

**Message Delivery Engine**

**Technical Design**

|  |  |
| --- | --- |
| Project Name: | Lead and prospect Management |
| External Reference Number: | None |
| Version Number: | 4.1 |
| Date Updated: | February 18, 2019 |

# Document Revision History

(Add relevant version, date and revisions to table).

|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Update Date | Update Author | Section, Page(s) and Text Updated |
| 1.0 | 11/2/2017 | Matt fisher | Initial Draft. |
| 2.0 | 03/23/2017 | Zhiyi Huang | Updated Tech Doc |
| 3.0 | 04/02/2018 | Brigadesh Chandrasekar | Added Load Balancing Section |
| 3.1 | 04/06/2018 | Zhiyi Huang | Updated Diagrams |
| 3.2 | 05/10/2018 | Zhiyi Huang | Add vCard message type |
| 3.3 | 07/30/2018 | Zhiyi Huang | Auto Response Queue |
| 4.0 | 10/25/2018 | Zhiyi Huang | SMS Suppression Framework |
| 4.1 | 02/18/2019 | Zhiyi Huang | Store/SMS Unsuppression |
| 5.0 | 6/28/2019 | Akash Ravi | Einstein Language |

[1. Document Revision History 2](#_Toc528235511)

[2. Introduction 4](#_Toc528235512)

[3. Application And Design 4](#_Toc528235513)

[3.1 High-Level Functionality & Assumptions 4](#_Toc528235514)

[3.2 High-Level Technical Design 4](#_Toc528235515)

[3.3 Heroku Add-ons required in solution and their intented functions 5](#_Toc528235516)

[The below add-ons have been identified by the Salesforce Heroku architects. 5](#_Toc528235517)

[3.4 Javascript Libraries used within the system 5](#_Toc528235518)

[3.5 Heroku data models and data dictionary 5](#_Toc528235519)

[3.5.1 Postgres tables: 5](#_Toc528235520)

[3.6 Job Queue and Job processing logic 10](#_Toc528235521)

[3.6.1 Dispatcher Functionality 10](#_Toc528235522)

[3.6.2 Parse Queue Worker Design 12](#_Toc528235523)

[3.6.3 Time Zone Queue Worker Design 15](#_Toc528235524)

[3.6.4 Immediate SMS Queue Worker Design 18](#_Toc528235525)

[3.6.5 Core Sync Queue Worker Design 21](#_Toc528235526)

[*3.6.6 Einstein Language Functionality 21*](#_Toc528235526)

3.6.7 ProcessSMSForDigital Functionality

[3.7 Job Priority 25](#_Toc528235527)

[3.8 Application Monitoring 25](#_Toc528235528)

[3.8.1 Monitoring Queue Activity 25](#_Toc528235529)

[3.8.2 Librato Configuration and queue monitoring 26](#_Toc528235530)

[3.9 Load Balancing 26](#_Toc528235531)

[3.9.1 Current bandwidth in jobs/sec allocated for different time zones 26](#_Toc528235532)

[3.9.2 Load balancing by adjusting speed on time zone queues at specific time intervals 27](#_Toc528235533)

[4. Contacts 30](#_Toc528235534)

# Introduction

The scope of the Message delivery system encompasses the connection points from Salesforce Core to Heroku and then back to Salesforce Core after the message has been delivered to the customer via LiveMessage. This document will include the postgres database design, Heroku connect settings and any and all application code required within the Message Delivery system.

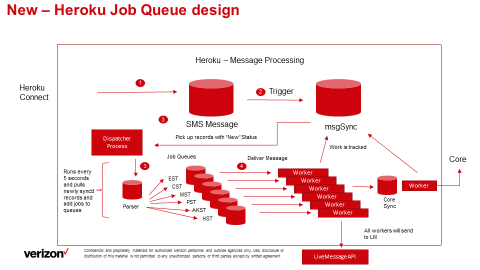
# Application And Design

## High-Level Functionality & Assumptions

The following assumptions have been identified throughout the design phase of this application

* The Heroku platform will be utilized as the platform to host the application due to scalability limitations within the Salesforce Core platform.
* Node.js will be utilized to instantiate the logic required to deliver the message
* Heroku Connect cannot be used to sync back to Salesforce Core in a near real-time requirement setting.
* Heroku Connect will be utilized to synchronize between Salesforce Core and the Heroku database called Postgres

## High-Level Technical Design



The above design shows that all messages that will be sent out to a customer/employee go thru a 3 step process. Process one is the parsing (taking dynamic content and formulating the real message), assigning the message to a time zone based Job queue, and finally to the Core sync job queue where the results will be updated back on the Salesforce Core platform.

## Heroku Add-ons required in solution and their intented functions

## The below add-ons have been identified by the Salesforce Heroku architects.

|  |  |  |
| --- | --- | --- |
| **Add-On Name** | **Function** | **Licensed Plan** |
| Heroku Postgres | Database store that is hosted within the private space in Heroku. Data is encrypted at rest | Private-4 |
| Heroku Connect | Product build by Heroku that is used to synchronize data between Salesforce Core and Postgres | Production |
| Redis | A NoSQL database that is being used to store messages within a job queue | Private-10 |
| PaperTrail | Syslog (console functionality) that is used to capture system messages from within the add-ons and dyno’s | Bekant |
| Librarto | System monitoring add-on | Nickel |
| Einstein Vision and Language | Add on for sentiment and intent analysis of customer responses | Starter |

## Javascript Libraries used within the system

|  |  |
| --- | --- |
| **Library** | **Function** |
| Bull | Bull is an open source job queuing and job management framework |
| Moment-Timezone | JS framework that is used to help calculate timezone differences within the JS code |
| Redis | JS framework that is required by bull to work with redis |
| Pg | JS framework that is required to work with Heroku Postgres |
| Pg-Pool | Js framework that provides connection pooling to Heroku Postgres |
| Logfmt | JS framework that generates logs in specific format so librarto can ipck it up and create charts |
| JSforce | JavaScript Library that utilizes Salesforce's API making asynchronous calls. |

## Heroku data models and data dictionary

### Postgres tables:

The table below is a list of tables that are utilized within the Message Delivery system.

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| SalesforceProd.ccpm\_SMSMessage\_\_c | SMS Table that is synchronized between Salesforce Core and postgres |
| SalesforceProd.ccpm\_Store\_\_c | Store Table that is synchronized between Salesforce Core and postgres |
| Msgsync | The authoritative source of truth for all messages and their current states within the delivery system. |
| Storesync | The table recording all the stores to be un-suppressed. |
| SalesforceProd.Businesshours | Hours of operation by time zone as defined in Salesforce Core |
| SalesforceProd. ccpm\_inboundCustomerResponse\_\_c | Customer response record synced from Salesforce |
| SalesforceProd. CCPM\_priorityDetermination\_\_c | Priority Determination table synced from Salesforce |
| Icrsync | Postgres table that keeps a track of all inbound customer responses and their status before the automated response is sent out from Heroku |

#### SalesforceProd.Business Hours definition

|  |  |
| --- | --- |
| **Field Name** | **Data Type** |
| Name | Text (ex. Est, mst,pst) |
| Sundaystarttime | Time without time zone |
| Sundayendtime | Time without time zone |
| Mondaystarttime | Time without time zone |
| Mondayendtime | Time without time zone |
| Tuesdaystarttime | Time without time zone |
| Tuesdayendtime | Time without time zone |
| Wednesdaystarttime | Time without time zone |
| Wednesdayendtime | Time without time zone |
| Thursdaystarttime | Time without time zone |
| Thursdayendtime | Time without time zone |
| Fridaystarttime | Time without time zone |
| Fridayendtime | Time without time zone |
| Saturdaystarttime | Time without time zone |
| Saturdayendtime | Time without time zone |

This definition is from the standard business hours feature on Salesforce core. The records are maintained on core and synced oneway for Heroku to pickup.

