ETL

Coding Standards

Version # : 1.0

Released on :

Author : R

Table of Contents

1. Introduction 3

1.1 Purpose 3

1.2 In Scope 3

1.3 Out-of-Scope 3

1.4 Contact Details 3

1.5 Intended Audience 3

1.6 Acronyms and Definitions 3

1.7 References 4

1.8 Requirement Summary 4

2. Snowflake dependencies & Design Considerations 4

Landing Prod/Minor/Major/Dev 6

3. Matillion Project Naming Considerations. 6

4. Design Considerations before Coding 7

4.1 Less Code 7

4.2 Job Design Complexity 7

4.3 Optimal Components usage 7

4.4 Technical Details for SDD/ISS 7

5. Guidelines 8

5.1 Process 8

5.1.1 Develop Code 8

5.1.2 IIT Code 9

5.1.3 Code Review 9

5.1.4 Export Code 9

5.1.5 Check in GIT 10

5.1.6 Check in Sharepoint 10

5.1.7 Update Release Notes 10

5.1.8 EDR Flow execution in CBIDEV 10

5.1.9 Implement Code in Test Minor/ Test Major 10

5.1.10 Implement Code in PROD 10

5.1.11 Prod Support Handover 11

6. Guidelines for Matillion Code Development 11

6.1 General 11

6.2 Error Handling 12

6.3 Coding Standards 12

6.4 Optimization 12

7. Appendix 13

8. Amendment History 13

# Introduction

## Purpose

The purpose of this document is to establish Matillion Coding Standards to be followed by BCBSRI & NTT teams that support EDR Supply & Demand applications.

This will be a live document that evolves with new design, development & deployment standards as and when BCBSRI & NTT teams mature in their understanding & utilization of Matillion & Snowflake toolsets.

## In Scope

NA

## Out-of-Scope

NA

## Contact Details

| Requested By | Name | Location | Contact Number |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Intended Audience

| Name – Primary Stakeholders | Role | Department |
| --- | --- | --- |
| Matillion Developers | Developers | NTT |
| EDR, D&A Teams |  |  |
| ADM Delivery Managers |  |  |
| Production Support Team |  |  |

## Acronyms and Definitions

| Acronym | Description |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## References

| Name |
| --- |
|  |

## Requirement Summary

|  |  |  |
| --- | --- | --- |
| **Requirement Number** | **Requirement Description** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Snowflake dependencies & Design Considerations

Matillion ELT platform is a SaaS services that is deployed on a snowflake instance. The “T” (Transformation) layer in ELT is typically executed as a SQL within Snowflake. i.e., the code that is designed & developed via Matillion UI is internally converted to Snowflake SQL. This makes Snowflake foundational to any Matillion code. Special attention to Snowflake DB/Schema/Table design is required while designing/developing Matillion Code.

The below speaks about the general standards/approach towards Snowflake DB/Schema design.



## Landing Prod/Minor/Major/Dev

Landing Prod/Minor/Major/Dev is the landing database that will hold schema’s for each individual application that is being implemented. (like EDR, FDA, VBP, CMS etc). The technical elements like Landing tables, Stages & Pipes should be created in appropriate Landing Database & Schema combination.

Below example illustrates how the naming conventions should be followed while creating landing/stage/pipe names.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Netezza/Source** | | **Snowflake Landing Database Object Names** | | | | |
| **Database Name** | **Table Name** | **Landing Database** | **Landing Schema** | **Landing Table Name** | **Snowflake Internal Stage** | **Snow Pipe Name** |
| EDRPRD1 | MEMBER | LANDINGPROD | EDR\_EDR | **LN\_**MEMBER | MEMBER**\_STAGE** | MEMBER**\_PIPE** |
| FDAPRD1 | CMC\_MEME\_MEMBER | LANDINGPROD | EDR\_FDA | **LN\_**CMC\_MEME\_MEMBER | CMC\_MEME\_MEMBER**\_STAGE** | CMC\_MEME\_MEMEBER**\_PIPE** |

# Matillion Project Naming Considerations.

Matillion projects names should clearly identify the application that is being implemented and the code needs to be appropriately grouped/managed. The project name should reflect the business name/definition of the underlying codebase.

Below are general guidelines for project naming standards within each Matillion Instance.

We should create group separate group for individual teams (Supply and Demand).

1. EnterpriseData\_Supply
   1. EDR
   2. FDA
   3. FDW
   4. INCOME
   5. UMCM
   6. MEMBERMART
2. EnterpriseData\_Demand
   1. PT
   2. VBP
   3. COB
   4. CMS

# Design Considerations before Coding

Before starting to develop code there are some design items to consider.

