IDP Design: Bank Unit data (via LSDB)

# Background

The Legal Structure Database (LSDB) holds details of UBS legal entities – their structure, ownership, and key corporate information. The data is mastered in an Oracle database in Switzerland and a number of daily file dumps are provided of the data.

GDS already consumes one of these file dumps, the Bank Unit file. This is an XML output that contains most (if not all) of the attributes. LSDB obtains this data from the Bank Unit module.

The primary use of this data is to derive the ownership structure of UBS Legal Entities. This is in turn used in transaction classification, to determine whether a trade is:

* Third party (UBS Contracting Entity with an external company)
* Intra-company (Two UBS Contracting Entities sharing a common parent)
* Inter-company (Two UBS Contracting Entities with different parents)

# Requirements

In order to provide transaction classification to Murex, MasterFiles needs to determine whether two UBS Contracting Entities have an intra- or inter-company relationship.

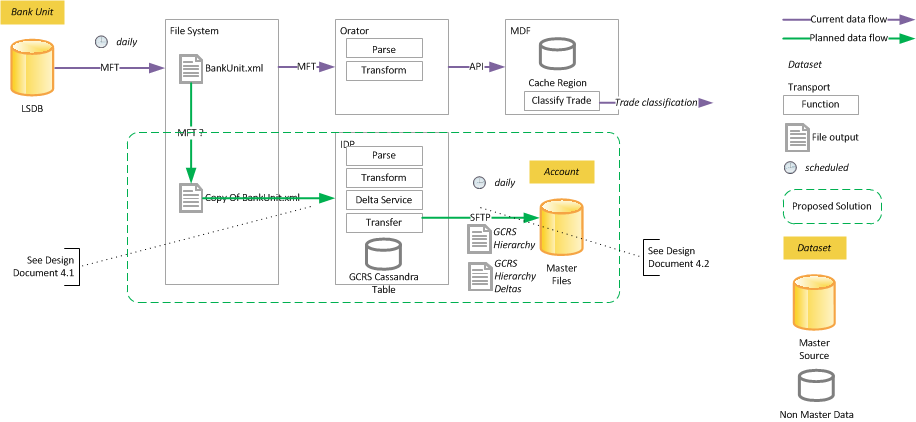
Rather than copy this data in to MasterFiles and other systems that require it, GDS will hold this data in a central aggregation point, namely the IDP.

The IDP will provide this data on a daily basis to MasterFiles in the form of a flat file transfer.

In addition, the IDP will store additional attributes on UBS entities obtained from the Bank Unit file.

The IDP **will not derive data** from the Bank Unit file. For example, if a UBS entity is part-owned by two others it will not assert an immediate parent of the entity that has the greater shareholding. Instead it will just supply the raw ownership percentages and it is up to the caller to derive. This keeps the dataset "neutral" and also gives it the greatest opportunity to be reused (as other clients may not wish to have this derivation).

# Data Flows



## Current Flows

* Daily extract of Bank Unit data as XML sent via Managed File Transfer (MFT)
* Orator receives notification (via MFT) and parses and transforms
* Transformed Bank Unit data published in to Master Data Fabric (MDF)

## Proposed Flows

* Orator process remains as-is
* On arrival of Bank Unit file a copy is made and moved to IDP location
  + Options:
    - Can the existing MFT job be used to provide this notification and movement?
      * Guaranteed delivery
      * Guaranteed file complete
    - Scheduled script that copies this file
      * Simpler
      * Faster time-to-market
* IDP listens for incoming files through a Spring XD process
* IDP transforms this files in to storage-ready shapes
  + see Data Model section below
* IDP stores data in Cassandra
* IDP prepares output for MasterFiles
  + see Data Model section below
* IDP sends file to MasterFiles
  + DECISION: Use SFTP rather than MFT as both IDP and MasterFiles are GDS-owned components supported by the same Production Services group. Hence extra guarantees and notifications not required so simpler solution (of SFTP) preferred.