#### msgsync data definition

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Description** | **Constraints** |
| Sfid | Varchar(18) | Unique Salesforce record number. Values are always 18 characters | Unique Index |
| lm\_responsemsg | Varchar(100) | Response message returned form LM after an api call | N/A |
| Lm\_responsecode | Varchar(5) | HTTP code that is returned from the LM api | N/A |
| Lm\_requestId | Varchar(255) | Unique Id that LM returns once you get a successful api call | N/A |
| Lm\_smsfailcounter | Integer | If api call’s are failing we are counting out how many times | N/A |
| Smssentactualtime | Timestamp | Date/Time value for when the API call to LM was executed | N/A |
| Msgbody | Varchar(500) | Actually message that will be sent to customer | N/A |
| Msgstatus | Varchar(50) | The message status tracks where in the set of queues the message is | Valid Value = NEW, Queued for Parsing, Queued for Sending,  Sending, Sent, Synced |
| Scheduleddatetime | Timestamp | Scheduled datetime of the message | N/A |
| Msgtimezone | Varchar(5) | Time zone that the message is intended for | Valid values (EST, CST, MST, PST, AKST, HST) |
| Lm\_livetextnumber | Varchar(20) | Number to text from in LM | Phone number format |
| Msgtonumber | Varchar(20) | Customer number | Phone number format |
| Msgtype | Varchar(100) | The sms message type in core | N/A |
| Leadrecord | Varchar(20) | The lead record connected to the message | N/A |
| Errordescription | Varchar(255) | Error description in Heroku during the message process | N/A |
| Retrycounter | Integer | Number of times failed messages have retried | N/A |
| Leadtype | Varchar(255) | Lead type of the lead record to implement priority queues | N/A |
| store | Varchar(18) | Store linked to the message and lead | N/A |
| smssuppressedtime | Timestamp | Time when the message got suppressed | N/A |
| accountnumber | Varchar(20) | Account number related to msg record | N/A |
| Icrsfid | Varchar(18) | Sfid of the ICR object synced from SF(ccpm\_inboundCustomerResponse\_\_c) | N/A |

#### SalesforceProd.ccpm\_smsmessage\_\_c definition

The table fields are defined by the mapping that is completed within the Heroku Connect application. There has been one small change on this object though and that is a the addition of a postgres trigger

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ccpm\_messagetype\_\_c | Varchar(255) |  |
| ccpm\_tophonenumber\_\_c | Varchar(40) | Phone number that will receive the SMS |
| ccpm\_formattedtophone\_\_c | Varchar(1300) |  |
| ccpm\_messagestatus\_\_c | Varchar(255) | (Queued, Error,Resend Failed, Sent, Removed) |
| ccpm\_internalcommunication\_\_c | boolean |  |
| ccpm\_sentto\_\_c | Varchar(18) | What Employee user is the message being sent to |
| Ccpm\_dynamicecode\_\_c | Varchar(255) |  |
| Ownerid | Varchar(18) | Owner of the record in core |
| ccpm\_scheduledatetime\_\_c | Timestamp without time zone | Date/time that message has been scheduled for delivery |
| ccpm\_dynamicstorerecommendedhrs\_\_c | Varchar(255) | Recommended hours of operation |
| ccpm\_smstimezone\_\_c | Varchar(1300) | Time zone of the store that the lead is associate to |
| ccpm\_leadrecord\_\_c | Varchar(18) | Lead that is related to this SMS message record |
| ccpm\_message\_\_c | Varchar(500) | Templated version of scheduled message |
| ccpm\_dynamicleadownername\_\_c | Varchar(255) | Name of lead owner from core |
| ccpm\_livetextnumber\_\_c | Varchar(255) | Live Message (from number). Either short code or 800 long code |
| Sfid | Varchar(18) | Sf unique id for each record |
| ccpm\_leadtype\_\_c | Varchar(50) | Lead type of the message |
| Ccpm\_store\_\_c | Varchar(18) | Sf id for the store lookup |
| CCPM\_accountNumber\_\_c | Varchar(20) | Account number related to message record |

The trigger hc\_trigger has been added to the SalesforceProd.CCPM\_SMSMessage\_\_c. The sole purpose of this trigger is to start the flow of required processing for each message (parsing, sending, sent, synced)

\* Please note that if Heroku connect is refreshed or uninstalled and reinstalled this custom trigger will be removed.

#### SalesforceProd.ccpm\_store\_\_c definition

The table fields are defined by the mapping that is completed within the Heroku Connect application.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ccpm\_issuppressed\_\_c | boolean | Whether or not the store is suppressed. |
| ccpm\_leadtypesforsmsremoval\_\_c | Varchar(4099) | Lead types to be removed if the store is suppressed |
| ccpm\_leadtypesforsmssuppression\_\_c | Varchar(4099) | Lead types to be suppressed if the store is suppressed |
| Sfid | Varchar(18) | Sf unique id for each store |

#### Storesync data definition

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| Id | Serial Primary Key | Id of the store unsuppression record |
| Storeid | Varchar(18) | The sfid of the store |
| storestatus | Varchar(50) | Status of this unsuppression. Possible values: ‘New’, ‘Queued’, ‘Unsuppressed’, ‘Error’ |
| Lastmodifiedtime | Timestamp | Tracking last modified time |

#### SalesforceProd. ccpm\_inboundCustomerResponse\_\_c data definition

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| CCPM\_ITR\_\_c | Varchar(18) | Id of the inbound customer response record |
| CCPM\_customerResponse\_\_c | Varchar(700) | Customer response text |
| CCPM\_intentModelId\_\_c | Varchar(255) | The id used to identify the einstein model used to train intent |
| CCPM\_intentProbability\_\_c | Varchar(255) | Probability value(between 0 and 1) for the customer’s intent as predicted by einstein |
| CCPM\_intent\_\_c | Picklist | The customer’s intent predicted from their response |
| CCPM\_isCustomer\_\_c | Boolean | Flag to indicate if the response is from a customer or prospect |
| CCPM\_lead\_\_c | Varchar(18) | Id of the lead associated with the ICR record |
| CCPM\_liveTextNumber\_\_c | Varchar(40) | The number from which the auto response message will be sent out |
| CCPM\_messageType\_\_c | Picklist | Message type will be ‘Autoresponse’ |
| CCPM\_priority\_\_c | Picklist | The priority estimated for the customer response based on intent and sentiment |
| CCPM\_sentimentModelId\_\_c | Varchar(255) | The id used to identify the einstein model used to train sentiment |
| CCPM\_sentimentProbability\_\_c | Varchar(255) | Probability value(between 0 and 1) for the customer’s sentiment as predicted by einstein |
| CCPM\_sentiment\_\_c | Picklist | The customer’s sentiment predicted from their response |
| CCPM\_toPhoneNumber\_\_c | Varchar(40) | The customer’s phone number |
| Id | Id | Unique id for the postgres table |

#### 3.5.1.7 SalesforceProd. ccpm\_priorityDetermination\_\_c data definition

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| CCPM\_customerMessage\_\_c | Varchar(1000) | Response message to be sent to customer |
| CCPM\_intent\_\_c | Varchar(255) | Intent as estimated by einstein |
| CCPM\_priority\_\_c | Varchar(255) | Priority calculated for given intent and sentiment |
| CCPM\_prospectMessage\_\_c | Varchar(1000) | Response message to be sent to prospect |
| CCPM\_sentiment\_\_c | Varchar(255) | Sentiment as estimated by einstein |
| Id | Varchar(18) | Unique id for table |
| Name | Varchar(80) | Name |

#### 3.5.1.8 icrSync data definition

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| sfid | varchar(18) | Inbound customer response record Id from Salesforce |
| customerresponse | varchar (700) | Response sent by the customer |
| itr | varchar (20) | ITR |
| sentiment | varchar (1000) | Sentiment predicted from einstein for the customer response |
| intent | varchar (1000) | Intent predicted from einstein for the customer response |
| intentmodelid | varchar (255) | The id used to identify the einstein model used to train intent |
| sentimentmodelid | varchar (255) | The id used to identify the einstein model used to train sentiment |
| senitimentprob | varchar (255) | Probability value(between 0 and 1) for the customer’s sentiment as predicted by einstein |
| Intentprob | varchar(255) | Probability value(between 0 and 1) for the customer’s intent as predicted by einstein |
| priority | varchar (255) | The priority estimated for the customer response based on intent and sentiment |
| lead | varchar (18) | Id of the lead associated with the ICR record |
| livetextnumber | varchar (20) | The number from which the auto response message will be sent out |
| tophonenumber | varchar (20) | The customer’s phone number |
| messagetype | varchar (255) | Message type will be ‘Autoresponse’ |
| status | varchar (255) | The status of the ICR message in Heroku. This changes from ‘new’ to ‘ready for callouts’ as we move down the pipeline |
| iscustomer | boolean | Flag to check if the response is to sent to a customer or prospect |