## Less Code

The developer should search for and attempt to use existing reusable code in the project. If not, the developer should write reusable code that will be used in the future rather than just for that requirement.

Example: We should create reusable job for File to Table loads with parameterize the Source path, filename and Table name etc. If we have any upcoming requirement to load new file to table, we should use the exiting reusable job with different parameters.

## Job Design Complexity

We should understand the complexity of requirement and discuss the same with in the team before working on design document. While designing, we should not plan to create any complex job. One job should be for one logical unit of work. Data processing should be broken up into Extraction, Staging, Validation and transformation. separate job should be created for each step. Job should be modular to make it easier to debug and more important, easier to modify or add new steps.

## Optimal Components usage

We should try to use as few components as possible in the jobs. Because using more components causes Matillion to generate more complex/outer queries, which effects the job performance.

Example: If we want to fetch only specific record from table we can use “Table Input component” to read data and “Filter” component to pass required records. Instead, we can use “Sql” component and write where condition in it to avoid “Filter” component.

## Technical Details for SDD/ISS

This section should contain below information:

1. Matillion Orchestration and Transformation job names and their details.
2. Job names should be as per standards.
3. Input, Intermediate and output table/file names for each job.
4. Should mention intermediate table types (Temporary/Transient/Permanent).
5. Parameters should be used and well explained for any criteria.
6. Mention if any existing reusable job is using.
7. DBA scripts also need to update with all DDL, DML scripts.
8. If possible mention under which requirement number one or more modules are created. Doing so would help in understanding the change that will come in future (Traceability Matrix).
9. If possible, mention the inputs and outputs details like the source and target teams/area. Doing so would help in seeing the larger picture.

# Guidelines

## Process

Here are the guidelines that expand upon the ten steps.

### Develop Code

* Design
  + Before beginning any code, do we have a good Design
  + Does the design promote code reuse
  + Are the Pseudo queries clear and easy to understand
  + While designing the Transformation and Orchestration jobs, focus should be on restart able. In case of Job abort, design should be such that Master Orchestration job can be triggered without affecting the data in the Database.
  + Rollback strategy for Table load should be kept in mind while designing the jobs. This helps in making the Orchestration job restart ability without affecting the Data in the tables.
  + While Implementing Truncate - Load on the tables, make sure no other process uses/loads the same tables.
* Matillion Code Development
  + Users should create their own version from Master branch and use that of development.
  + Users should take backup of created/updated jobs every day.
  + Follow all Standards for Variable names, Notes, comment, design flow, etc.
  + All modules validation should be successful.
  + When the unit testing is done, move the code up to the Development version.
  + No “CopyOf…” jobs are in the Development/Release/Master branch.
  + The variables should be flowed from master Orchestration job to child Orchestration/Transformation job in such a way that for any adhoc run variables value changes can be made at the master Orchestration job level.
  + If loop/Iterate functionality is needed for a specific requirement, i.e. certain jobs need to run in a loop, determine and use suitable Iterate component.
  + When a job is divided into multiple jobs to reduce the complexity, Transient tables should be used as intermediate tables.
  + Parameters for the job should be defined properly, keeping the standards. Maximum length for Parameter Names should not be more than 20. Description for Parameters should be understandable and explain the use of the parameter.
  + Job Variables should be kept to minimum.
  + If we are making the changes in existing jobs, then create copy of job and do the changes in that and make it actual after unit testing done.
  + In addition, if we are making changes to the existing table. Create copy of table in corresponding schema in Landing DB and use that until unit testing done.
  + Notify the Matillion admin to add environment variables if needed. The earlier the better.
  + Coordinate DBA work requests

### IIT Code

* IIT code (Integrated Testing)
  + Move code from individual version to Development branch after Unit Testing done or rename the copy of job to original if we are integrated with GIT.
  + Execute end to end flow and make sure flow should success.

### Code Review

* Matillion code review
  + Make sure the Release notes are available and up to date.
  + The Code Peer review document should be filled with Job names by Developer and provided to the Code reviewer.
  + Follow all standards for coding.
  + Have the proper Notes for all jobs
  + Validation should be completed.

### Export Code

* Export code in into Json file
  + If GIT integration is done, directly we can merge the code from individual branch to Development/Release/Masted branch. In this case we need not to export and import the code.
  + If not, we should export the jobs manually in the form of Json format.
  + Do the GIT check-in manually and use commit id for further deployments.
  + The code must go to and come from GIT at every step of the process thru testing to Production.
  + In case of a Change Request or job code change for an incident, only the changed code will be promoted to higher environments.

### Check in GIT

* Check the JSON file (code) into the GIT repository if Matillion GIT integration not setup
  + Use the corresponding GIT folder name as is in the Matillion category.
  + Without GIT information, Matillion ADMIN will not promote the code into higher environment.