# IDP Orchestrations and Components

In order to reduce binding between flows, and to make sure flows constitute an atomic unit of work, the data flow should be implemented as two separate orchestrations:

* Source Bank Unit data
* Send Bank Unit data to MasterFiles

## Source Bank Unit Data

* DECISION: Spring Batch job that listens for files and then runs through steps to parse, transform, and store
  + Bank Unit file is a monolithic dump of data and hence doesn't necessarily lend itself to stream-type behavior
  + The ingestion process is sequential in nature
  + The file is delivered once a day
    - Hence suited for scheduled processing (or listening)

### File Connector

* File Listener waiting for the Copy of the Bank Unit.xml to be delivered

### Transformer

* DECISION: SAX (or StAX?) Parser to ingest the file as it is large[[1]](#footnote-1)
  + Less memory-intensive so makes for a better IDP citizen
    - There are likely to be a number of data feeds that IDP receives at the start of a business day so each of them should be designed to be as light as possible on memory
  + More complex to prepare data
    - OPTION:
      * Either build up smaller trees by hand in memory
      * Store data as retrieved and then look up as needed[[2]](#footnote-2)
* The transformer should emit log entries with metrics information
  + Rows processed
  + Time to process
  + Number of exceptions

### ~~Delta Service~~

* ~~Compares data in Cassandra table with the latest data and determines whether it is different or not~~
* ~~If it is will update the data and the transaction timestamp~~
* ~~Will use this to supply the delta file to MasterFiles~~
* **THIS IS NOT REQUIRED IN THE INITIAL RELEASE**
  + But kept for reference

### Cassandra Proxy

* Reuse existing Cassandra classes to store data
* TODO: Determine commit interval
* TODO: Determine exception processing
  + E.g. Does all data fail if one Bank Unit insert fails?

## Send Bank Unit Data

* Sends ~~two files~~:
  + Full extract containing all GCRS
  + ~~Delta extract containing only data that has changed today~~ 
    - **THIS IS NOT REQUIRED IN THE INITIAL RELEASE**
      * But is kept for reference
* OPTION: Batch or Stream job
  + Could treat the data retrieval as a source of data that is sent after each row to a formatter

### Cassandra Proxy

* Performs a full-table scan on the *GCRS\_Hierarchy* Cassandra table
  + Not ideal for Cassandra but desk benchmarks show reasonable performance[[3]](#footnote-3)
  + Should emit a log entry with query performance details
    - Rows
    - Elapsed
* Iterates over each row
  + Each row from the *GCRS\_Hierarchy* will correspond to a line in the output file

### File Formatter

* Receives rows from the Cassandra Proxy step
* Formats the data with the appropriate separators, enclosing characters, etc.
  + TODO: Agree this format with the MasterFiles team
  + Should emit a log entry with file formatting details

### File Transport

* Receives notification of a completed file
* Sends the file via SFTP
  + On Exception
    - Logs exceptions in the appropriate format to trigger Netcool alerts
    - Log message must include instructions on how to remediate
      * E.g. Where the prepared file is
      * Where the prepared file should be copied to
  + On Success
    - Log success output
    - Remove previous day's file[[4]](#footnote-4)

# Data Models

## Incoming Bank Unit XSD

This schema is two years old so might be out of data, but contains the key information needed.