#### 3.5.1.9 digitalEcodePersistence data definition

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| sfid | varchar(18) | Salesforce Id of the message record synced to msgsync |
| accountnumber | varchar(20) | Account number related to the msg record |
| mtn | varchar(18) | Customer number to which message will be sent |
| ecode | varchar(18) | Ecode that will be sent out as part of the mylink message |
| repecode | boolean | True or False value that specifics whether the ecode is a store code or rep code |
| sentdatetime | timestamp | Timestamp of the message send time |
| status | varchar(50) | Status of the record in the persisting ecode table |
| errordescription | varchar(255) | Logs the error message during the pipeline process |
| retrycounter | integer | Counter that determines how many times a callout attempt has been made to dotcom end point |

## Job Queue and Job processing logic

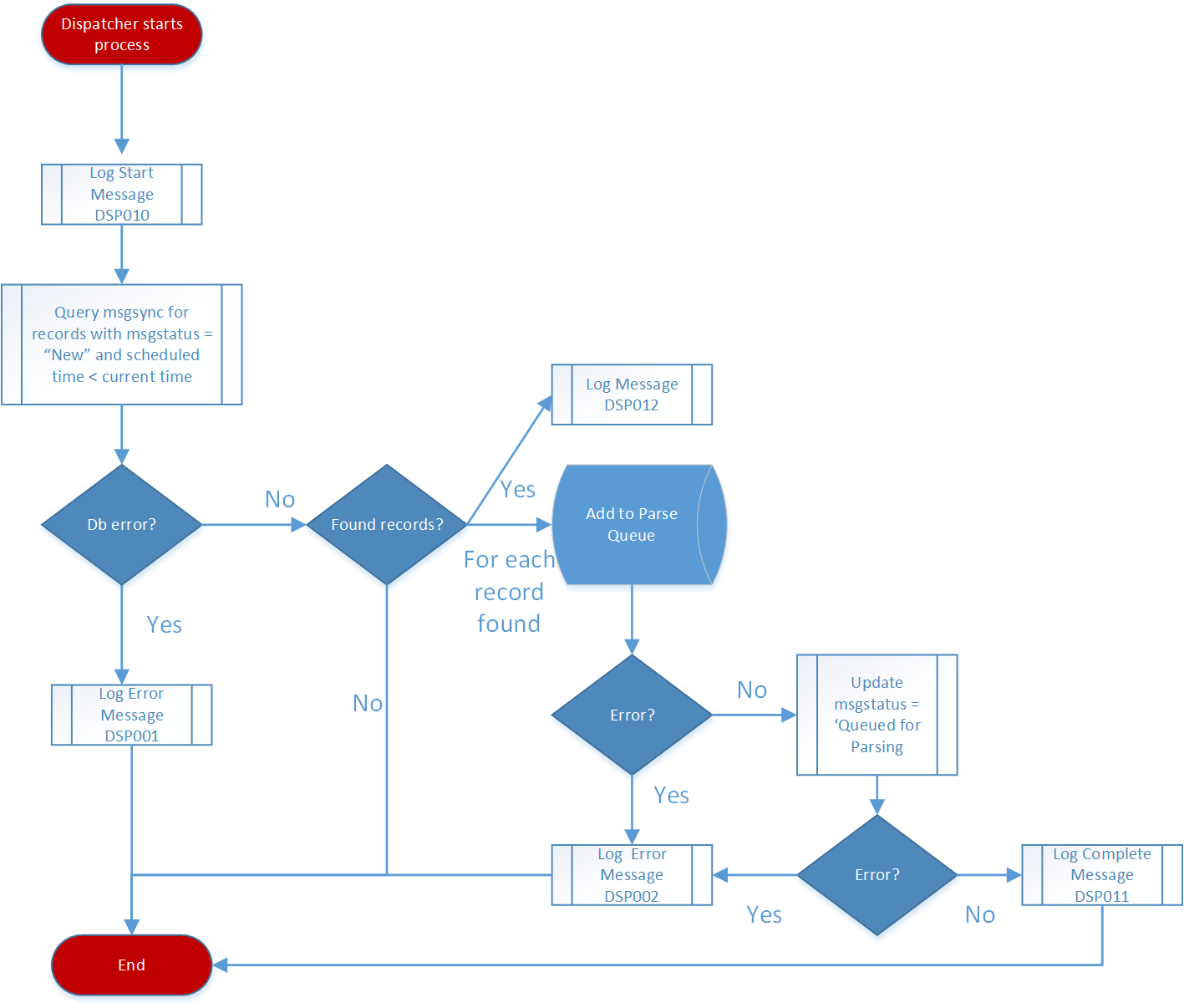
A series of jobs and functionality will be utilized to process messages that are synced to Postgres via Heroku Connect. Once the data has been brought onto the Heroku platform and stored within Postgres a series of queues will be utilized. The process will start with parsing the message (think of merge field substitution), then scheduled based on a specific time zone that message is designated for and to the last queue that then updates data back within the Salesforce Core Platform. The complete pipeline is required for each and every message that needs to be sent to a customer MTN (Phone number).

### Dispatcher Functionality

The first process that runs will be a dispatcher. This dispatcher will need to utilize the “Clock” Process design to run at an interval as it will be utilized to start/stop and control a series of job and operational needs of the whole pipeline.

The first responsibility of the dispatcher will be to run every 30 seconds and will read the Msgsync table to find work that has a MsgStatus = “New” and where the scheduled date/time has elapsed.

Once the messages have been designated as “needing to be parsed” the job will iterate over each new message that has been received and add to the job queue called “Parse Queue”. After the message has been added to the Parse queue the msgsync record should have the msgstatus updated to “Queued for Parsing”.See the below diagram to review overall flow of the dispatcher.



#### Exception Handling

If you receive an error while retrieving the records with a “New” status, log out to the console the message Error code DSP001.

If you receive an error while adding the message to the job queue, or updating the table for msgsync, log out to the console the error code d

#### Dispatcher Logging

The following additional logging needs to be included in the dispatcher. For example message to log review the section on Logging Codes

1. Initialization of the job log the started message DSP010.
2. Completion of dispatcher log the completed message DSP011.
3. Once messages have been Retrieving messages that need to be parsed log out message DSP012

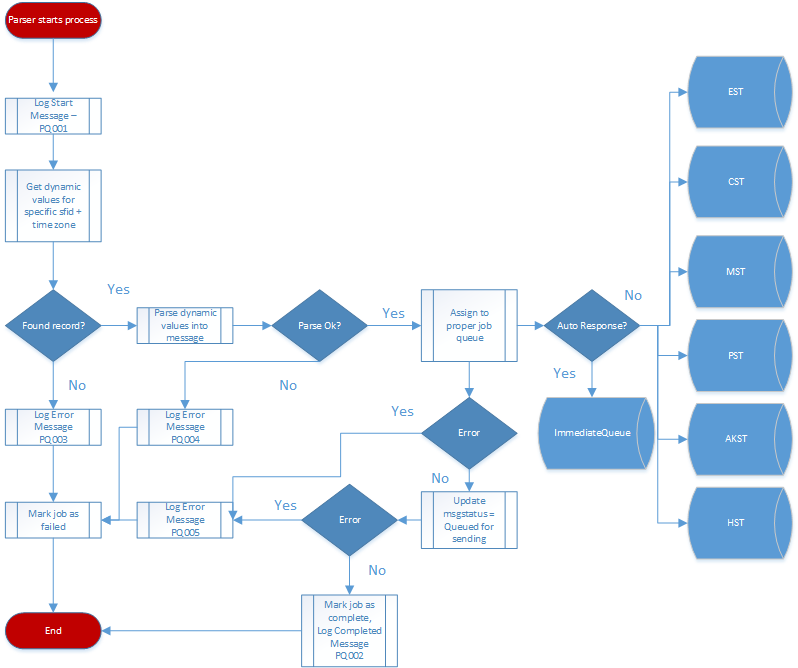
**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| DSP001 | Failed to retrieve “New” Messages from MsgSync | Critical |
| DSP002 | Failed to update MsgSync from “New” to “Queued for Parsing” or add job to parse queue | Critical |
| DSP010 | Dispatcher tick | Info |
| DSP011 | Dispatcher completed -> <SFID> | Info |
| DSP012 | Found messages in NEW status -> <SFID> | Info |

### Parse Queue Worker Design

The existing parser functionality will be moved from BDParser and then updated to include new logging/exception handling functionality.

