Billdesk Repository

Table of Contents

[Scope 3](#_Toc444101744)

[Software dependencies 4](#_Toc444101745)

[Menu list 5](#_Toc444101746)

[Tables:- 6](#_Toc444101747)

[Creation of transactional data files from live environment. 7](#_Toc444101748)

[a) PGI 7](#_Toc444101749)

[b) BMS (10.20.7.2) 7](#_Toc444101750)

[c) CP (10.20.7.2) 8](#_Toc444101751)

[Config Management. 9](#_Toc444101752)

[a) PGI 9](#_Toc444101753)

[b) BMS 9](#_Toc444101754)

[c) CP 10](#_Toc444101755)

[Invoice tax/charges management 10](#_Toc444101756)

[Invoice calculation and template creation 11](#_Toc444101757)

[Read from files and push transactional data into repo database 11](#_Toc444101758)

[Logic and process for calculations 13](#_Toc444101759)

[Calculate the charges: 15](#_Toc444101760)

[BankShare calculation and template creation 19](#_Toc444101761)

[Calculate the share: 20](#_Toc444101762)

[Annexure 1 22](#_Toc444101763)

# Scope

This document will provide a detailed description of the requirements and implementation for the Billdesk Repository Invoicing System(BRIS). This document will allow a complete understanding of what is to be expected of the BRIS to be constructed. The clear understanding of the BRIS and its’ functionality will allow for the correct software to be developed for the end user and will be used for the development of the future stages of the project. This document will provide the foundation for the project. From this document, the BRIS can be designed, constructed, and finally tested.

This document will be used by the accounts team constructing the invoices for the merchants. The accounts team will use this to fully understand the expectations of this BRIS to construct the appropriate invoices. The end users will be able to use this document as a test to check if the software will be constructing the system to their expectations. If it is not to their expectations the end users can specify how it is not to their liking and requirements.

# Software dependencies

There are list of dependencies in the project as below:

1. Spring framework supporting jars
2. Docx read and write supporting jars
3. Database jdbc connection supporting jars
4. Ant for building project and deployment

# Menu list

1. Add Group User
2. View Group User
3. Add User
4. View User
5. Add Group Merchant
6. View Group Merchant
7. Add Merchant
8. View Merchant
9. Add Merchant Pay Config
10. View Merchant Pay Config
11. Add Merchant Rate Config
12. View Merchant Rate Config
13. Download Invoice

# Tables:-

* GROUP\_MERCHANT;
* MERCHANT\_MASTER;
* TAX\_MASTER;
* BILLDESK\_ID\_GENERATOR;
* MERCHANT\_RATE\_CONFIG;
* MERC\_INV\_RATE\_TYPE\_MASTER;
* ZONE\_COUNTER;
* TRANSACTION\_DETAILS;
* INVOICE\_FILE\_INFO;
* BIN\_DETAILS;
* MICR\_BRANCH\_NAME;
* SIP\_CONF\_MASTER;
* MERC\_INVOICE\_DETAILS;
* MERC\_INVOICE\_MASTER;
* BANK\_MASTER;
* BANK\_SHARE\_CONFIG;

Table structures attached in annexure 1

# Creation of transactional data files from live environment.

Creation of transactional data files will be pulled into system for repo from different other systems. This is a daily process and requires monitoring of pulled data from systems in repo defined files for further process.

We should keep track in the sheet (attached in annexure -1) for reference and maintenance. This sheet should be updated and monitored on daily basis. We can manage sheet with the mail reference received daily by pulled crone data server on daily basis.

Data monitoring is required because we have to make sure that complete and correct data is pulled from different systems.

We have to resolve the issues in pulling the data on regular basis so that at the end we will get correct transactional data.

**TODO**: One dedicated person for monitoring and resolving issues.

Below is the list of systems from where we are going to pull the data and the process defined for the pulling.

## PGI

Need to check with Ankit for the same.

