Prepared by: Buzz IT Company Limited

Author: Steven Chen

Date: 03/02/2017

Version: 0.1

Maxim’s POS Polling Enterprise Service Bus Implementation Service

**EL-FY16-902**

**Sales/Master/Pricing Data Exchange**

Technical Design Specification

# Document Control

## Document History

| Version | Date | Author | Revision Remark |
| --- | --- | --- | --- |
| 0.1 | 03/02/2017 | Steven Chen | 1st draft |

## Document/Design Owner

| Name | Title |
| --- | --- |
| Steven Chen | System Analyst |
|  |  |
|  |  |

## Key Comments

| Name/Title |  | |
| --- | --- | --- |
| # | Comments |
| Comment | 1 |  |
| Response | 1 |  |

# Table of Content

[Document Control 2](#_Toc473712148)

[Document History 2](#_Toc473712149)

[Document/Design Owner 2](#_Toc473712150)

[Key Comments 2](#_Toc473712151)

[Table of Content 3](#_Toc473712152)

[1 Background 5](#_Toc473712153)

[1.1 Document Purpose 5](#_Toc473712154)

[1.2 Document Scope 5](#_Toc473712155)

[1.3 Document Audience 5](#_Toc473712156)

[1.4 Terms & Abbreviations 6](#_Toc473712157)

[1.5 Reference Materials 6](#_Toc473712158)

[2 Executive Summary 7](#_Toc473712159)

[3 Data Flows 8](#_Toc473712160)

[3.1 POS Clients to Staging 8](#_Toc473712161)

[3.2 Staging to EDW 8](#_Toc473712162)

[4 Database Schema 9](#_Toc473712163)

[4.1 Business Related Tables 9](#_Toc473712164)

[4.2 Non-Business Related Tables 9](#_Toc473712165)

[4.3 Pricing/Master Tables 9](#_Toc473712166)

[5 Design Approach 10](#_Toc473712167)

[5.1 System Technical Overview 10](#_Toc473712168)

[5.2 Design Approach – Application 10](#_Toc473712169)

[5.3 UI Application Technical Specification 11](#_Toc473712170)

[5.4 Batch Job Technical Specification 21](#_Toc473712171)

[6 Infrastructure 34](#_Toc473712172)

[7 Interface Clients Management 35](#_Toc473712173)

[7.1 History Table 35](#_Toc473712174)

[7.2 DBF File / CSV File 35](#_Toc473712175)

[7.3 OSB Log Files Removal 35](#_Toc473712176)

[7.4 Database Backup 35](#_Toc473712177)

[8 High Level Interface Design 36](#_Toc473712178)

[9 Functional Design 37](#_Toc473712179)

[10 Non-Functional Design 38](#_Toc473712180)

[11 System Interface Specification 39](#_Toc473712181)

[12 Interface Specification 39](#_Toc473712182)

# Background

## Document Purpose

The purpose of the Technical Design Specification (TDS) is to describe the detailed system design specification for a project and its main aim is to provide system design context for the project and its objectives. It will provide the input for system development activities.

The Technical Design Specification is part of the deliverables in the Business Case Development phase of Project Delivery Lifecycle.

## Document Scope

The scope of the Technical Design Specification (TDS) is to describe the technical views of the system. It has section such as Data Schema, Job Control Logic Specification, etc.

## Document Audience

The audience of this Technical Design Specification (TDS) is the technical staff of the IT department of the project owner.

## Terms & Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| ESB | Enterprise Service Bus |
| API | Application Programming Interface |
| EDW | Enterprise Data Warehouse |
| EOD | End of Day |
| POS Client | One Database Owner on the Sales side, provided by the POS machine vendor/manufacturer |
| DB | Database |
| POS | Point of Sales |
| Staging DB | The service bus database to stage the polling data |

## Reference Materials

| Document Names |
| --- |
| Maxim’s POS Polling ESB Implementation Service Proposal EL-FY16-902-v3.docx |
| POS Polling User Requirement Confirmation-20161121-Discussion Note.xlsx |
|  |

# Executive Summary

The aim of the ESB project is to migrate three data processing flows in Maxim’s current enterprise architecture using database provided technologies (linked server & stored procedure) to a new platform using Oracle Enterprise Services Bus technology. In the new ESB polling system (ESB system), three data processing flows will be implemented:

* Sales data real time processing to EDW
* Sales data EOD processing to EDW
* Synchronize master data to POS clients (e.g. Pricing/Master)

The ESB system will use JDBC to connect to the databases of existing POS clients in outlets by pre-configured connection information in order to collect sales data from the existing POS systems and update the pricing/master data back to the existing POS clients. Referring to the polling logic found in the production IT51 server, for any connection error, the ESB system will log down the error and retry data synchronization. After reaching maximum retry count, the ESB system will halt the synchronization for that particular POS client and generate alert to related parties for follow up.

All POS clients’ connection settings are configurable and maintainable by Maxim’s IT. It makes the adding of new POS client easy and no alteration of programming code is required. The ESB system can invoke several configurable concurrent threads to poll/push the data to/from POS clients concurrently for maximizing system performance.