Please see the high level flow of the Parse processor



#### Parse Queue Functionality

The parse queue work will be used to take the dynamic portion of the message and will substitute the merge fields with actual values. Below is the order of processing that is required for this work:

1. With the SFID passed in with the job, query the salesforceprod.ccpm\_smsmessages\_\_c for the msgbody, supported merge fields, and time zone designated for the message
2. Process thru the message looking for the supported merge fields and complete the substitution
3. Identify and add this message to the appropriate time zone job queue or the immediateSMSQueue
4. Update the msgsync record with parsed message and status to “Queued for Sending”

#### Exception Handling

A wide variety of error conditions can be encountered while parsing messages. Here is a list and how to handle them.

1. If querying salesforceprod.ccpm\_smsmessage\_\_c by sfid results in 0 record, log PQ003 and fail the job
2. If a failure takes place while parsing the message itself, fail the job with a PQ004 message
3. If a failure takes place while adding the job to the time zone based queue or update msgstatus in msgsync to ‘Queued for Sending’, mark the job as failed and log the message PQ005.

#### General Parse Queue Logging

The following additional logging needs to be included in the parse queue worker.

1. When each job starts up log the message PQ001
2. When each job completes log the message PQ002

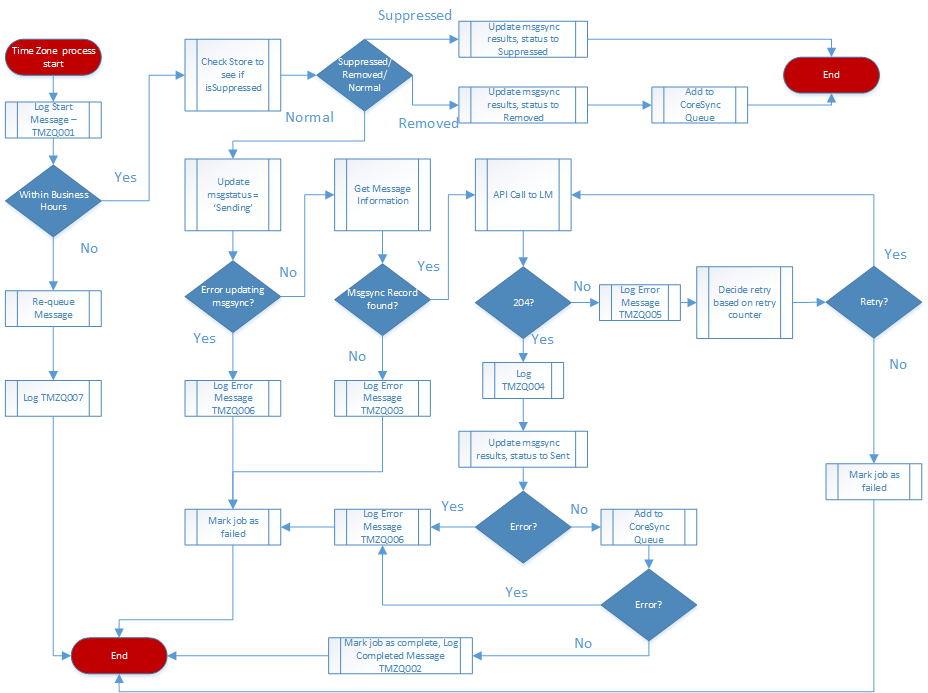
**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| PQ001 | Start parsing the SMS -> <SFID> | Info |
| PQ002 | Parsing completed -> <SFID> | Info |
| PQ003 | Failure in querying records from Heroku Connect table | Critical |
| PQ004 | Unhandled Parse Error -> <SFID> | Critical |
| PQ005 | Failure in updating records in MsgSync to Queued for Sending/ adding to <time zone queue> -> <SFID> | Critical |

### Time Zone Queue Worker Design

#### Time Zone Queue worker Functionality

Each time zone worker processes all messages that are within the given time zone. Any messages scheduled outside of business hours will be re-queued back into the queue.



#### Rate limiting

Every time zone queue needs to have a configurable value for how many jobs/second the queue will process. In general, all of the time zone queues must add up to < 50 jobs / second which is the max number of api calls that LiveMessage can currently handle. The rate limiting in practice will end up being skewed to EST and CST time zones as currently the daily volume is much larger in those two time zones. The current msgs / second are identified below:

|  |  |
| --- | --- |
| **Time Zone** | **Job Rate / Second** |
| EST | 10 |
| CST | 4 |
| MST | 2 |
| PST | 2 |
| AKST | 1 |
| HAST | 1 |

After a time zone has completed the daily work that time zone will simply remain idle for the remaining period.

\* Please note that the number of Jobs / Second is lower than the overall 50 msgs / second that Live Message gives us but a lot of these jobs end up becoming multiple messages that get delivered to Live Message.

For example see the message types currently in the system and the actual number of Live Message API calls are generated

|  |  |
| --- | --- |
| **Message Type** | **Number of API calls** |
| Abandoned Cart | 2 |
| Abandoned Visit | 2 |
| Campaign Messages | 2 |
| Customer Quote | 2 |
| Prospect | 2 |
| Follow Up messages | 1 |
| Employee Messages | 1 |

#### Business Hours

For each time zone the logic will be built where each job queue will only process jobs within business hours. This means that the queue’s will never start/stop/pause. They will be active all the time and jobs picked up outside of the business hours will be put back into the queue.

#### Callout Retries

For each failed callout to LM for each message job, the current retry counter is increased by one. Once it reaches the maximum count, no more retries will be done. The job will log error status in msgsync.

#### Suppression Framework

Before sending each message, check the store object based on the store field lookup on the message. If the Boolean field ccpm\_issuppressed\_\_c is true, check the following lists: ccpm\_leadtypesforsmsremoval\_\_c and ccpm\_leadtypesforsmssuppression\_\_c. If the current message lead type is found to match any in the removal list, mark the msgsync status to be Removed and push to coresync queue. If the current message lead type is found to match any in the suppression list, mark the msgsync status to be Suppressed, and update the current time as the smssuppressedtime.

#### Exception Handling and General Logging

The following additional logging needs to be included in the time zone queue worker.

1. When each job starts up, log the message TMZQ001
2. When each job completes, log the message TMZQ002
3. If query for the msgsync record by sfid returns 0 records, mark the job as failed and log TMZQ003.
4. If calling the Live Message API call returns a 204 response, log the response TMZQ004
5. If calling the Live Message API call returns a non 204 response, mark the job failed and log a TMZQ005 message
6. If an error is received while updating the msgstatus on msgsync or adding the job to the Core sync queue, log the response TMZQ006
7. If message is re-queued due to being out of business hours, log the message TMZQ007

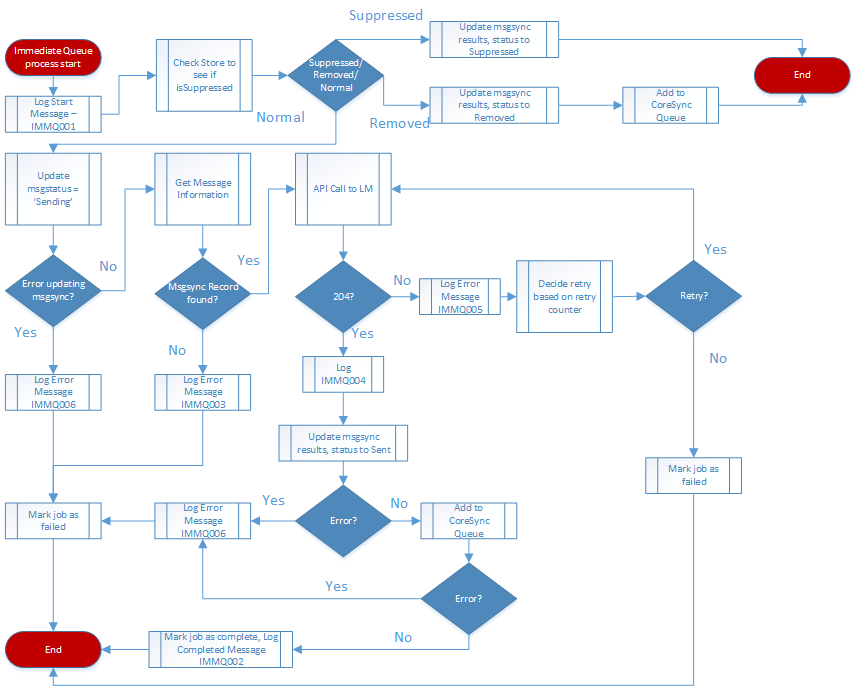
**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| TMZQ001 | Received message in <timezone> -> <SFID> | Info |
| TMZQ002 | Job completed -> <SFID> | Info |
| TMZQ003 | Msgsync record not found -> <SFID> | Critical |
| TMZQ004 | API call to LiveMessage returns Status Code -> 204 RequestId -> <RequestId> -> ${smsId} | Info |
| TMZQ005 | API call to LiveMessage returns non-204 Status Code -> <Status Code> -> ${smsId} | Critical |
| TMZQ006 | Msgsync failed to set message to 'Sending'/’Sent’ or add job to Core Sync Queue -> <sfid> | Critical |
| TMZQ007 | Message Requeued due to out of hours -> <sfid> | Info |

### Immediate SMS Queue Worker Design

#### Immediate SMS Queue worker Functionality

Immediate SMS Queue sends messages out immediate without the limit of timezones and business hours.