## Attribute Mapping

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute | LSDB.xml XPath | Comments | Column (in gcrs\_hierarchy) | Position in Flat File |
| GCRS Code | //Companies/Company/GCRS/GCRS\_id/text() |  | gcrsCode | 1 |
| Immediate Parent GCRS Code | //Companies/Company/Holdings/Holding[Type/@id='1']/@id | Derived.  Obtain LSDB Id from XPath and then look up record | immediateParentGcrsCode | 3 |
| Name | //Companies/Company/Names/Name[@actual='true' and Type/@id='6']/Content/text() |  | name | Not req'd |
| Multiple Parents Flag | //Companies/Company/Holdings[count(Holding[Type/@id='1'])>1] | Denormalise to make easier for MF processing.  Can derive through the count of holdings | multipleParents | 5 |
| LSDB Id | //Companies/Company/@id |  | lsdbId | Not req'd |
| Sub-category | //Companies/Company/General/Entity\_type/@code |  | subCategory | 2 |
| immediateParentName | See above | Derived.  Obtain LSDB Id from XPath and then look up record | immediateParentName | Not req'd |
| immediateParentLsdbId | See above | Derived.  Obtain LSDB Id from XPath and then look up record | immediateParentLsdbId | Not req'd |
| immediateParentSubCategory | See above | Derived.  Obtain LSDB Id from XPath and then look up record | immediateParentSubCategory | Not req'd |
| immediateParentOwnershipPercentage | //Companies/Company/Holdings/Holding[Type/@id='1']/Percentage/text() | Derived.  Obtain LSDB Id from XPath and then look up record | immediateParentOwnershipPercentage | Not req'd |
| legalParentGcrsCode | See above | Derived.  Walk tree based on parentage rules (see below) | legalParentGcrsCode | 4 |
| legalParentName | See above | Derived.  Walk tree based on parentage rules (see below) | legalParentName | Not req'd |
| legalParentLsdbId | See above | Derived.  Walk tree based on parentage rules (see below) | legalParentLsdbId | Not req'd |
| legalParentSubCategory | See above | Derived.  Walk tree based on parentage rules (see below) | legalParentSubCategory | Not req'd |

See rules for deriving immediate parentage in first mail and clarifications in deriving legal parent in second mail



## Physical Model

This will be held in a single table in Cassandra.

Note: the true GCRS hierarchy is probably better held as a graph and hence in Neo4j. This should be considered as part of a future release.

### GCRS\_Hierarchy

* Intention is to make creation of the flat file for MasterFiles as easy as possible
  + Hence table row and file row are pretty much identical
  + Provide a denormalized row that holds the Parent GCRS array that MF requires
    - When there is no row this value should be set to [] (an empty array)
* Table is keyed by:
  + GCRS Code
* A GCRS can have multi-valued attributes
  + E.g. Immediate parents (and their attributes)
  + These stored as Cassandra Maps
    - All keyed by the parent's GCRS code
    - Parent LSDB Ids
    - Parent GCRS Names
    - Parent Ownership Percentages
    - Parent Sub-categories
* Attributes for the Immediate and Legal Parent are denormalised
  + As these attributes (name, ownership percentage) will be queried along with the parent GCRS codes
  + To eliminate the need for joins (as Cassandra isn't join-optimised)

|  |  |  |
| --- | --- | --- |
| Column | Description | Datatype |
| gcrsCode | A unique string identifying a UBS Legal Entity that is also present in the GCRS Hierarchy.[[5]](#footnote-5)  The first part of the primary key | Text |
| mfParentGcrsCodes | A custom structure for MasterFiles consumption that gives the GCRS codes of the parents.  This is an array separated by colons, e.g.  [9090:7002] – two parents  [9090] – one parent  [] – no parents | Text |
| name | The name of the UBS Legal Entity | Text |
| multipleParents | Denormalization to indicate whether the GCRS has multiple parents | Boolean |
| lsdbId | The LSDB primary key of the UBS Legal Entity.  This could be indexed to allow lookups. | Text |
| subcategory | A code indicating the type of UBS Legal Entity this is | Text |
| immediateParentNames | A map of names of the immediate parent keyed by the GCRS code of the immediate parent. | Map<Text,Text> |
| immediateParentLsdbIds | A map of LSDB Ids of the immediate parent keyed by the GCRS code of the immediate parent. | Map<Text,Text> |
| immediateParentOwnershipPercentages | A map of ownership percentages of the immediate parent keyed by the GCRS code of the immediate parent.  This is defined as a decimal in the source  If there is no Immediate Parent then this value should be defaulted to -1 | Map<Text,Decimal> |
| immediateParentSubCategories | A map of sub-categories of the immediate parent keyed by the GCRS code of the immediate parent. | Map<Text,Text> |
| legalParentGcrsCode | The GCRS code of the Legal parent  If there is no Legal Parent then this value should be defaulted to NONE | Text |
| legalParentName | The name of the legal parent  If there is no Legal Parent then this value should be defaulted to NONE | Text |
| legalParentLsdbId | The LSDB Id of the immediate parent  If there is no Legal Parent then this value should be defaulted to NONE | Text |
| legalParentSubCategory | The subCategory of the Legal Parent  If there is no Legal Parent then this value should be defaulted to NONE | Text |
| transactionTimestamp | The timestamp when we last inserted or updated this data stored in ms since Jan 1 1970 | Bigint |