# Data Flows

## POS Clients to Staging

## Staging to EDW

# Database Schema

## Business Related Tables

## Non-Business Related Tables

## Pricing/Master Tables

# Design Approach

## System Technical Overview

## Design Approach – Application

### MVC Architecture

The application employs Spring framework version 4.x, which adopts Model-View-Control architecture:

* Model, which represents the business objects and communicate with other tiers
* View, which represents the page design code (mainly JSF).
* Controller, which controls the page navigation and invoke calls on model objects.

### Design Pattern

The application employs the following patterns in the business layer:

* Front Controller: This controller serves as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.
* Data Access Objects (DAO): This is used to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data.
* Business Delegate: The Service classes are used to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the other systems.

### Data Access Framework

The data access framework will be available for all portal applications extending the framework. All database access logic should inherit from this framework. Hibernate is used in the data access framework, all HQLs and SQLs are maintained in the XML configuration files.

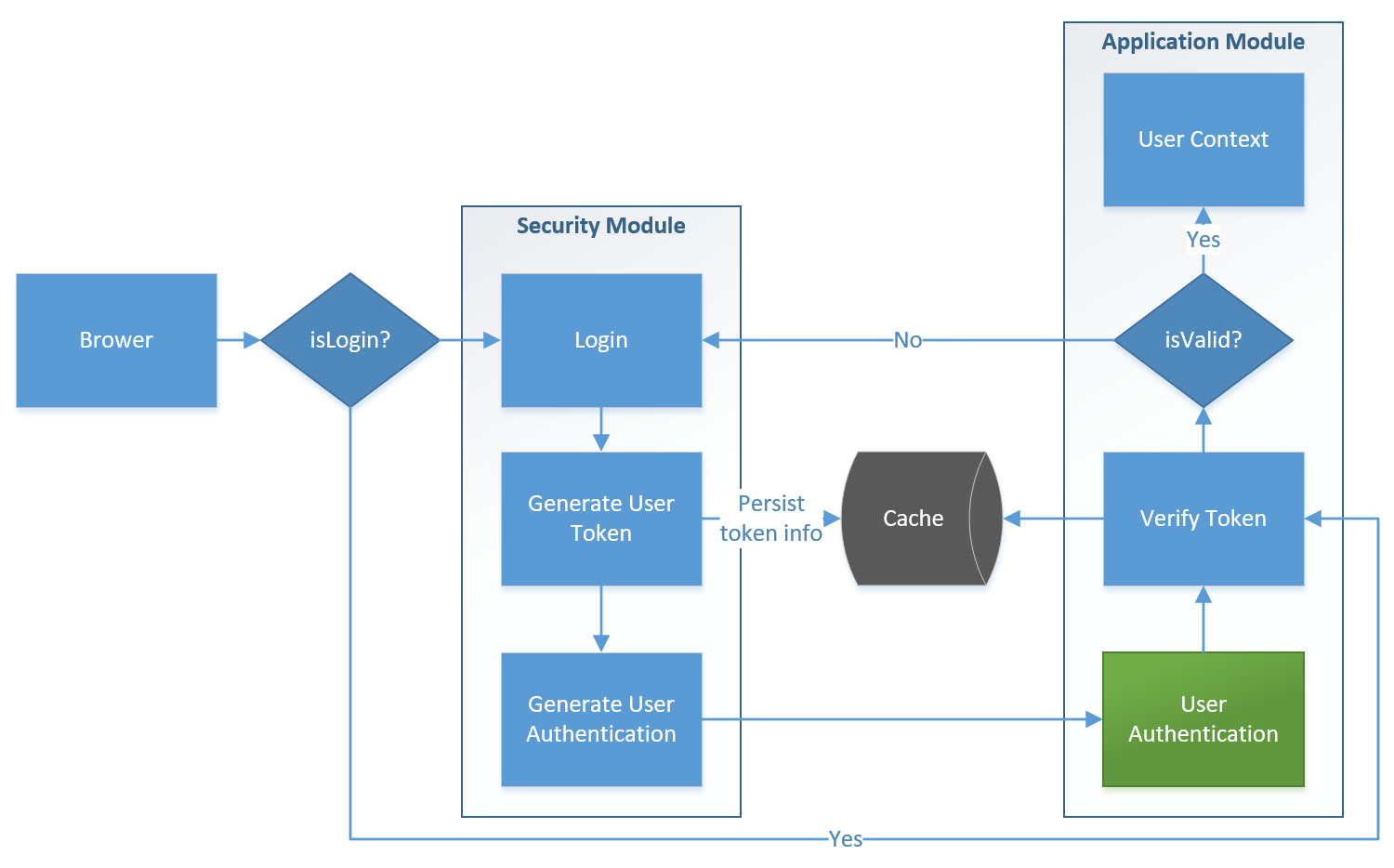
### Logging Framework

The logging framework will be available for all portal applications extending the framework. All logging access logic should inherit from this framework. Options will be available for the caller to log to the database (using the data-accessing framework) and/or a text file via log4j.