#### Rate limiting

The rate for ImmediateSMSQueue is 2 jobs / second

#### Callout Retries

For each failed callout to LM for each message job, the current retry counter is increased by one. Once it reaches the maximum count, no more retries will be done. The job will log error status in msgsync.

#### Suppression Framework

Before sending each message, check the store object based on the store field lookup on the message. If the Boolean field ccpm\_issuppressed\_\_c is true, check ccpm\_leadtypesforsmsremoval\_\_c and ccpm\_leadtypesforsmssuppression\_\_c. If the current message lead type is found to match any in the removal list, mark the msgsync status to be Removed and push to coresync queue. If the current message lead type is found to match any in the suppression list, mark the msgsync status to be Suppressed, and update the current time as the smssuppressedtime. When the store gets un-suppressed, it will fire a trigger to requeue this message and other messages with the status Suppressed and related to the store.

#### Exception Handling and General Logging

The following additional logging needs to be included in the immediate sms queue worker.

1. When each job starts up, log the message IMMQ001
2. When each job completes, log the message IMMQ 002
3. If query for the msgsync record by sfid returns 0 records, mark the job as failed and log IMMQ 003.
4. If calling the Live Message API call returns a 204 response, log the response IMMQ 004
5. If calling the Live Message API call returns a non 204 response, mark the job failed and log a IMMQ 005 message
6. If an error is received while updating the msgstatus on msgsync or adding the job to the Core sync queue, log the response IMMQ006

**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| IMMQ001 | Received message in immediateSMSQueue -> <SFID> | Info |
| IMMQ002 | Job completed -> <SFID> | Info |
| IMMQ003 | Msgsync record not found -> <SFID> | Critical |
| IMMQ004 | API call to LiveMessage returns Status Code -> 204 RequestId -> <RequestId> -> ${smsId} | Info |
| IMMQ005 | API call to LiveMessage returns non-204 Status Code -> <Status Code> -> ${smsId} | Critical |
| IMMQ006 | Msgsync failed to set message to 'Sending'/’Sent’ or add job to Core Sync Queue -> <sfid> | Critical |

### Core Sync Queue Worker Design

#### Core Sync Queue Functionality

The core sync queue’s job is to take sent messages from timezonequeue and update the results back to the Lead object, create tasks, and update the LM results to the CCPM\_SMSMessage\_\_c object. These are done through a single JSON request to a Salesforce Apex Web Service.



#### Lead Update

The following overall process needs to be followed when updating core.

1. Use the sfid from job queue and query the msgsync table to obtain the following values:
   1. LeadId
   2. sentActualTime
   3. Msgbody
   4. messageType
   5. MsgStatus
   6. lmRequestId
   7. lmResponseMsg
   8. lmResponseCode
2. Use the LeadId from part 1 to query the core lead table to obtain the following values:
   1. Status
   2. OwnerId
3. With the data obtained, populate the following lead fields in JSON format.

The following fields need the following update:

|  |  |
| --- | --- |
| **Conditions** | **Core Lead field** |
| If msg.messageType is ’InitialSMSPart1’ or ‘CustomerQuoteSMSPart1’ or ‘CampaignInitialSMSPart1’ or ‘AbandonedVisitInitialSMSPartI’  'RetailCreditProspectInitialSMS',  'DigitalACProspectInitialSMS',  'DigitalBrowsingSMS1',  'MyOffer',  'Customer 5G Home SMS' | CCPM\_isInitialSMSSent\_\_c = true |
| CCPM\_initialSMSSentDateTime\_\_c = sentActualTime |
| If msg.messageType is ‘NewProspectSMS1’ | CCPM\_isProspectSMSSent\_\_c = true |
| CCPM\_prospectSMSSentDate\_\_c = sentActualTime |
| If msg.messageType is ‘Followup’ or ‘CampaignFollowup’ 'mccampaignOnePart','RetailCreditProspectFollowupSMS','DigitalACProspectFollowupSMS' | CCPM\_isFollowUpSMSSent\_\_c = true |
| CCPM\_followUpSMSDate\_\_c = sentActualTime |
| If messageType is ‘ContactCardSMS’ or ‘POSBusinessCard’ | CCPM\_lastInteraction\_\_c=sentActualTime |
| If msg.messageType is ‘ContactCardSMS’, lead.status is ‘Responded’ and lead.CCPM\_leadSubStatus\_\_c is ‘Action Required’ | status= ‘Qualified’ and CCPM\_leadSubStatus\_\_c = ‘In Discussion’ |
| If msg.messageType is not ‘ContactCardSMS’ and lead.status is ‘Eligible’ | status= ‘Contacted’ |

#### Task Creation

In the previous section the data required to create the tasks were retrieved in step 1 & 2. Use those fields to populate the task fields.

|  |  |
| --- | --- |
| **Conditions** | **Core Task field** |
| If msg.messageType is ‘InitialSMSPart1’ or ‘CustomerQuoteSMSPart1’ or ‘CampaignInitialSMSPart1’ or ‘AbandonedVisitInitialSMSPartI’ or ‘NewProspectSMS1’ | Need to create two tasks, both having all the other identical fields but the CCPM\_SMSMessageType\_\_c field.(Will be ‘InitialSMSPart2’ or ‘CustomerQuoteSMSPart2’ or ‘CampaignInitialSMSPart2’ or ‘AbandonedVisitInitialSMSPartII’ or ‘NewProspectSMS2’ respectively) |
| If msg.messageType is anything else | CCPM\_SMSMessageType\_\_c = msg. messageType |
| N/A | Description = msg.msgBody |
| CCPM\_communicationDirection\_\_c = ‘Outbound’ |
| Priority = ‘Normal’ |
| Status = ‘Completed’ |
| Subject = ‘Text Message Sent’ |
| WhoId = lead.Id |
| OwnerId = lead.ownerId |

#### CCPM\_SMSMessage\_\_c Updates

In the lead update section the data required to update the CCPM\_SMSMessage\_\_c were retrieved. Use those fields to populate the SMS fields.

|  |  |
| --- | --- |
| **MsgSync FIeld** | **Core CCPM\_smsmessage\_\_c field** |
| lm\_responsemsg | ccpm\_responsemessage\_\_c |
| lm\_requestid | ccpm\_requestid\_\_c |
| lm\_responsecode | ccpm\_responsecode\_\_c |
| sentActualTime | CCPM\_smssentactualtime\_\_c |
| messageType | CCPM\_messageType\_\_c |
| status | CCPM\_messagestatus\_\_c |
| Msgbody | CCPM\_message\_\_c |

#### Auto retries

When encountering errors in any part of the core sync process, the module captures the error message and update the msgsync table accordingly. Then it will attempt to auto retry up to a retry counter.

1. If an error is received during querying for lead information or constructing the lead JSON fields, log CS003 and update the msg status field = “Error Lead Update” and put the error message in the error description field. Then retry the job starting from the beginning if the retrycounter < CORESYNC\_RETRY\_COUNTER.
2. If an error is received during constructing the task JSON fields, log CS005 and update the msg status field = “Error Activity Insert” and put the error message in the error description field. Then retry the job starting from constructing the task JSON fields if the retrycounter < CORESYNC\_RETRY\_COUNTER.
3. If an error is received during constructing the SMS JSON fields, log CS006 and update the msg status field = “Error SMS Update” and put the error message in the error description field. Then retry the job starting from constructing the SMS JSON fields if the retrycounter < CORESYNC\_RETRY\_COUNTER.
4. If retrycounter > CORESYNC\_RETRY\_COUNTER, log CS007 and fail the job.

#### General Core Sync Queue Logging

The following additional logging needs to be included in the parse queue worker.

1. When each job starts up log the message CS001
2. When each job completes log the message CS002

**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| CS001 | Job started – <SFID> | Info |
| CS002 | Job completed - <SFID> | Info |
| CS003 | Webservice callout failure/ Lead Update failure <error message> | Critical |
| CS004 | MsgSync update failure - <sfid> | Critical |
| CS005 | Task Creation failure , <error message> | Critical |
| CS006 | Ccpm\_\_smsmessage\_\_c failure, <error message> | Critical |
| CS007 | Job failed <sfid> | Critical |
| CS008 | MsgSync not found <sfid> | Critical |
| CS009 | Salesforce Login Error | Critical |

### Einstein Language Functionality

The Einstein language functionality syncs data from Inbound customer responses, sends it to the Einstein endpoint for intent and sentiment analysis, determines the priority and generates an automated response. Once the automated response is created, this message is added to the msgsync table and then is picked up by the dispatcher(section 3.6.1 in this design document) and then eventually sent out and synced back to Salesforce

#### ICR Dispatcher Module

The first responsibility of the dispatcher will be to run every 30 seconds and will read the icrsync table to find the records that has a icrStatus = “New”. These records will be updated and their status will be set as “Ready for callout” and then the sfid will be added to the job queue called “Dispatcher Queue”.

#### EL Callout Module

This module processes the jobs from the “Dispatcher Queue”. Using the sfid of the ICR object retrieved from the job, the ICR record is fetched from the icrsync table.

The “makeIntentCallout” function is called next. To make the Einstein callout, an access token is required. If the access token is empty, then “getAccessToken” is called to get this token. Once the access token is retrieved, the callout is made to get the “Intent”.

**Intent Callout responses**

* If the callout was successful then the response returned from this callout is as follows



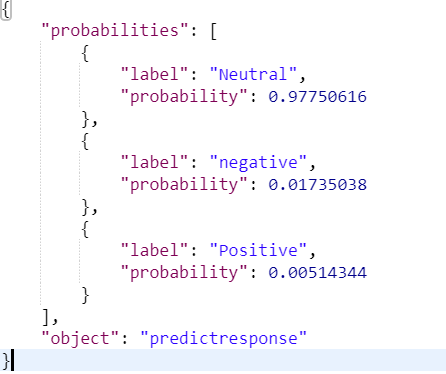
The first label is the “intent” value and the probability represents “intent probability”. These values are updated to the icrsync table in the “updateIntent” function

* If the callout was a failure and it returns a “401” error, then the access token has expired. So, the access token is fetched again and then the intent callout is made again
* If the error status is something else, then an error message is logged and the icrstatus is updated to “Intent callout failed”

If intent callout was successful then “makeSentimentCallout” function is called next to get the “sentiment”

**Sentiment Callout responses**

* If the callout was successful then the response returned from this callout is as follows



The first label is the “sentiment” value and the probability represents “sentiment probability”. These values are updated to the icrsync table in the “updateSentiment” function

* If the callout was a failure and it returns a “401” error, then the access token has expired. So, the access token is fetched again and then the sentiment callout is made again
* If the error status is something else, then an error message is logged and the icrstatus is updated to “Sentiment callout failed”

If sentiment callout was successful then the icrStatus is upated to “EL Callout Successful” and then the icr sfid is added as a job to the “ELCallout Queue”

#### Priority Determination Module

The job is processed from the “ELCallout Queue” and then the icr record is retrieved from icrSync table. The “intent” and “sentiment” from the icr object is used to retrieve the “priority” , “customer message” and “prospect message” from the “priority determination” table. This “priority” is updated to icrSync table and the icrStatus is set as “Autoresponse process complete”.

Depending on whether the message is to be sent to a customer or a prospect, the appropriate message is chosen and a new record is created in msgSync table with the msgstatus as ‘NEW’. This new record will be picked up by the dispatcher module explained in section 3.6.1 in this design document. From there, the standard procedure is followed. The message goes through the dispatcher, then to parser, then to immediate queue(because the message is an automated response) and then back to coresync.

#### Code changes to dispatcher, parser, immediate and coresync modules

The automated response created in section 3.6.6.3 will not have a Salesforce Id as it will be created inside the Heroku code. So, we create a new field called icrsfid on msgsync table to keep track of these records. The sfid will be null for these ICR messages.

All the records in dispatcher, parser, immediate and coresync modules were previously retrieved using “smsId” only. Now we will be modifying these queries so that the record is retrieved based on “smsId” or the “icrSfid”. As soon as the record is retrieved, we check to see if the record is a normal message record(with a valid smsId) or if it is a “ICR” message.

* If the message is a normal message, then the standard procedure explained in sections 3.6.1 to 3.6.5 is followed.
* If the message is ICR message, then new functions are added to handle these messages. The functionality will work similar to the normal flow.
  + The dispatcher module adds the icr Id to the dispatcher queue
  + The parser has no work to do as the ICR message is already parsed. So it adds the icr Id to immediate queue
  + The message gets sent out in immediate queue using the existing callout and is then added to the coresync queue
  + In the coresync queue, we make a new callout to the endpoint ‘/api/v1/einsteinlanguage/setActions/’. The request body is created in “buildRequestBodyForICR” function and then the callout is made. If the response is successful the msgstatus is converted to “Synced” as usual

### ProcessSMSForDigital Functionality

If the message contains a MyLink with an Ecode(this code is needed to give commissions to store reps based on the number of customers they successfully reachout to) then a record entry is created in the “digitalEcodePersistence” table in the Parser Queue(section 3.6.2).

In the coreSync module(section 3.6.5), after the callout to SF is successful:

* If the environment is Prod, , the message is added to the “ecodePersistence Queue”
* If the environment is not Prod, then the message is not added to the queue and the flow ends in the CoreSync module

The ProcessSMSForDigital module picks up jobs from the “ecodePersistence Queue”, builds a request JSON and then makes a callout to the DOTCOM endpoint 'https://api-secure.verizon.com/vz/mylink/persistecode'. The purpose of this callout is to send the Ecode and smsId data to Verizon so that they can track the leads generated with MyLinks. On success response, the status in the “digitalEcodePersistence” table is updated to “Sent”, else the status is updated to “Error”.

If the environment is not prod, no jobs will be added to “ecodePersistence Queue” and hence the ProcessSMSForDigital dyno can be turned off. A new config variable “MAKE\_DIGITAL\_CALLOUT” is created in the Heroku prod app and its value is set as “True”. This variable is used in an IF condition to determine if the jobs should be added to the queue or not.