## BMS (10.20.7.2)

**Crone Shell script Path:**

/home2/sh3dsk/Applications/BMSAPPS/bin/BMSRepo.sh

(BMSRepo.sh file attached in annexure 1)

**Crone Timing :**

1:23 am daily (takes around 1 hour to create the files)

**Crone Report Auto mail format:**

BMS Repo File Creation Process Completed:

Txn File Name : REPO\_BMS\_TXN\_20151215012302\_1130427.txt

Total Number of Txns : 1130427

BMS Repo process is completed for : 20151215

Cust File Name : REPO\_BMS\_TXN\_CUST\_20151215012302\_1130427.txt

Total Number of Txns : 1130427

BMS Repo process is completed for: 20151215

**Created File Path:**

/IBMBULKFiles/Invoice/Input/Created/TxnData

/IBMBULKFiles/Invoice/Input/Created/CustData

## CP (10.20.7.2)

**Crone Shell script Path:**

/home2/sh3dsk/Applications/BillPay/bin/RepoFileCreation.sh

(RepoFileCreation.sh file attached in annexure 1)

**Crone Timing :**

1:21 am daily (takes around 30-45 mins to create the files)

**Crone Report Auto mail format:**

CardPay Repo File Creation Process Completed:

Txn File Name: REPO\_CP\_TXN\_20151215012109\_31429.txt

Total Number of Txns: 31429

CardPay Repo process is completed for :20151215

Cust File Name: REPO\_CP\_TXN\_CUST\_20151215012109\_31429.txt

Total Number of Txns: 31429

CardPay Repo process is completed for :20151215

**Created File Path:**

/IBMBULKFiles/Invoice/Input/Created/TxnData

/IBMBULKFiles/Invoice/Input/Created/CustData

# Config Management.

This is the process where manual interaction required. Mostly accounts team will be the correct ones to handle the same.

Below is the list of systems from where we are going to pull the config data and the defined process.

Server local IP: **10.20.7.2**

## PGI

Need to discuss with Ankit to create one single file and schedule crone for the same.

## BMS

**Shell script Path:**

/home2/sh3dsk/Applications/Reports/bin/ bmsrepoconfig.sh

(bmsrepoconfig.sh file attached in annexure 1)

**Crone Timing :**

Yet to discuss. Right now it is a manual process by Victor.

**Config Auto mail:**

Mail Subject: BMS REPO Config- 25 Sept to 4th Oct

Mail Text : BMS REPO Config.

File in attachment : bmsrepoconfigpayin\_25 Sept 2015.txt

## CP

**Shell script Path:**

/home2/sh3dsk/Applications/Reports/bin/ cprepoconfig.sh

(cprepoconfig.sh file attached in annexure 1)

**Crone Timing :**

Yet to discuss. Right now it is a manual process by Victor.

**Config Auto mail:**

Mail Subject: CP REPO Config- 25 Sept to 4th Oct

Mail Text : CP REPO Config.

File in attachment : cprepoconfig\_25 Sept 2015.txt

**Existing flow:**

After getting the txt file, accounts team will copy paste data in excel and update the rate for all the configs as per GROSS and NET merchants. There are many limitations in the copy paste directly ex. we have many fields where we required n times zero and excel will convert in one zero. This will be a very critical process because on basis of all the parameters we are making nemonics which will decide the rate of merchant’s transactions.

After the update in the sheet by accounts team, Victor will generate the inserts query and inserts the data in the repo system for the calculation.

**TODO**: Create one interface for accounts team to upload configs in csv format for merchants on regular basis.

# Invoice tax/charges management

Mostly taxes and charges are not changing on daily basis but may change in the middle of the month/week.

**TODO**: Interface to change taxes and charge for particular time period/time range.

# Invoice calculation and template creation

Invoice calculation and template creation are the two different processes.

Initial process will read the file from the created path and dump the files after process for the calculation in processed path.

Charges calculation of all the transactions according to different rate config will be a time taking process that’s why we are calculating on the same day when we are inserting data in the repo database. We are picking 2 lakhs transactions at a time for the month and calculate charges and taxes and insert it in the database.

Right now we are generation multiple scripts for calculation and charges and taxes for the transactions depending upon the total number of transaction for the day.

