly recommended that developers for all Talend projects review the Talend Standards and Guidelines before engaging in the actual development with Talend in the UHG environment.

All Talend projects on the UHG Talend servers will be monitored and enforced to ensure they adhere to the standards and common practices. Failure to comply will unnecessarily complicate the setup and promotion of the project’s code to other environments which will impact the project’s timeline.

## Intended Audience

The intended audience for this document includes the UHG Talend Administrators who are responsible for the setup and ongoing maintenance of the UHG Talend servers and the development teams whose Talend projects will be hosted in the UHG environments.

# Naming Conventions

## Job Naming Conventions

A Job on Talend must have a name that explains the purpose\intent of the job.

The Job should be named in the below format:

<Inbound\_outbound>\_<Functional\_Area>\_<Frequency>\_<Meaningful\_Job\_Description>

Eg: A daily membership outbound extract to be sent to XYZ:

**Ib\_Elig\_Dl\_ Xyz**

|  |  |
| --- | --- |
| **Job Type Abbreviation** | **Description** |
| Ib | Inbound Job |
| Ob | Outbound Job |

|  |  |
| --- | --- |
| **Functional Area Abbreviation** | **Description** |
| Mbr | Membership |
| Clm | Claims |
| Fin | Finance |
| Prv | Provider |

|  |  |
| --- | --- |
| **Job Frequency Abbreviation** | **Description** |
| Dl | Daily |
| Wk | Weekly |
| Mth | Monthly |
| Adh | Adhoc\ On Demand |

The last segment of job name, Meaningful Job Description, should give a meaning description to the job in no more than 20 Characters

## Joblet Naming Convention

A Joblet is a code snipped that will be re-used within a job one or more times.

The joblet should be named in the below format:

JL\_ Functional\_Area>\_<description>

Eg: **Jl\_Mbr\_Xyz**

## Component Naming Convention

The various Talend components will use the same name as provided in the Talend Palette. The names are auto appended with numbers in ascending order when the component with same name is used multiple times in the same job.

## Context Variable Naming Convention

Variables play a very important role in the code execution lifecycle. It is very important to give a meaningful name to these variables. This also makes it very easy for subsequent developers to understand what the variable is used for or what role it plays in the job.

## File Naming Convention

Below are the three different types of files that will be discussed in this segment:

1. Inbound Extract:
2. Outbound Extract:
3. Intermittent Files:
4. Context Files:

The files can be named as per the Business requirement. It is advisable to use timestamp while naming the files.

Note: Context files are used to hold different environment specific variables and should follow naming conventions as mentioned below:

* DEV:  "talend\_context\_dv02"
* DR:   "talend\_context\_dr"
* DR2:  "talend\_context\_dr2"
* PR2: "talend\_context\_prod2"
* PROD:  "talend\_context\_prod"
* STAGE: "talend\_context\_stg"
* TEST:   "talend\_context\_ts02"
* TST01:   "talend\_context\_ts01"
* TST03:   "talend\_context\_ts03"

Note: The above examples are just indicative in nature, however, it should follow similar nomenclature to avoid any confusion across platform during build process.

## Folder Naming Convention

The Folder naming convention applies to the directory structure that should be used for creating or consuming files during the execution of any job .It is advisable to use meaningful names that explain the content of the folder.

## Test Case Naming Convention

It is a good programming practice to develop keeping the testing strategy in mind. Talend comes with an in-built Testing feature that creates test case skeleton in a test instance.

A Test Instance is a set of data that allows you to run the Test Case with different parameters that you define (input, reference files, etc.).

The name of the test job should reflect two things:

1. It is a test job and not the actual job.
2. What does the particular job intend to test.

It is a good practice to name the test job as below:

**Test\_<actual\_Job\_Name>**: This way the developers can easily differentiate between the actual and the test job and the name itself would help identify the functionality that can be tested using this job.

# Best Practices

## Talend Workspace Path Should Not Contain Any Spaces

While creating the directory structure to create the workspace for a Talend Project, kindly ensure that the directories do not contain SPACES in the name:

Incorrect Path:

c:\Open Project\Talend Open Studio\workspace

Correct Path:

c:\OpenProject\TalendOpenStudio\workspace

Or

c:\Open\_Project\Talend\_Open\_Studio\workspace

## Every Job Should Have an Entry and Exit Point

All Talend Jobs need to start and end somewhere. Talend provides two basic components: tPreJob and tPostJob whose purpose is to help control what happens before and after the content of a job executes. I think of these as ‘Initialize’ and ‘WrapUp’ steps in my code. These behave as you might expect in that the tPreJob executes first, then the real code gets executed, then finally the tPostJob code will execute. Note that the tPostJob code will execute regardless of any devised exit within the code body (like a tDie component, or a component checkbox option to ‘die on error’) is encountered.