## Store/SMS Unsuppression

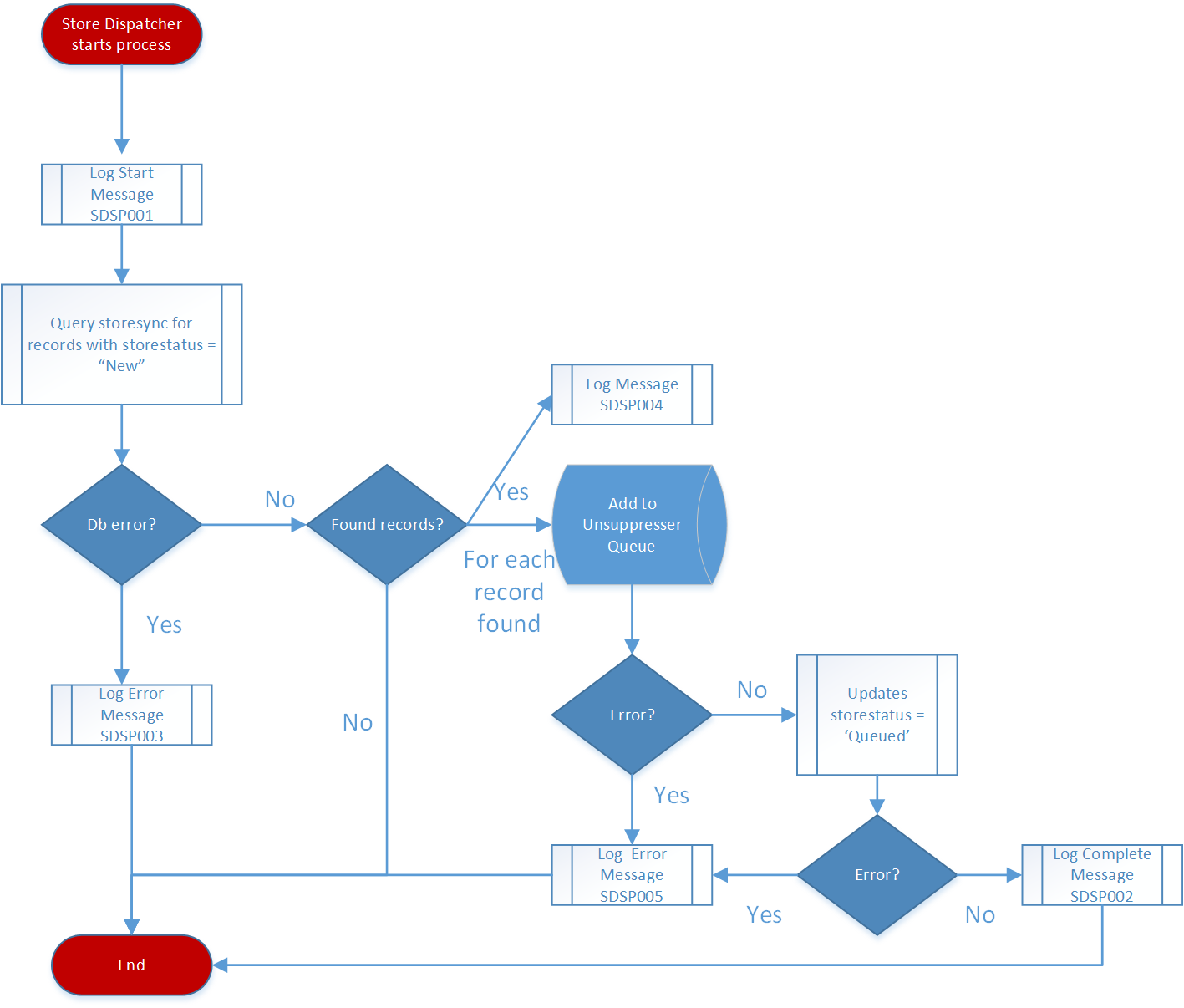
When the suppression period in Salesforce has passed, the stores will be un-suppressed. The Heroku Connect will sync the status and insert a new row into the storesync table with status ‘New’. The storeDispatcher and unSuppresser will process the stores using the same design as the regular dispatcher and parser.

### Store Dispatcher Design

This store dispatcher will utilize the “Clock” Process design to run at an interval

every 30 seconds and will read the storesync table to find stores that have storeStatus = “New”.

Once the stores have been designated as “to be unsuppressed” the job will iterate over each new store returned by the query and add to the job queue called “unSuppresserQueue”. After the stores have been added to the unSuppresserQueue the storesync records should have the msgstatus updated to “Queued”.See the below diagram to review overall flow of the store dispatcher.



#### Exception Handling

If you receive an error while retrieving the records with a “New” status, log out to the console the message Error code SDSP003.

If you receive an error while adding the store to the unSuppresser queue, or updating the table for storesync, log out to the console the error code SDSP005.

#### StoreDispatcher Logging

The following additional logging needs to be included in the dispatcher. For example message to log review the section on Logging Codes

1. Initialization of the job log the started message SDSP001.
2. Completion of storedispatcher log the completed message SDSP002.
3. Once stores are found that need to be unsuppressed log out message SDSP004

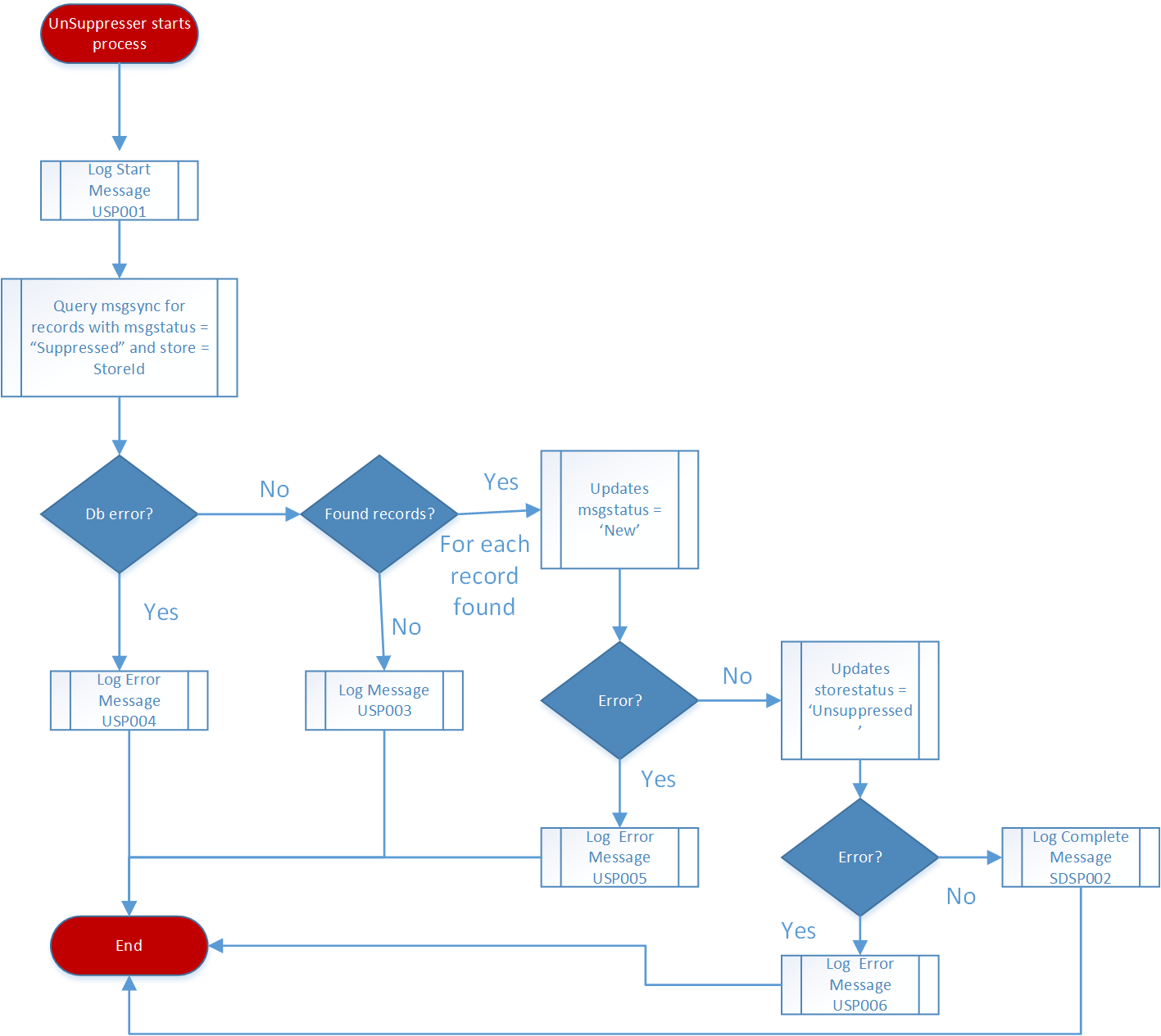
**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| SDSP001 | StoreDispatcher tick | Info |
| SDSP002 | StoreDispatcher completed -> <StoresyncID> | Info |
| SDSP003 | Failed to retrieve 'New' Stores from storesync | Critical |
| SDSP004 | Found store in New status -> <StoreId> | Info |
| SDSP005 | Failed to update StoreSync from “New” to “Queued” or add job to unSuppresser queue | Critical |

### Unsuppresser Design

This unsuppresser will pick up stores from the unSuppresserQueue. For each store, the unsuppresser query the msgsync table to find all messsges with status = ‘Suppressed’ and store = ‘Storeid’. Each message will have its status changed to ‘New’ and follow the normal msg sending process.

After the updates, the stores should have the msgstatus updated to “Unsuppressed”.See the below diagram to review overall flow of the unSuppresser.



#### Exception Handling

If you receive an error while retrieving the records with a “Suppressed” status with the selected store, log out to the console the message Error code USP004.

If you receive an error while updating the msgsync record msgstatus to ‘NEW’ log out to the console the error code USP005.

If you receive an error while updating the storesync record storestatus to ‘NEW’ log out to the console the error code USP006.