## UI Application Technical Specification

### Common UI Application Logic

#### User Authentication



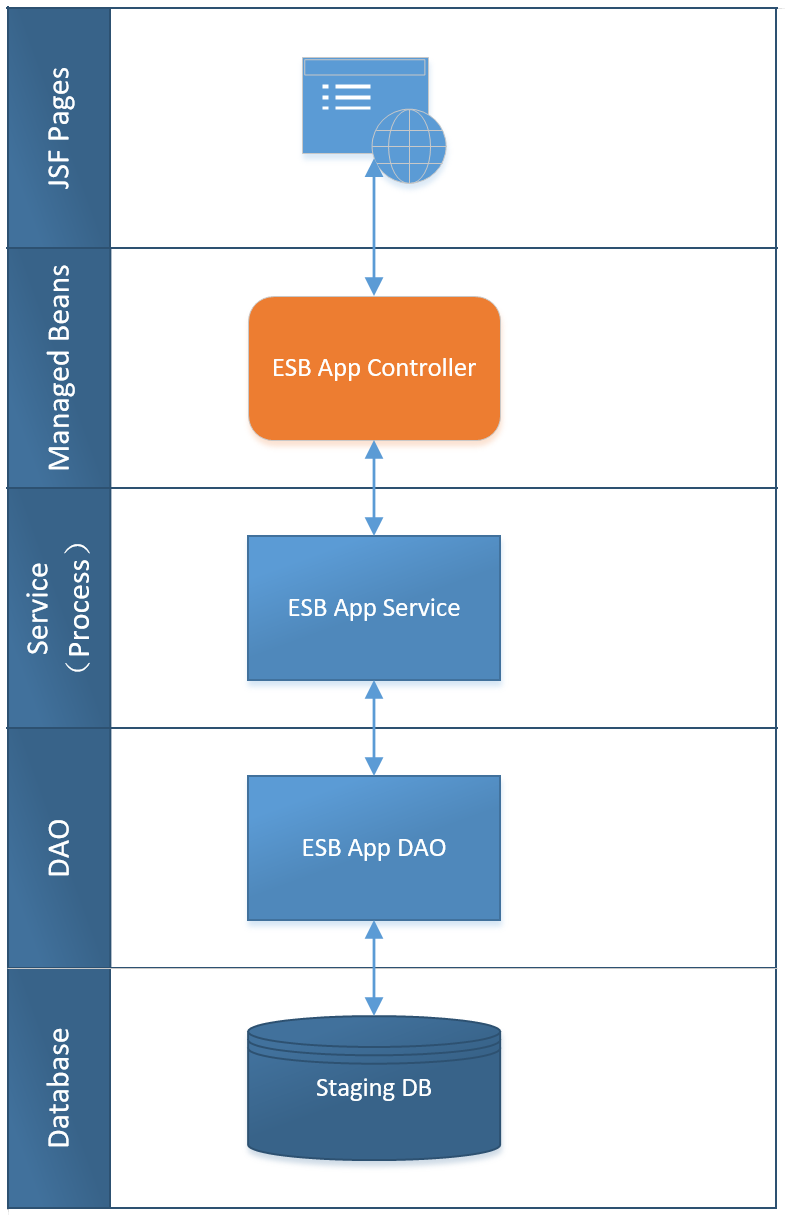
1. The user login page, that triggers the user module to generated a token for the user. The token is composed of the JSESSIONID and the login timestamp.
2. After the token is generated by the user module, the system will:
   1. Persist the corresponding user id and session token to SECURITY\_SESSION table in cache database. The token expiry time was set when the user login the application and will not be updated since then. That indicates how long the token will be expired for cross applications authentication after the user has login the application.
   2. Generate an user authentication object and redirect to application Url
3. Through the redirect, and the token will be passed from the user module to application page
4. The controller class of the page in application module will verify for the existence of the token in user authentication object
   1. If the token was found, the system will cross-check with cache for its validity, and create a user context object, together with the function access list in the session if appropriate.
   2. Else, the system will check if any valid user object can be found in the session. The existence of the session user object implies that the user has logged into application module.
   3. If both token and session user object cannot be found, the user will be redirected to the login page for login.
5. With the session user object created by the token, the user can navigate through the application without any additional login, and a seamless integration between user module and application module can be achieved.
6. When user logout the application, the user will be redirected to the user login page and the corresponding user record will be deleted from SECURITY\_SESSION table. This action has no effects on any existing session objects.

#### Function Access Control

1. During the user login, list of allowed functions will be kept as one of the user info in the session after passing user authentication (whole login process will be handle by calling functions in module esb-security-core.).
2. Browser (Presentation layer) sends the request to trigger an action which is handled by corresponding controller class.
3. The controller class will invoke the service class for further business processing.
4. The particular process will be invoked by the service to execute the required action.
5. User object stored in the session will be checked whether that particular action is allowed for the user. It will proceed the action if allowed and throw exception if not.
6. The results are displayed on the browser according to the result.

\* This function access control will apply on ALL UI functions listed below. Therefore, the function list kept in the session will be checked upon all entry points of the functions. It is not illustrated below for clarity of the mentioned flow.

#### Typical Maintenance Page Data Retrieval Logic



1. Browser (Presentation layer) sends the request to trigger an action which is handled by corresponding web service / action class.

2. The web service / action class will invoke the service class for further business processing.

3. The particular process will be invoked by the service to execute the required action.

4. The process determines the necessary data to be presented and invoke the DAO.

5. The DAO retrieve data from the Maxim Staging database and return as an entity.

6. The entity is then transformed as DTO in the process and return to the service.

7. The service gathers all necessary info and returns the DTO response back to the web service / action class.

8. The results are displayed on the browser appropriately.

Exception caught in any class is thrown back to the caller class until to the web service/action class that will pass exception list to the page and display on the browser.