Reports of processed data will also generate and stored in the specific path.

## Read from files and push transactional data into repo database

Server local IP: **10.20.7.2**

**Shell script Path:**

/home2/sh3dsk/Applications/Invoice/bin/ProcessRepoTxns.sh

(ProcessRepoTxns.sh file attached in annexure 1)

**Crone Timing :**

04:01, 09:01, 14:01 daily

**Picked File Path:**

/IBMBULKFiles/Invoice/Input/Created/TxnData

/IBMBULKFiles/Invoice/Input/Created/CustData

**Transfers the file after processing :**

/IBMBULKFiles/Invoice/Input/Processed/TxnData

/IBMBULKFiles/Invoice/Input/Processed/CustData

**Generates the processing Reports :**

/IBMBULKFiles/Invoice/Input/Reports

**Note :** File name contains total file name and error records count (ex: REPO\_CP\_TXN\_CUST\_20151216012109\_220506\_PR\_20151216060622\_220506\_0.html)

**Invoice creation:**

Invoice creation needs number of unique merchants for that particular month. Right now this is a manual task by Victor.

Get total unique MERCHANT\_ID from the spool.

Create one spool every time to get the unique MERCHANT\_IDs for that period ex. NOV15\_INV\_GROUPMERCHANTID.sql attached in annexure1. Run that spool and arrange the data in the format required for the invoice template creation ex. NOV15\_INV\_GROUPMERCHANTID.txt attached in annexure1. Now copy paste that formatted text in the shell script which will generate the invoice for that particular merchant ex. GenerateMercInvoice.sh attached in annexure 1.

**TODO**: An Interface from where one can trigger the process of getting unique merchant and arrange all unique merchants in required format for invoice creation and also interface for invoice creation for merchant and time interval.

## Logic and process for calculations

For calculation java class used is MerchantInvoiceCalculateChargesSingleTable. Main method of class will take two parameters:

1. dateStr : Date for which program will pick the transaction from transaction table. Default will be null.
2. countTotal : How many times process will pick the transaction from database table transaction\_details in repodb for the calculation. Default will be 1.

Now create a connection with database. Get list of Group merchant and put it in HashMap for further use. Now get all required data for the calculation i.e Rate Type Master, Card bin details, Merchant Master, Merchant rate config and SIP Config. For CITI billers we have configs in properties files. Will discuss next.

Get transactions from TRANSACTION\_DETAILS table for the particular date if specified else random from the table where MERC\_INV\_STATUS IS NULL OR MERC\_INV\_STATUS = 'R' in the batch of 2 lakhs only. This is because we found if we go for more transactions connection timeout and JAVA heap out of memory problem comes in picture. So research says that 2 lakhs is the best count for fetching in existing system.

If count of transactions is zero then return from the process. Now get customer details for the transactions and set details for all the transactions.

Get transaction object from the list in the loop one by one. There may be some transaction that are not valid for invoicing so we will ignore them and put repo\_txn\_id in the ignore transaction list. The logic for valid transactions are:

1. If SYSTEM\_ID = CardPay then transaction status should be either success or failed
2. If SYSTEM\_ID = BMS then transaction status should be success.
3. If SYSTEM\_ID = PGI then transaction status should be success.
4. Else transaction are not valid for invoicing.

We are also ignoring some transactions which comes in below criteria and put them in transactionCardPayUBPType List.

1. If SYSTEM\_ID = CardPay and MerchantId ends with UBP and sourceId ends with UBP
2. If SYSTEM\_ID = CardPay and MerchantId ends with VBP and sourceId ends with VBP
3. If SYSTEM\_ID = CardPay and MerchantId ends with UBP and sourceId ends with UBPBANK

Check payconfig present in the system for calculations with the combination of unique parameters i.e.

SYSTEM\_ID~Merchant\_ID~Bank\_ID~PayeeBank\_ID~ProductCode~Merchant\_ID\_BANK. If not present in the merchantRateConfig hashmap then put that transaction in config error list and log that rate config error in the log file and continue with the next transaction.