#### Unsuppresser Logging

The following additional logging needs to be included in the Unsuppresser. For example message to log review the section on Logging Codes

1. When each job starts, log the started message USP001.
2. When each job completes, log completed message USP002.
3. When no suppressed sms record can be found with the store, log message USP 003.

**Logging Codes:**

|  |  |  |
| --- | --- | --- |
| **Message Number** | **Error Message** | **Message Level** |
| USP001 | Start parsing the Store -> <storeId> | Info |
| USP002 | Unsuppressing completed for store -> <storesyncId>, <storeId> | Info |
| USP003 | No SMS records are found in msgsync table with status Suppressed and store -> <storeId> | Info |
| USP004 | Error when trying to find SMS in msgsync table with status Suppressed and store -> <storeId> | Critical |
| USP005 | Failure in updating SMS in MsgSync to NEW -> <sfId>, <storeId> | Critical |
| USP006 | Failure in updating store in StoreSync to Unsuppressed -> <storesyncId>, <storeId> | Critical |

## Job Priority

The lead type custom field on the lead object is put in a formula field ccpm\_leadtype\_\_c on the sms message object. Based on the desirable relative priority shown below, we assign priority values to all the jobs in every queue.

|  |  |
| --- | --- |
| **Lead Type** | **Priority Value** |
| 'Customer SFDC Manual' | 1 |
| 'Prospect SFDC Manual' |
| 'Customer POS Manual' |
| 'Customer Telesales POS Manual' |
| 'Customer Telesales SFDC Manual' |
| 'Prospect Telesales SFDC Manual' |
| 'Indirect Prospect SFDC Manual' |
| ‘Indirect Customer SFDC Manual’ |
| 'Customer Retail CRM Email' |
| 'Customer Abandoned Cart Retail' | 2 |
| 'Customer Abandoned Cart Digital' |
| 'Prospect Abandoned Cart Retail' |
| 'Prospect Abandoned Cart Digital' |
| 'Customer Abandoned Visit' | 3 |
| 'Customer Quotes' | 4 |
| 'Customer Campaign' | 10 |

## Application Monitoring

### Monitoring Queue Activity

Every 20 seconds a job needs to run where the following queues stats are logged out using the console.log statement.

The format below will be used for all queues used within the system (Dispatcher, EST, CST, MST, PST, AKST, HST, CoreSync)

.

sample#Parse Queue.length=<number of jobs queued>

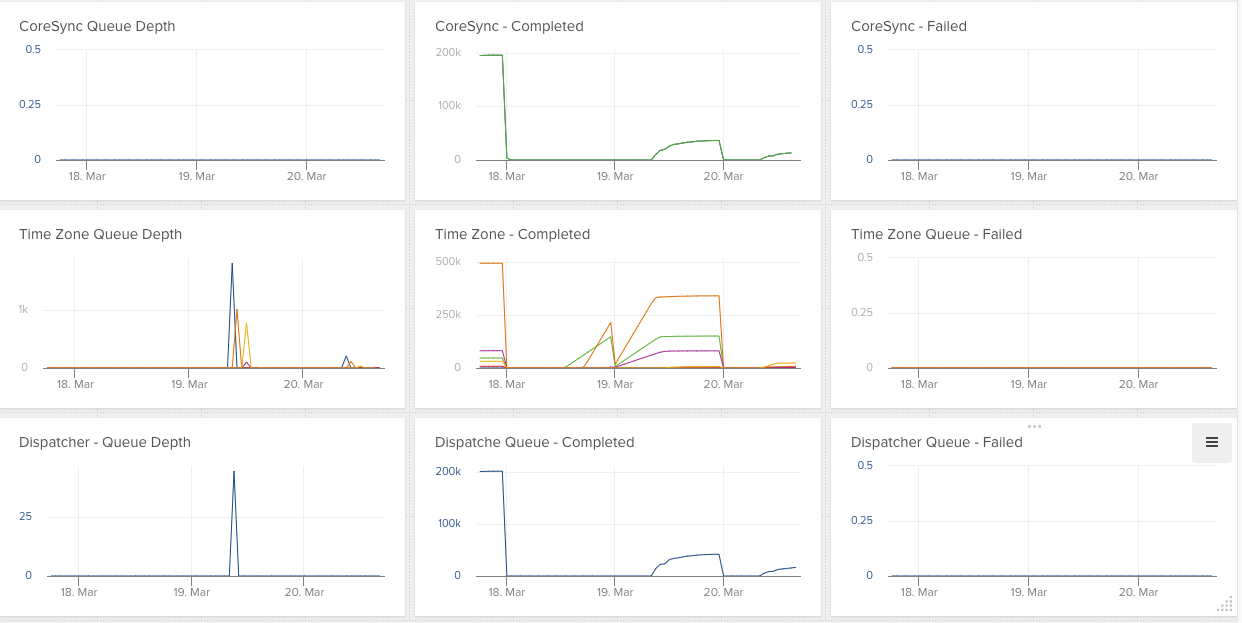
sample#Parse Queue.active=<number of jobs active>

sample#Parse Queue.failed=<number of jobs failed>

sample#Parse Queue.completed=<number of jobs completed>

### Librato Configuration and queue monitoring

We create charts like this for dispatcher, each time zone based, and coresync queue so that we can have historical reporting over each queue that will show number of messages completed, failed and queued.



## Load Balancing

### Current bandwidth in jobs/sec allocated for different time zones

|  |  |
| --- | --- |
| TimeZone | Speed in job/sec |
| AKST | 1 |
| CST | 4 |
| EST | 15 |
| HAST | 1 |
| MST | 2 |
| PST | 2 |
| TOTAL | 25 |

Current SMS per second entry and exit points –

|  |  |
| --- | --- |
| Module | Speed in job/sec |
| Dispatcher | 23 |

### Load balancing by adjusting speed on time zone queues at specific time intervals

1. Start of EST Business Hours-

|  |  |
| --- | --- |
| TimeZone | Speed in jobs/sec |
| AKST | 0 |
| CST | 0 |
| EST | 20 |
| HAST | 0 |
| MST | 0 |
| PST | 0 |
| TOTAL | 20 |

1. Start of CST Business Hours-

|  |  |
| --- | --- |
| TimeZone | Speed in jobs/sec |
| AKST | 0 |
| CST | 4 |
| EST | 16 |
| HAST | 0 |
| MST | 0 |
| PST | 0 |
| TOTAL | 20 |

1. Start of MST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 0 |
| CST | 4 |
| EST | 14 |
| HAST | 0 |
| MST | 2 |
| PST | 0 |
| TOTAL | 20 |

1. Start of PST Business Hours-

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 0 |
| CST | 4 |
| EST | 12 |
| HAST | 0 |
| MST | 2 |
| PST | 2 |
| TOTAL | 20 |

1. Start of AKST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 1 |
| CST | 4 |
| EST | 11 |
| HAST | 0 |
| MST | 2 |
| PST | 2 |
| TOTAL | 20 |

1. Start of HAST Business Hours-

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 1 |
| CST | 4 |
| EST | 10 |
| HAST | 1 |
| MST | 2 |
| PST | 2 |
| TOTAL | 20 |

1. Close of EST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 1 |
| CST | 14 |
| EST | 0 |
| HAST | 1 |
| MST | 2 |
| PST | 2 |
| TOTAL | 20 |

1. Close of CST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 1 |
| CST | 0 |
| EST | 0 |
| HAST | 1 |
| MST | 16 |
| PST | 2 |
| TOTAL | 20 |

1. Close of MST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 1 |
| CST | 0 |
| EST | 0 |
| HAST | 1 |
| MST | 0 |
| PST | 18 |
| TOTAL | 20 |

1. Close of PST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 19 |
| CST | 0 |
| EST | 0 |
| HAST | 1 |
| MST | 0 |
| PST | 0 |
| TOTAL | 20 |

1. Close of AKST Business Hours -

|  |  |
| --- | --- |
| TimeZone | Speed in msg/sec |
| AKST | 0 |
| CST | 0 |
| EST | 0 |
| HAST | 20 |
| MST | 0 |
| PST | 0 |
| TOTAL | 20 |

|  |  |
| --- | --- |
| Module | Speed in msg/sec |
| Dispatcher | 20 |

# Contacts

The following table shows the key points of contact involved in this project.

|  |  |
| --- | --- |
| Contact Name | Area of Responsibility |
| Matt Fisher/Charlie Guo | Architect |
| Huang Zhiyi/Brigadesh Chandrasekar | Developers |
| Kathleen Casey | Feature owner |