#### Typical Maintenance Page Add / Edit / Delete Logic

1. Browser (Presentation layer) sends the request to trigger an action which is handled by corresponding web service / action class. This is done via an AJAX call such that a page refresh is not necessary.

2. The web service / action class will invoke the service class for further business processing.

3. The particular process will be invoked by the service to validate the input according to the business rules and execute the required action.

4. The Add / Edit / Delete operations are performed according to the logic below:

i. Edit: The process determines the data to be edited and invoke the DAO to retrieve data from the Maxim Staging database and return as an entity which is then updated by the process.

ii. Add: The process creates the entity according to the inputs.

iii. Delete: The process determines the data to be deleted and pass those record Ids to DAO.

5. DAO is invoked to execute the action.

6. The service gathers all necessary info and returns the response back to the web service / action class.

7. The results are displayed on the browser appropriately.

Exception caught in any class is thrown back to the caller class until to the web service/action class that will pass exception list to the page and display on the browser.

#### Typical Maintenance Page Import Logic

1. Browser (Presentation layer) sends the imported file to trigger the corresponding action class.
2. The web action class then validate the import file and parse the imported file data into business bean.
3. The action class invoke the service class for further business processing.
4. The particular process will be invoked by the service to execute the required action.
5. The process determines the data and invokes the DAO to insert data into STG database.
6. The service gathers all necessary info and returns the response back to the action class.
7. The results are displayed on the browser appropriately.

Exception caught in any class is thrown back to the caller class until to the action class that will pass exception list to the page and display on the browser.

#### Typical Maintenance Page Export Logic

1. Browser (Presentation layer) sends the request to trigger an action which is handled by corresponding web service.
2. The web service will invoke the service class for further business processing.
3. The particular process will be invoked by the service to execute the required action.
4. The process determines the necessary data to be presented and invoke the DAO.
5. The DAO retrieve data from the Staging database and return as an entity.
6. The entity is then transformed as DTO in the process and return to the service.
7. The service gathers all necessary info and returns the DTO response back to the web service.
8. The DTO is retrieved by the browser and parses into CSV string.
9. The browser sends the CSV string to corresponding web service and the web service will parse the CSV string into a CSV file.
10. The web service sends back the response with a CSV file attachment.
11. The CSV file is downloaded on the browser appropriately.

Exception caught in any class is thrown back to the caller class until to the web service that will pass exception list to the page and display on the browser.

### Main Module Design

#### Common Module

| **Package** | **Component** | **Description** |
| --- | --- | --- |
| Controller | CommonController |  |
| Service | AuditLogService |  |
| AuditLogServiceImpl |  |
| PollBranchSchemeService |  |
| PollBranchSchemeServiceImpl |  |
| PollSchemeInfoService |  |
| ScheduleJobService |  |
| SchemeQuartzTaskExecutor |  |
| SpringBeanUtil |  |
| DAO | AuditLogDao |  |
| PosDaoCmd |  |
| Queries |  |
| ScheduleJobDao |  |
| AuditLogDao |  |
| PosDaoCmd |  |
| Queries |  |
| ScheduleJobDao |  |
| Model | AuditLog |  |
| BranchInfo |  |
| BranchScheme |  |
| SchemeInfo |  |
| SchemeJobLog |  |
| SchemeRecordJobLog |  |
| SchemeScheduleJob |  |
| SchemeTableColumn |  |