If merchant rate config present then get the ratetypemnemonic from the MerchantRateConfig. Get revised Rate Type Mnemonic based on card bin. Check if customer details present for that transaction or not. If not then log the error wit message “Processing\_Error: No TxnCustDetails found for:” that repo\_txn\_id and continue with next transaction. If transaction details found then get the cardbin from the customer details for that transaction and validate null, blank and NA and get CardBin details from the binDetails hashmap with the key cardbin. If ratetypemnemonic starts with “C\_” and bindetails is not null then we will check for different ratetype mnemonic with the CardProductId from bindetails appended with underscore and previous ratetypemnemonic.

Get Merchant Inv Rate Type Master Model from the ratetypemnemonic key from the mercInvRateTypeMaster HashMap. If mercInvRateTypeMaster List is null for the card bin ratetypemnemonic key then check with the original ratetypemnemonic key. If with both the cardbin ratetypemnemonic key and original ratetypemnemonic key mercInvRateTypeMaster List is null then ignore the transaction and log it and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Now we have mercInvRateTypeMaster List for the calculations and set the same for that transaction for further use.

Now check the merchant for the unique key combination to find out Group Merchant ID for the invoicing purpose because we are generating invoices for the GroupMerchantId. Unique merchant key id combination of SYSTEM\_DI~MERCHANT\_ID~SUB\_MERCHANT\_ID. Find out merchant master for that unique key in the Merchant master hashmap with the unique key. If not found then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

We have now merchant master and group merchantId. Set the same in the transaction. For some merchant we have some logic for the group merchant search. Find below the logic:

1. If groupmerchantid is SBIELRCESB and transaction additionalInfo2 is not null then group merchant id will be groupMerchantId+"\_"+AdditionalInfo2(MICR Branch Code)
2. If groupmerchantid is SBIELRCand transaction additionalInfo3 is not null then group merchant id will be groupMerchantId+"\_"+AdditionalInfo3(MICR Branch Code)
3. If groupmerchantid is MTNLDELCRMand transaction additionalInfo2 is not null and starts with 900 then group merchant id will be groupMerchantId+"\_"+900

To check is there a split in invoice at merchant rate config level we will check group merchantId in merchantRateConfig and if it is NA then continue with the above groupmerchantId else get group merchantId from merchantRateConfig and continue. If we found nothing in groupmerchant hashmap for the groupmerchantid key then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list. If group merchant found and detail format not found for that group merchant then also log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list. From details format we are able to check the merchant format for invoicing and if it is CITI then we will discuss later and put the same CITI transaction in the transactionCitiMerchantId List.

Check If merchant is Axamfb and SIP is D for different rateTypeMnemonic. If group merchant id is AXAMFB then get SIP day from the transaction. Get sIpConfMaster from the sipconfmaster hashmap with the key SYSTEM\_ID~MERCHANT\_ID. If sipConfMaster is null then sip is NA. If not then get infofieldname from the sIpConfMaster. If infofieldname is null or empty then sip is NA. If not then get the token position and token separator from sIpConfMaster and get SIP from transaction field mentioned in sIpConfMaster with token separator and token position. If SIP equals D then rateTypeMnemonic will be rateTypeMnemonic+”\_D”. Now check mercInvRateTypeMasterList for new rateTypeMnemonic. If could not get the rate master for revised mnemonic which came based on sip day then shifted to the original mnemonic.