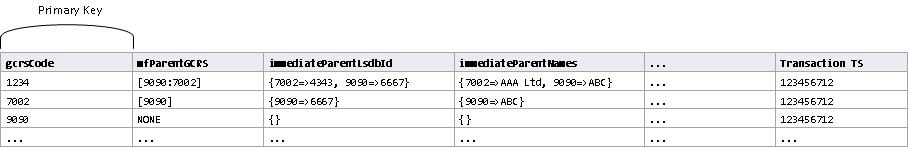


Figure 1: Example data for hierarchy table (not all columns shown)

The CQL to create this table is below:

CREATE TABLE GCRS\_HIERARCHY

(

gcrsCode text,

mfParentGcrs text,

name text,

multipleParents text,

lsdbId text,

subCategory text,

immediateParentLsdbIds map<text,text>,

immediateParentNames map<text,text>,

immediateParentOwnershipPercentages map<text,decimal>,

immediateParentSubCategories map<text,text>,

legalParentGcrsCode text,

legalParentName text,

legalParentLsdbId text,

legalParentSubCategory text,

transactionTimestamp text,

primary key (gcrsCode)

);

## Outgoing File Format

* Comma-separated file
* Each row will be made up of:
  + GCRS Code
  + SubCategory Code
  + "Array" of Immediate Parent GCRS Codes
  + Legal ("Ultimate"[[6]](#footnote-6)) Parent GCRS Code
  + Multiple Immediate Parents Flag
* Output will contain a single row for one GCRS Code
  + In the case of multiple or single parentage
* Output will be ordered by GCRS Code
  + Low to High
  + Note this is a text value so will be alphabetical low to high

### Example

GCRS Code, Subcategory, Immediate Parent, Legal Parent, Multiple Parents

1234, BR, [9977:5533], 4444, Y

1235, BR, [9997], 4444, N

GCRS 1234 has two Immediate Parents (9977 and 5533). These are separated by a colon in the array structure.

GCRS 1235 has only one parent.

# Appendix – External References

Orator currently consumes the LSDB XML file and creates a Lingua message based on it.

Most of the parsing code is a combination of JAXB and the UBSLEMapperV5 class, a copy of which can be found at:

<https://github.ldn.swissbank.com/distribution/idp-old/blob/master/idp-data-loader/orator2/pub-sub-ubs-legal-entity/src/main/java/com/ubs/datait/dist/orator/legalentity/pubsub/mapper/UBSLEMapperV5.java>

A copy is also in the zip file below



An example Lingua message for UBS Legal Entity can be found on the Lingua portal at <http://xldn2354dww.ldn.swissbank.com:8050/lingua2/protocols/#protocol::legalentity::5.2.0.0::examples> (**choose UBSLegalEntity-3918.xml** in the drop-down list)

For real-life LSDB data type goto/lsdb in to a browser.

From the menu bar on the page, choose **Report > Company** and then **GCRS** (to get general GCRS data) and **Relations** (to get holdings information)

1. Around 50 – 60 MB for each file [↑](#footnote-ref-1)
2. Will be able to determine which way once BankUnit.xml data fully analysed – i.e. to see whether there's any need to look up data that has already been read [↑](#footnote-ref-2)
3. 330,000 rows in 2 mins on a single-node cluster. Bank Unit data 2 orders of magnitude smaller than this [↑](#footnote-ref-3)
4. Or previous week's file for example, depending on how long we want to retain our prepared data. As this is a daily load it seems unnecessary to hold previous days' files. [↑](#footnote-ref-4)
5. We are only interested in UBS Legal Entities that are in the GCRS Hierarchy [↑](#footnote-ref-5)
6. This isn't actually the ultimate parent – see explanation in Section 5.2 (Attribute Mapping) [↑](#footnote-ref-6)