#### POS Polling Module

#### Security Module

### Branch Info Maintenance

#### Maintain Branch Info – List

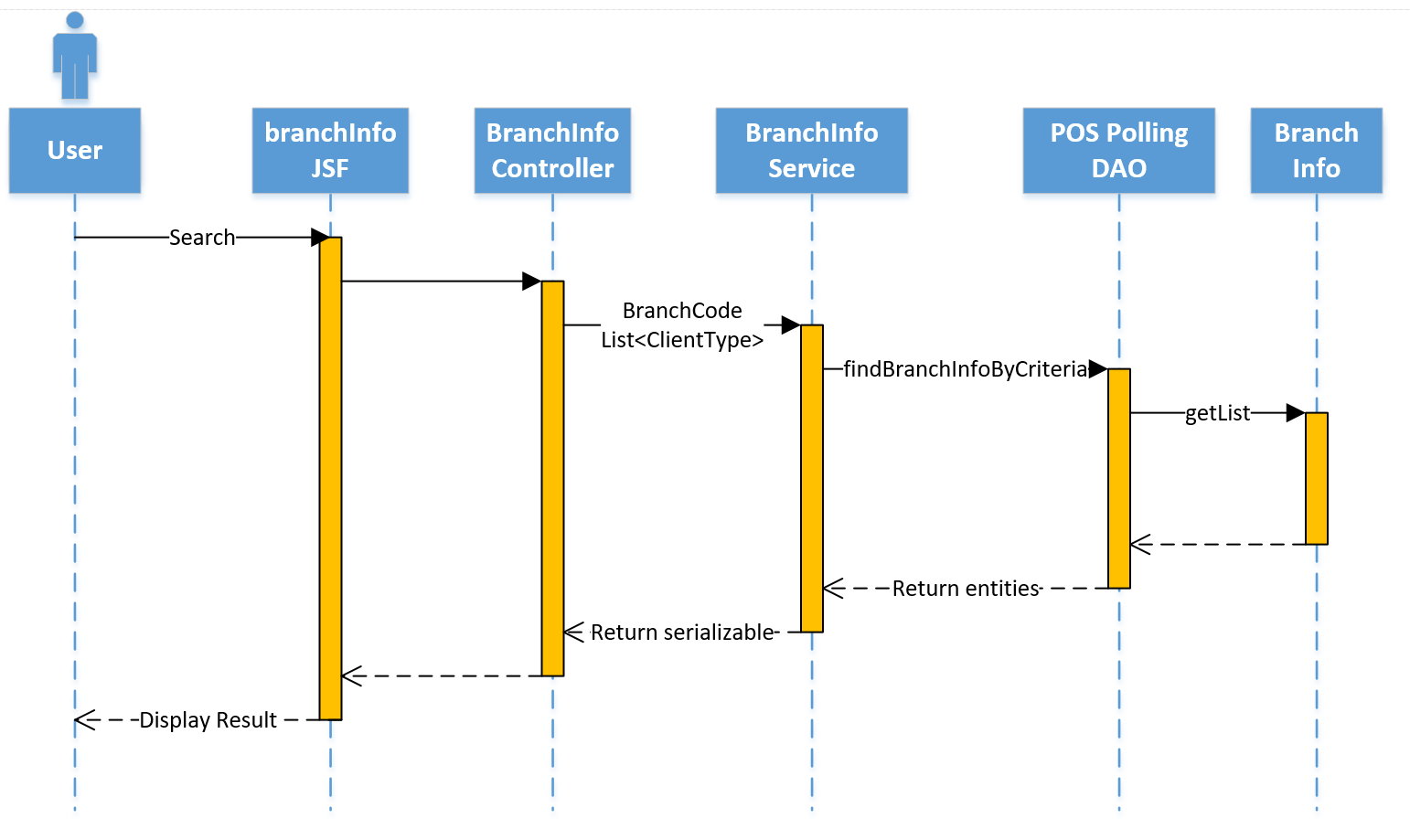
**Page Access Control**

This page can be accessed by any login user of with Administrator role with view and search functions.

**Program Flow**

The typical Branch Info maintenance page operations are applied, and the diagrams below summarize the sequence flows.  
(Please refer to section 5.3.1.1 [Typical Maintenance Page Data Retrieval Logic](#_Typical_Maintenance_Page))

Detail mapping between UI and data base fields can be referred to the *User Interface - Table Field Mapping* in SDS.



**Sorting**

Default Sorting of Branch Info List, Prioritized by sequence number

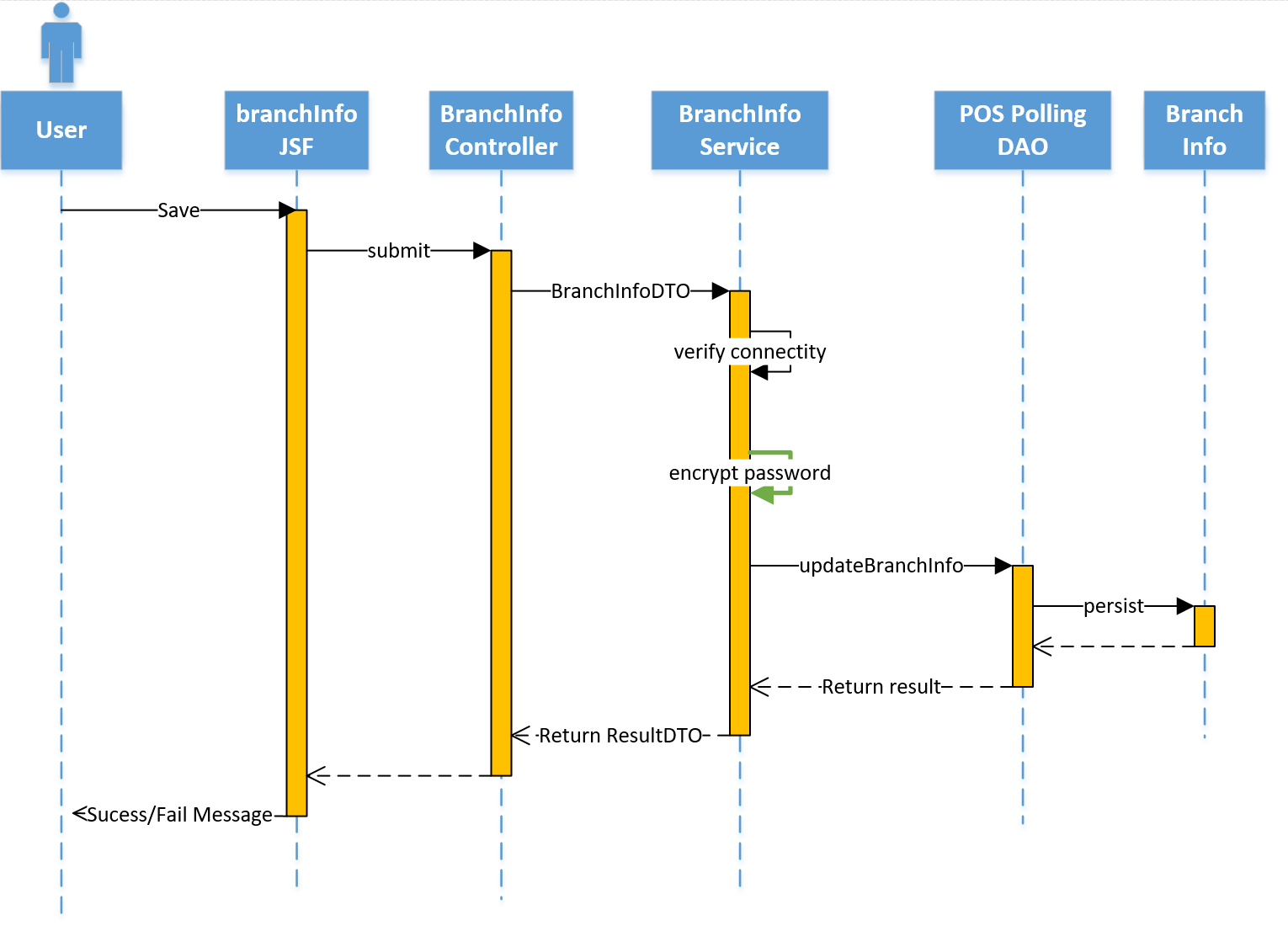
| **Sort column** | **Description** | **Pseudo Code** |
| --- | --- | --- |
| **Branch Code** | Sorted ascending by value. | order by branchCode asc |

**Filtering**

| **Filter** | **Description** | **Pseudo Code** |
| --- | --- | --- |
| **Branch Code** | Input value as keyword | branchCode like ‘%{input}%’ |
| **Client Type** | Allow multiple options | clientType in (‘{input1}, {input2}, …’) |

#### Maintain Branch Info – Create/Edit/Delete

Program Flow



Validation

| **Validation** | **Description** | **Pseudo Code** |
| --- | --- | --- |
| **Branch Code** | Input value as keyword | branchCode like ‘%{input}%’ |
| **Client Type** | Allow multiple options | clientType in (‘{input1}, {input2}, …’) |

### Scheduler Job Maintenance

#### Maintain Scheduler Jobs – List

**Page Access Control**

This page can be accessed by any login user of with Administrator role with view and search functions.

**Program Flow**

**Sorting**

**Filtering**

#### Maintain Scheduler Jobs – Edit

Program Flow

Validation

### Branch Scheme Maintenance

#### Maintain Branch Scheme – List

Screen Flow

Sorting

Filtering

#### Maintain Branch Scheme – Create/Edit/Delete

Screen Flow

Validation

#### Maintain Scheme Info – List

Screen Flow

Sorting

Filtering

#### Maintain Scheme Info – Create/Edit/Delete

Screen Flow

Validation

#### Maintain Table Column – List

Screen Flow

Sorting

Filtering

#### Maintain Table Column – Create/Edit/Delete

Screen Flow

Validation

#### Job Log View – List

Screen Flow

Sorting

Filtering

#### Job Log View - Detail

Screen Flow

Sorting

Filtering

## Batch Job Technical Specification

### Common Job Control Logic

#### Job submission

Parameter:

* Job Name
* Batch ID (the primary key of the Job Log)
* Poll branch scheme

Job Control for Scheduler Job Log

* Start Time/End Time (monitor job run period)
* Job Status (log the status of job)
* Latest Job Indicator (prevent duplicate submission)

#### Job Completion

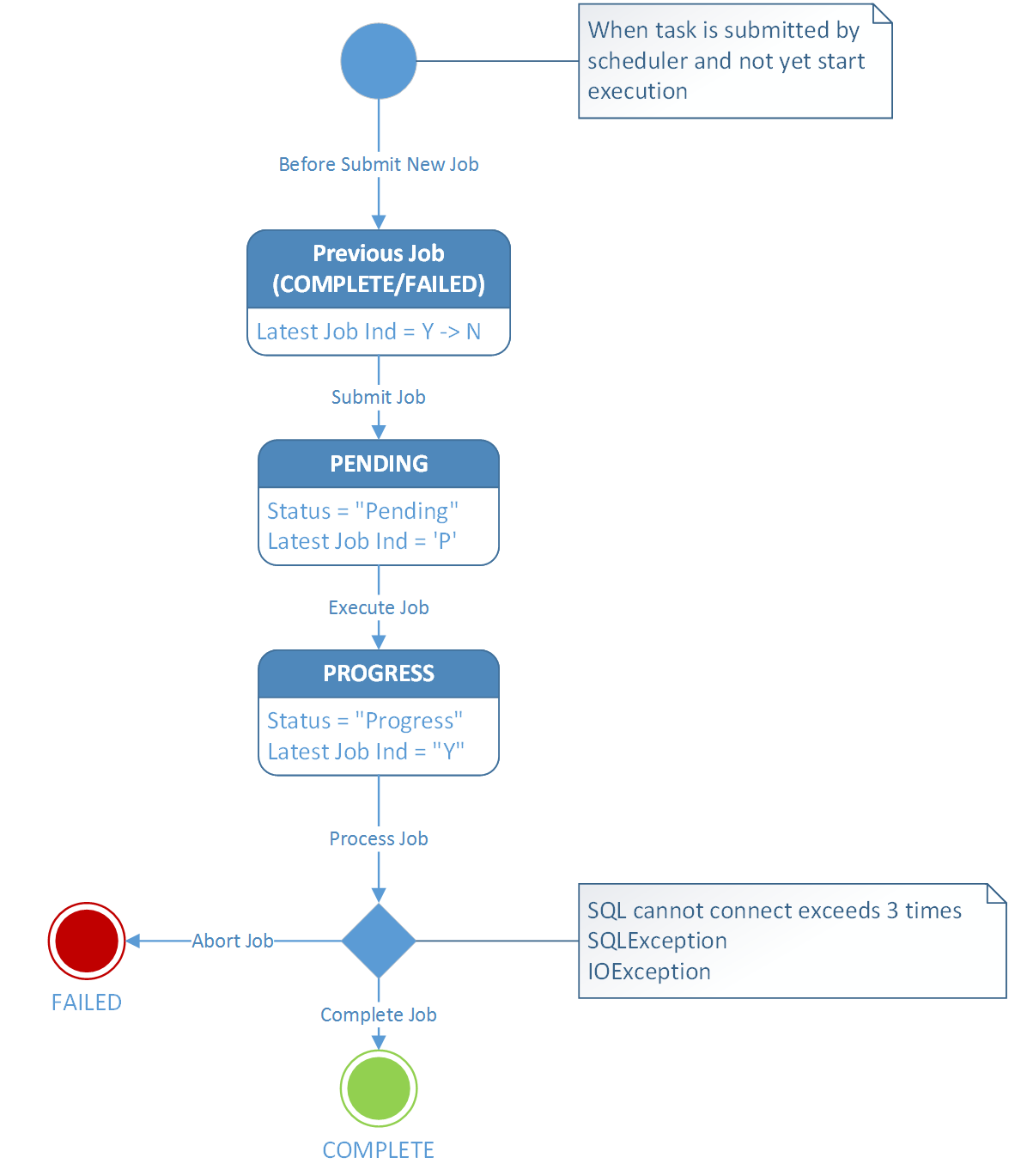
Scheduler

* Number of scheme processed (number of task submitted)
* Error Code
* Error Message

Task

* Parent (scheduler) ID – trace record of which scheduler submits this task
* Error Code
* Error Message
* Direction – poll scheme detail
* Poll Scheme name – poll scheme detail
* Branch Code – poll scheme detail
* Log detail
  + From table/to table
  + Number of records processed
  + Number of records insert
  + Number of records update
  + Number of records delete
  + Exception data (SQL state, error line text, etc…)

#### Job/task state logic



1. Before new job/task is submitted, update the previous job’s job log
   1. Update latest job indicator from “Y” to “N” where latest job indicator = “Y” and status in (“COMPLETE”,”FAILED”).
2. Upon submission of the new job/task, write a new job/task log into database with status = “PENDING” and latest job indicator = “Y”, managed within the same transaction with the job submission. Then the job submission encounter failure, the persistence of new log/task log will be rolled back.
3. After the job is successfully executed by the task executor,
   1. Update status from “PENDING” to “PROGRESS”;
   2. Update start time = current time
4. If encountering exceptions, update the log’s status to “FAILED” and end current job/task
   1. Update status from “PROGRESS” to “FAILED”
   2. Update end time = current time
5. When the job/task successfully finish, update the log’s status to “COMPLETE” and end it.
   1. Update status from “PROGRESS” to “COMPLETE”
   2. Update end time = current time

#### Job state logic

### Sales Data Real Time Polling Batch Job

#### Sales Data Real Time (POS - Staging) Scheduler

Job Detail Steps (Pseudo Code)

**Run method (with cron trigger)**

Find previous scheduler job log by criteria (where latest job ind = ‘Y’ and status in (“FAILED”, “COMPLETE”));

If(result > 0) {

result.get(1).setLatestJobInd = “N”;

Insert new scheduler job log;

}

Else {

Logger “Skip Scheduler Job ID = ” + current scheduler Job Id

Throw InterruptedException

}

Find poll branch scheme by criteria (where enable = true and (not exist in hist\_possystem where business date = today’s date & and branch code = branch code)

int countSchemeSubmitted = 0;

For (list: poll scheme array list) {

Try{

Submit task to taskExecutor (scheduler job log Id, poll branch scheme);

++ countSchemeSubmitted;

}

Catch() {

Log job submission exception with scheduler job log Id and poll branch scheme;

}

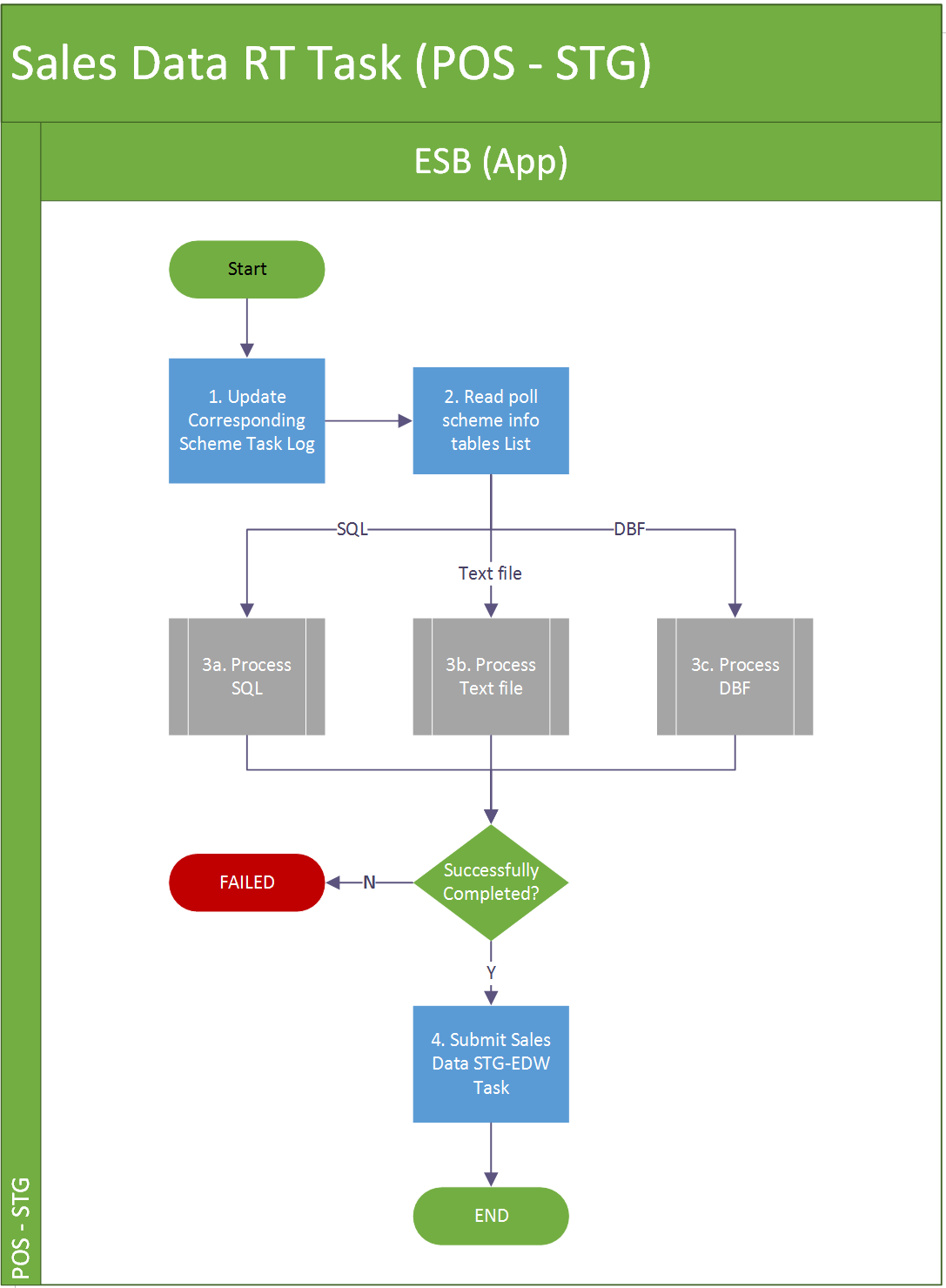
}

##### Data Flow/Details

#### Sales Data Real Time (POS - Staging) Task

##### Job Details

**Job Flow Diagram**



**Job Run Steps (Pseudo Code)**

Find previous **task** job log by criteria (where latest job indicator = ‘Y’ and status in (“FAILED”, “COMPLETE”) and poll scheme ID = current poll scheme ID);

If (result > 0) { //the result size suppose to be only one

result.get(1).setLatestJobInd = “N”;

Insert new scheduler job log;

}

Else {

Logger “Skip Scheduler Job ID = ” + current scheduler Job Id

Throw InterruptedException

}

If (PollScheme.getPollBranch().getClientType equals (ClientType.SQLSERVER.toString()){

processSQLServer();

}

Else if ((PollScheme.getPollBranch().getClientType equals (ClientType.ORACLE.toString()){

processOracleDB();

}

Else if ((PollScheme.getPollBranch().getClientType equals (ClientType.DBF.toString()){

processDBF();

}

Else if ((PollScheme.getPollBranch().getClientType equals (ClientType.CSV.toString()){

processCSV();

}

Else if ((PollScheme.getPollBranch().getClientType equals (ClientType.TEXT.toString()){

processTextFile();

}

Else{

Throws IllegalClientTypeException();

}

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

### Sales Data EOD Polling Batch Job (POS-Staging)

#### Sales Data Real Time (POS - Staging) Scheduler

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  |  |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

#### Sales Data Real Time (POS - Staging) Task

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  | Triggered |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

### Pricing/Master Data Generation (Pricing Server - Staging)

### Pricing/Master Data Distribution Batch Job (Staging-POS)

#### Pricing/Master data distribution (POS - Staging) Scheduler

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  |  |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

#### Pricing/Master data distribution (POS - Staging) Task

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  |  |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

### Backup & Housekeeping

#### Daily Sales Data Backup Procedure Job

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  |  |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

#### Server Bus Staging Data Purging Job

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Job Name** | **Job Type**  **(Scheduler/Task)** | **Class Names** | **Interfaces (DTO)** | **Runtime (Cron)** |
|  |  |  |  |  |

##### Job Details

**Business Logic**

**Job Flow Diagram**

Refer to system design spec

**Job Run Steps (Pseudo Code)**

##### Data Flow/Details

**Data Flow**

**Text file samples:**

**DBF file data samples:**

**SQL Server table data sample:**

**Data Sources mapping**

# Infrastructure

# Interface Clients Management

## History Table

## DBF File / CSV File

## OSB Log Files Removal

## Database Backup

# High Level Interface Design

# Functional Design

# Non-Functional Design

# System Interface Specification

# Interface Specification

.

- End -