### Check in Sharepoint

* All documents should be checked into Sharepoint. Sharepoint location will be shared.

### Update Release Notes

* Update Release Notes with GIT/Sharepoint versions and any other entries that are needed.

### EDR Flow execution in CBIDEV

Please email to RSS Team/ Project Team. If any team planning to execute the EDR flow in Development environment at least one day before execution.

### Implement Code in Test Minor/ Test Major

* Developer need to raise the request to merge the code into GIT Release branch.
* Developer needs to raise the Snow ticket with GIT detail and Job name for Matillion Admin.
* Matillion Admin will promote the code from GIT Release branch to Test Minor/ Major.
  + The Matillion code goes into the corresponding project in QA environment.

### Implement Code in PROD

* Data Center implements into PRD based on Implementation Plan documentation and Change Ticket.

### Prod Support Handover

* Developer needs to provide an updated Production Map Sheet to the Prod Support Team.
* KT needs to be given for the code to the Prod support team.
* Prod Support team needs to verify that all documents for the code are present at Worksite.

# Guidelines for Matillion Code Development

## General

The basic concepts of the Matillion tool – Extract, Load and Transform each have its own standards that will be followed to get new or changed code into production.

A detailed study of data should be done from the Database so that any data anomalies or any other condition can be added based on the data analysis. Data analysis helps in knowing and understanding the data and in turn helps in making correct joins leading to correct and successful delivery.

1. Extracts get data using Loading components through Orchestration job.
2. We should select all available columns from source tables/files and dump into landing DB until unless some specific reason to pull specific columns.
3. Use the required columns from landing table for further processing.
4. Error handling must be included in all extract processes. Error handling should consider that testers and production triage specialists don’t necessarily know the details of the Matillion code and will focus on the data first.
5. If the load appends records to a table or updates records in a table, there should be a way to identify each record set, such as a last updated timestamp.
6. The load process should have balancing and control data.
7. The transformation jobs uses the extract data as input and applies business rules and data assignments using appropriate components.
8. Transformation components format data, determine values based on if-then logic and combine data from multiple input sources.
9. Join components allow records from multiple sources/tables to be combined based on key values.
10. Notification emails should be sent for every batch (Start notification and finish/abort notification)
11. Validate log for each job in each environment in which the flow is being run and tested.

If we are using file as input in our code. Below point should be taken into consideration while doing UTC.

1. Testing should be done with Empty File.
2. Testing should be done only one record in Input File.
3. Testing should be done with more than one record in file.
4. Container path and format of the file should be finalized before completing the IIT.
5. Multiple file handing if the requirement is there.
6. Capture the rejected records from file.

## Error Handling

* Check the return code of all jobs called and traps any non-success status.

## Coding Standards

* Verify job name, component name, input file names are as per standards, ensure that the job developed adheres to the naming standards.
* All Orchestration Job names should start with ‘O\_’.
* All Transformation job names should start with ‘T\_’.
* All Environment variable names should start with ‘EV\_’ and should be in upper case only.
* All Grid Variable names should start with ‘GV\_’ and should be in upper case only.
* All Job Variable names should start with ‘JV\_’ and should be in upper case only.
* Create the Shared jobs with parameters which will use across the project. It will minimize the number of components and easy to maintance.
* Job and/or Grid variable names behavior should be ‘Copied’ for shared jobs.
* Try to map as much as possible job or grid variables from Main orchestration job to sun-orchestration/Transformation job.
* Make sure that the pathname/format details are not hard coded and job parameters are used for the same, these details are generally set as environmental variable.
* Avoid hardcoding, use appropriate variables instead of hardcoding.
* Component names should have appropriate and sensible names instead of default names.
* Print the job/grid variable values at starting the orchestration job.
* Job description must be clear, short and readable.
* Use Upper Case for column names and table names in SQL queries.
* Reject records should be inserted into reject table and use that for further analysis.
* Ensure that all the character fields are trimmed before any processing joining, if not it may lead to in correct data.

## Optimization

* If a Matillion job is taking longer to execute, then validate the design and make necessary changes if required.
* Jobs should not be overly complex. As the number of components are gets increase in a job, the complexity of the job increases. This further impacts the Performance of the job.
* Split the job if we have more complex logic and store intermediate data in Transient tables.
* Select the required columns instead of all.
* Use the suitable Warehouse depends on data volume and complexity.

# Appendix

# Amendment History

| Version | Date | Additions / Modifications | Prepared / Revised By |
| --- | --- | --- | --- |
| 1.0 | 12/20/2022 | Initial version |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |