Rafiur Rashid

RBC  [Company address]

**Auto Creation of USD/CAD Side of The CAD/USD Accounts**

**Introduction**

Currently in Clientlink USD (001) sides of the accounts do not appear until they have been funded. Once the accounts have had money in them, they will appear with a balance, or a zero if money has been taken out, but until there has been some activity there is no instance of the account on Clientlink.

This causes issues for Flexpak as well as modeling. For Flexpak, an account cannot be added to a household as the account does not appear, and if the team funds the account later, they have to remember to go back and add it to the household. This is a commonly overlooked process and creates issues particularly for PIM households which are required (regulatory) to be complete and sent quarterly. For modeling, an account cannot be added to a model unless the account shows up on the Clientlink.

Moreover, the “no US side appearing” issue has been further brought to light with the new ‘multi-currency’ modeling, which is currently being used by about 20 teams, with more to come likely. To use multi-currency modeling, at first, a multi-currency portfolio (the CAD and USD sides combined) need to be created, which cannot be done unless both accounts appear on the Clientlink.

Lastly, there is a likelihood of TAC will pull all accounts in a Clientlink household (that is the current working model). If the US side does not exist and is therefore not in the Clientlink household, it will also not be on the TAC. The TAC will not let us add unofficial accounts but this will create issues for teams in the setup and management of their TAC sheets.

In order to solve the four issues mentioned above (Flexpak, modeling, multi-currency modeling and TAC) scripts have been developed that automatically creates US side of the Canadian Dollar Accounts and the reverse for the existing lonely accounts. In future, if CAD/USD side an account is created the nightly batch tasks immediately creates the USD/CAD side of the account so that the other currency side of the account will be available to be attached to the model by the IA on next day.

**Development Process**

The scripts developed basically follows the **ETL** (Extract, Transform and Load) process where the USD and CAD 13 digit accounts that do not have the other currency account (i.e. the 13 digit CAD account exists but not the matching USD account, and vice versa) were extracted from the database. Then, the extracted dataset was transformed to the SIF file. Finally, for each of these ‘lonely’ accounts, the matching account was created via an ‘ACC02’ loader. Following is a summary of the steps followed during this development process:

* Initial Setup:

The first step involved creating a PTR (*SCR 57732*) and reserving a build number (*61JD*). Then, a matching CA package was created for the build.

Metadata:

Prior to developing the new scripts, some parameters (e.g. Interface ID, Sys. parms) were set up on the DB’s metadata tables. The existing SQL scripts (that inserts the parameters into specific metadata table) were checked out and modified.

Extract:

In order to perform the task of extraction the following algorithm was used into the SQL statement [Appendix A]:

* Search the DB for USD and CAD 13 digit accounts that do not have the other currency account i.e. the 13 digit CAD acct exists but not the matching USD acct, and vice versa.
* Do this only for certain charge\_type\_cds (sysparm controlled)
* Do not do this for certain acct\_type\_cds (sysparm controlled)
* Flipped the last 3 digits (currency code) of the 13 digit back office account no. so that it converts the account no. in the other currency.
* The variable length bkoff\_client\_id was right-padded to length 10 characters to match the Production SIF file format.

Transform:

For this project, the transform step simply mapped the output from the *Extract* step to the SIF file. Being fixed-width records; the records were trimmed and placed to the SIF file in a special format so that it conforms to the SIF specification.

Load:

Finally, the load part was handled by the Temenos copy loader – ‘ACC02’ that created the new accounts in the other currency.

Unit Testing:

Throughout this task, an incremental development process was followed where each step was developed and tested before proceeding to the next step.

Each table was tested to check if the parameter was added to the table properly and rollbacked to check its consistency. The SQL statements that tests for the addition of a new interface ID is as follows:

**--** *Counts total # of records in the table*

select \*

from controller.bia\_interfaces i

order by i.interface\_id;

**--** *Before 664 records*

***--*** *After 665 records*

**--** *Checks the tuple with the new interface ID*

select \*

from controller.bia\_interfaces i

where i.interface\_id = 'PMOLOTHCURRACC'

order by i.interface\_id;

**--** *Rollback (Expects 664 recodes)*

delete controller.bia\_interfaces i

where i.interface\_id = 'PMOLOTHCURRACC';

For the extract, edge cases were tested with 0 and only 1 record and then with some arbitrary number of records which was verified by the number of record counts printed in the footer statement. Eventually, it was tested for the entire set of records and verified the result by executing the SQL statement directly on the Database (using SQL Navigator).

Similarly, the transform step was also tested carefully to match the format of the SIF file used by the production. Finally, the loader was tested by creating sibling accounts for a few accounts into the DB table.

**Discussion**:

The major challenge experienced during this development was:

Waiting for the EOT file:

After running and testing different modules when the Xeye Job (uq1 script) was run, the execution halted and kept waiting at the *Extract* step for the ‘*EOT*’ file.

Resolution:

During the execution of the *uq1* scripts, different files related to the job are generated for logging purpose or as Start/End token. However, they are cleaned up at the end when the job completes successfully. But, if the job somehow fails before it finishes, the job needs to be restarted from the beginning and at this point, a cleanup [Appendix B] is required to be performed to get the job run properly next time. In order to automate the cleanup process and save time, a script was developed to use on the local machine. This approach made the cleanup process very convenient.

**Result**:

After preparing the manifest file, the application was built and deployed using Jenkins and UCD. The successful execution of the script created around 131,000 accounts in the other currencies for which there were existing accounts in only one currency (CAD/USD).

**Conclusion**:

It was a great learning experience being able to perform the development task for a complex application - Clientlink. It was a great opportunity to understand how different components work together. I learned how to be organized while working on a large piece of software as well as writing quality code and meaningful commenting for future maintenance and consistency. Moreover, the importance of unit testing was also discovered.