### Calculate the charges:

**For Gross:**

1. First step is to getInvMonthBase timestamp.
2. If Invoice gross date base in group merchant is TRANSACTION\_TIMESTAMP then base will be transaction timestamp.
3. If Invoice gross date base in group merchant is MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction.
4. If Invoice gross date base in group merchant startsWith TRANSACTION\_RECON\_DATE then base will be as follow:
   1. If SYSTEM\_ID equals CardPay then base will be recon date (debitfile date)
   2. If indexof(“~”)==-1 then base will be recon date (debitfile date)
   3. If indexof(“~”)!=-1 then substring Invoice gross date base in group merchant with indexof(“~”)+1 and if result equals TRANSACTION\_TIMESTAMP then base will be transaction timestamp else if result equals MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction else log the config error and put it in transactionConfigErrorId List.
5. Else log the config error and put it in transactionConfigErrorId List.
6. If baseDate is null then log the config error and put it in transactionConfigErrorId List.
7. Include transactions happening on 1st of month to previous month for list of merchants i.e {"BSES\_423", "BSES\_493", "BSES\_503", "BSES\_533", "BEST"}. Include transactions happening on last day of previous month to this month for list of merchants i.e {"TATASKYDS"}. Include transactions happening on 2nd of month to previous month for list of merchants i.e {"BSNLKYN"}. Include transactions happening on 2nd of month to previous month if 1st is Sunday for list of merchants i.e {"BSES\_423", "BSES\_493", "BSES\_503", "BSES\_533"}.
8. Calculate Invoice charges

For CITI invInvRateTypeMnemonic will be invInvRateTypeMnemonic+"\_FOR\_CITI". Get mercInvRateTypeMaster list from mercInvRateTypeMaster hashmap for the invRateTypeMnemonic lie in date range of timestampDateBase calculated above. If list is empty then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Now get the final mercInvRateTypeMaster from the list above. Get the slabtype for the first element from the list to start the calculation. Below is the logic to get the final mercInvRateTypeMaster.

Note : SlabType - A - Refers to the Slab based on the Amount of the transaction; hence, the MinAmt/MaxAmt will be specified; while N - Refers to the Slab based on the Number of the transactions; hence the MinTxn/MaxTxn will be specified, T - Total of txn amount per month would be applied and again MinAmt and MaxAmt fields will be used in this case. If NA then no slab

**If slabtype is NA:**

1. If slabtype is NA and more than two mercInvRateTypeMaster found from the list then log the config error and put it in transactionConfigErrorId List because for slabtype NA only two mercInvRateTypeMaster is possible. Continue with next transaction in the list.
2. Just comparing slab types again to make sure whether config has any mistake. If then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.
3. If transaction status is SUCCESS and rateApplicable is empty or B or S then pick the mercInvRateTypeMaster for further calculation. Else if transaction status is Failed and rateApplicable is B or F then pick the mercInvRateTypeMaster else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

**If slabtype is other then NA:**

1. If slabtype is N or T get transactions and store them in hashmap for further calculation with key group merchantId ~ MercGrossInvoiceMonth. Get total transactions and total amount for the merchant.
2. If slabtype is A get min and max amount for merchant transactions.
3. If slabtype is N get min and max transaction count for merchant transactions.
4. If slabtype is T get min and max amount for merchant transactions.
5. Else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Check the final mercInvRateTypeMaster and processed further else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list. If merchant is CITI then calculated which we will discuss later else calculate FinalChargesAndTaxes for other merchants and set same in the transaction.

Check service tax and applicability and set zero if not null and groupMerchant.getSTaxApplicable().equals("N"). Set merc invoice month to null for further check of the same with some more logic.

If there is gross charges and ECS success and failure charges or invoice transaction charges present then set merc gross invoice month to timestamp base. Else if there is no tomerchant charges and transaction charges we will check default invoice type. If transaction has no surcharge, no net charges and no gross charges, and default invoice type is net then set merc net Invoice month for timestamp calculated below for net transaction.

1. If Invoice net date base in group merchant is TRANSACTION\_TIMESTAMP then base will be transaction timestamp.
2. If Invoice net date base in group merchant is MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction.
3. If Invoice net date base in group merchant startsWith TRANSACTION\_RECON\_DATE then base will be as follow:
   1. If SYSTEM\_ID equals CardPay then base will be recon date (debitfile date)
   2. If indexof(“~”)==-1 then base will be recon date (debitfile date)
   3. If indexof(“~”)!=-1 then substring Invoice net date base in group merchant with indexof(“~”)+1 and if result equals TRANSACTION\_TIMESTAMP then base will be transaction timestamp else if result equals MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction else log the config error and put it in transactionConfigErrorId List.
4. Else log the config error and put it in transactionConfigErrorId List.
5. If baseDate is null then log the config error and put it in transactionConfigErrorId List.

If transaction has no surcharge, no net charges and no gross charges, and default invoice type is surcharge then set merc surcharge Invoice month for timestamp calculated below for surcharge transaction and set zero charges and service tax for surcharge.

1. If Invoice surcharge date base in group merchant is TRANSACTION\_TIMESTAMP then base will be transaction timestamp.
2. If Invoice surcharge date base in group merchant is MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction.
3. If Invoice surcharge date base in group merchant startsWith TRANSACTION\_RECON\_DATE then base will be as follow:
   1. If SYSTEM\_ID equals CardPay then base will be recon date (debitfile date)
   2. If indexof(“~”)==-1 then base will be recon date (debitfile date)
   3. If indexof(“~”)!=-1 then substring Invoice surcharge date base in group merchant with indexof(“~”)+1 and if result equals TRANSACTION\_TIMESTAMP then base will be transaction timestamp else if result equals MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction else log the config error and put it in transactionConfigErrorId List.
4. Else log the config error and put it in transactionConfigErrorId List.
5. If baseDate is null then log the config error and put it in transactionConfigErrorId List.

If there is surcharge transaction charges exists then set merc surcharge invoice month to surcharges invoice month base as calculated above. For PGI surcharge is net charges plus net service tax. If net and surcharge charges are same then net charges will be zero for that transaction else final net charges will be difference of net charges minus net charges. If groupMerchant.getSTaxApplicable().equals("N") ok else remove services tax and charges from surcharge which is inclusive of both and set surcharge charges and service tax to the transaction.

If there is net charges set merc net invoice month to net invoice month timestamp base.

There are transactions which has to excluded from the invoicing part. Check MercInvRateTypeMaster().getInvExclude() transaction status for failed transaction. Failed transaction, not to be included and transaction status is also failed. Nullify all the gross, net, surcharge merc inv months. Transaction - not to be included in any invoice but should be present in merc\_inv\_details for MIS purposes.

Finally update the transaction in transaction\_details table and insert in BANK\_SHARE\_DETAILS table accordingly.

# BankShare calculation and template creation

For Bank Share calculation java class used is BankShareReportCalculateCharges. Main method of class will take two parameters:

1. dateStr : Date for which program will pick the transaction from transaction table. Default will be null.
2. countTotal : How many times process will pick the transactions from database table transaction\_details in repodb for the calculation. Default will be 1.

Now create a connection with database. Get list of Group Bank and put it in HashMap for further use. Now get all required data for the calculation i.e Rate Type Master, Card bin details, Bank Master, BankShare config.

Get transactions from TRANSACTION\_DETAILS table for the particular date if specified else random from the table where BANK\_SHARE\_STATUS = 'N' in the batch of 2 lakhs only. This is because we found if we go for more transactions connection timeout and JAVA heap out of memory problem comes in picture. So research says that 2 lakhs is the best count for fetching in existing system.

If count of transactions is zero then return from the process. Now get customer details for the transactions and set details for all the transactions.

Get transaction object from the list in the loop one by one. There may be some transaction that are not valid for invoicing so we will ignore them and put repo\_txn\_id in the ignore transaction list. The logic for valid transactions are:

1. If SYSTEM\_ID = CardPay then transaction status should be either success or failed
2. If SYSTEM\_ID = BMS then transaction status should be success.
3. If SYSTEM\_ID = PGI then transaction status should be success.
4. Else transaction are not valid for invoicing.

Check shareConfigKey present in the system for calculations with the combination of unique parameters i.e. SYSTEM\_ID~Merchant\_ID~Bank\_ID~PayeeBank\_ID~ProductCode~Merchant\_ID\_BANK~ME\_CODE. If not present in the bankShareConfig hashmap then put that transaction in config error list and log that rate config error in the log file and continue with the next transaction.