Using the tWarn and tDie components should also be part of your consideration for job entry and exit points. These components provide programmable control over where and how a job should complete. It also supports improved error handling, logging, and recovery opportunities.

One thing the developers should do for this Job Design pattern is to use the tPreJob to initialize context variables, establish connections, and log important information.

For the tPostJob: closing connections and other important cleanup and more logging.

## Error Handling and Logging

This is very important, perhaps critical, and if you create a common job design pattern properly, a highly reusable mechanism can be established across almost all your projects. My job pattern is to create a ‘logPROCESSING’ joblet for a consistent, maintainable logging processor that can be included into any job, PLUS incorporating well defined ‘Return Codes’ that offers conformity, reusability, and high efficiency. Plus is was easy to write, is easy to read, and yes, quite easy to maintain. I believe that once you’ve developed ‘your way’ for handling and logging errors across your project jobs, there will be a smile on your face a mile wide. Adapt and Adopt!

## Always Perform Null Handling

NULLS are not the same as “ ”(EMPTY String) and the behavior of NULLs is different than that of an empty string. Whenever a variable is used in any kind of expression\evaluation, a NULL check should always be performed. If this is not done and the variable\value comes out to be a NULL, Talend would throw a **NULL Pointer Exception** and cause the job to fail.

## Create Repository Metadata for DB Connections

Database connections should be created in the repository and used from the repository instead of defining and using it separately. The DB Table Schema should be imported at repository level so as to facilitate rapid development. Importing the schema for each job separately and at the individual job level is very time consuming.

## Use Repository Schema for Files/DB and DB Connections

It allows you to change the schema at one place, without having to change the schema in every job. Also, you don't need to open every job to find out if the changed schema is part of the Job or not. Changing at one place in Repository will allow you to change in every job.

## Create DB Connection Using t<Vendor> Connection

By using the t<Vendor>Connection (Vendor like Oracle MySQL), we can reuse the connection rather than creating a new connection for each component. Most of the time, there is a MAX connection limit defined at the DB level. By Using the connection already created, we can prevent reaching the max limit defined and hence causing the job to fail.

## Always Close the Connection to DB using t<Vendor> Close Component

Always ensure that your job closes the connection after the processing is completed. Not closing the connection can lead to the some problem as defined above

## Create a Documentation Note Corresponding To Every Talend Job

Creating a document for the job and adding the job details and version history to it helps understand the job and dependencies and also track the changes done over a period of time.

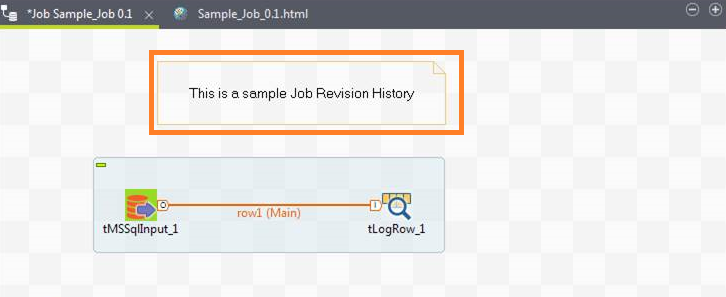
This will allow you to track the changes done on any Talend Job in the below format.

Revision History

1.0 04-10-2014 Initial Development

1.1 07-10-2014 Modification to Source and Target repository Schema

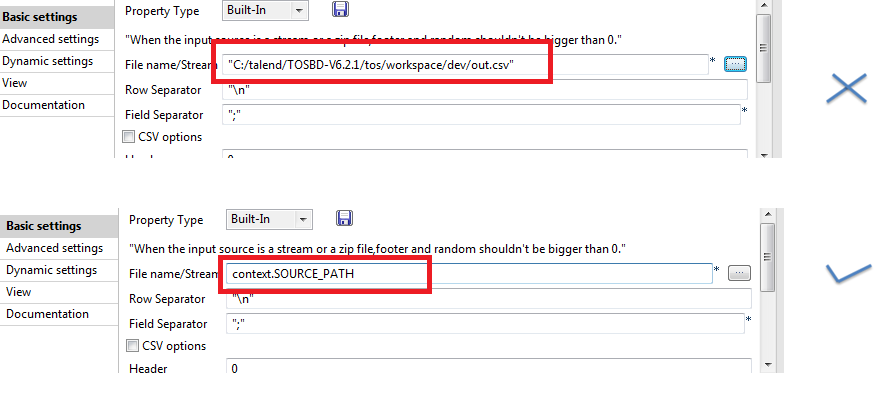
1.2 09-10-2014 Modification to transformation logic.

****

## Avoid Hard Coding In Talend Job Component

Hard coding is never a good programming practice. It hampers extensibility of code.

Developers should rather be using context variable to achieve this. This also ensures that if the same change is to be done at multiple places, the changes can be done at one place and the new requirement can be met.



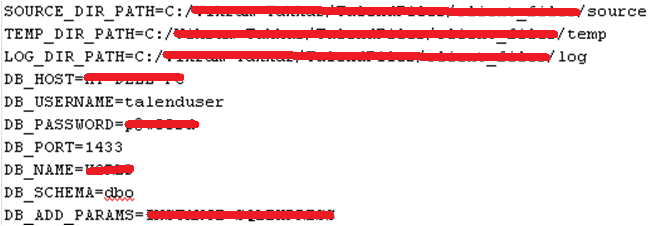
## Create Context Groups in Repository

Context Group will allow you to use the same context variables in any number of jobs without having to create again and assign value again to them. Imagine your Project requires 20 context variables and there are 10 jobs that require those context variables. It will be very difficult to create those context variables again and again in every job without context groups

You can create different context groups for different functionality of variables. For example, you can have different context group for database parameters, SMTP params and SFTP params etc.

## Use Talend.properties and tContextLoad to Intialize Context Variables

Always provide the value of context variables either through database table or through Talend.properties file. Below is sample of Talend.properties file.

[](http://1.bp.blogspot.com/-DyMACxnEuNY/VE8lNNL7zPI/AAAAAAAAKsw/a32kqUQ6L-I/s1600/Talend+Config+file+-+best+practice.png)

## Create User Routines/Functions for Common Transformation and Validation

Always create routines/functions for all common transformations and validation rules which can be used in multiple Talend jobs.

## Develop Talend Job Iteratively

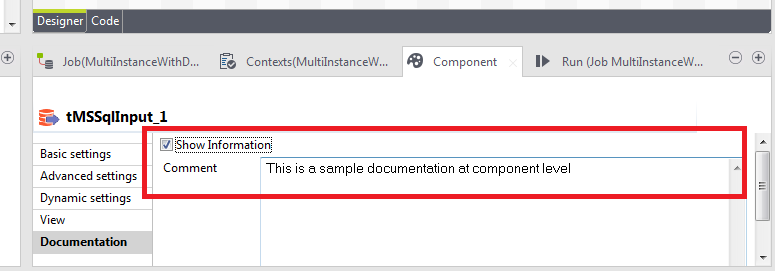
Modularity is a very important concept in software development. It makes the code more readable and maintainable. This also makes it easier to test the jobs, test one sub job and then move to the next one.

## Always Rename Main Flows in Talend Job to Meaningful Names

It is always good to rename the main flows in Talend Job to more meaning full names so that when you refer the fields in tMap component or using tFlowIterate it will be easy to refer and understand which data is coming from which flow.

## Provide Relevant Documentation for each component

It is always good to provide documentation for every component used in the job. One should check the show information option after adding the comments in the documentation section to be able to see the description just by hovering the mouse on that component.



# Appendix

## Best Practices - Performance Optimization

## Remove Unnecessary Fields/Columns using tFilter Component

It is very important to remove the data from the Job flow which is not required. This removal of data is done by using tFilter component.

## Remove Unnecessary Data/Records using tFilter Component

Similarly it is necessary to remove the data from the job flow which is not required in the Job. Having less data in your job flow will always allow your Talend Job to perform better.

## Use Select Query to Retrieve Data from DB

When data is being retrieved from database, it is recommended to use the select query in the t<DB>Input component, in order to select only the required data. In select query itself you can provide the required fields to fetch and also provide the where condition and filter only required data. By doing this only required data will be fetched in the job flow rather than complete table.

## Store on Disk Option

In Jobs that contain buffer components such as tSortRow as well as tMap, the basic configuration can be changed to store temporary data on disk rather than in memory.

For example, tMap,  select the option Store on disk for lookup data to be stored on a defined path. This will not allow taking the whole data into memory which will keep the memory available for operations and temp data will be fetched from disk.

## Parallelism

Most of the time we need to run few jobs/sub jobs in parallel to maximize the performance and reduce overall job execution time. In Talend Parallelism can be achieved in two ways.

* Use tParallelize component. (only available in Talend Integration Suite)
* Running Sub Jobs in Parallel by using the MultithreadedExecutions.

## Use Talend ETL Components When Required

ETL components are very handy and help to optimize performance of the job when transformation on data within a single database is needed to be performed. Benefit of using ETL component is that it will not unload the data from database tables into Job flow for performing the transformations.

# References

<https://help.talend.com/display/HOME/Talend+Big+Data>