If bank share config present then get the ratetypemnemonic from the BankShareConfig. Get revised Rate Type Mnemonic based on card bin. Check if customer details present for that transaction or not. If transaction details found then get the cardbin from the customer details for that transaction and validate null, blank and NA and get CardBin details from the binDetails hashmap with the key cardbin. If ratetypemnemonic starts with “C\_” and bindetails is not null then we will check for different ratetype mnemonic with the CardProductId from bindetails appended with underscore and previous ratetypemnemonic. Get card type from the CardProductId. If charat(2) of CardProductId is C then set cardtype is creditcard in transaction and if D then cardtype is debitcard.

Get BankShare Rate Type Master Model from the ratetypemnemonic key from the bankShareRateTypeMaster HashMap. If bankShareRateTypeMaster List is null for the card bin ratetypemnemonic key then check with the original ratetypemnemonic key. If with both the cardbin ratetypemnemonic key and original ratetypemnemonic key bankShareRateTypeMaster List is null then ignore the transaction and log it and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Now we have bankShareRateTypeMaster List for the calculations and set the same for that transaction for further use.

Now check the bank for the unique key combination to find out Group Bank ID for the invoicing purpose because we are generating reports for the GroupBankId. Unique bank key id combination of SYSTEM\_DI~BANK\_ID~PAYEE\_BANK\_ID. Find out bank master for that unique key in the Bank master hashmap with the unique key. If not found then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

To check is there a split in report at bank rate config level we will check group bankId in bankRateConfig and if it is NA then continue with the above groupbankId else get group bankId from bankRateConfig and continue. If we found nothing in groupbank hashmap for the groupbankid key then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list. If group bank found and detail format not found for that group bank then also log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list. From details format we are able to check the bank format for reporting

### Calculate the share:

1. First step is to getBankShareMonthBase timestamp.
2. If bankshare date base in group bank is TRANSACTION\_TIMESTAMP then base will be transaction timestamp.
3. If bankshare date base in group bank is MERCHANT\_TID\_DATE then base will be MerchantTID date from transaction.
4. Else log the config error and put it in transactionConfigErrorId List.
5. Calculate bank share charges.

Get bankShareRateTypeMaster list from bankShareRateTypeMaster hashmap for the bankShareMnemonic lie in date range of timestampDateBase calculated above. If list is empty then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Now get the final bankShareRateTypeMaster from the list above. Check if bank share is merInv based or not from the groupbank.

1. If yes then get the slabtype form the first element from the bankShareRateTypeMasterList to start the calculation. Below is the logic to get the final bankShareRateTypeMaster.

**If slabtype is NA:**

1. If slabtype is NA and more than two bankShareRateTypeMaster found from the list then log the config error and put it in transactionConfigErrorId List because for slabtype NA only two bankShareRateTypeMaster is possible. Continue with next transaction in the list.
2. Just comparing slab types again to make sure whether config has any mistake. If then log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.
3. If transaction status is SUCCESS and rateApplicable is empty or B or S then pick the bankShareRateTypeMaster for further calculation. Else if transaction status is Failed and rateApplicable is B or F then pick the bankShareRateTypeMaster else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

**If slabtype is other then NA:**

1. If slabtype is N or T get transactions and store them in hashmap for further calculation with key bankShareMonth~groupbankId~ bankShareTypeMnemonic. Get total transactions and total amount for the merchant.
2. If slabtype is A get min and max amount for merchant transactions.
3. If slabtype is N get min and max transaction count for merchant transactions.
4. If slabtype is T get min and max amount for merchant transactions.
5. Else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.
6. If bankShare is not mercInvoice based then check bankShareRateTypeMasterList list. If it is not equals to 1 then and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Now find the final bankShareRateTypeMaster for SUCCESS and FAILED transaction. If SUCCESS then rateApplicable should be empty, B or S and if FAILED then rateApplicable should be B or F else log the config error and put it in transactionConfigErrorId List. Continue with next transaction in the list.

Calculate FinalChargesAndTaxes for merchants and set same in the transaction.

# Annexure 1



