Rabobank – HCP Ingestion Engine

Author: Clifford Grimm

Date: 10/28/2015 3:13 PM

Revision: 2.2

Table of Contents

Revision History

Date	Author	Revision	Description
10/28/2015	Clifford Grimm	2.2	Final Edits for Algomi based on review with Rabobank and SearchTech.
10/27/2015	Clifford Grimm	2.1	Tweaks to Algomi metadata based on ongoing discussions.
10/13/2015	Clifford Grimm	2.0	Adding Algomi data set. Also, replaced the figures with folder structures to have the right date/time folder naming.
9/17/2015	Clifford Grimm	1.0	Post implementation pass.
4/16/2015	Clifford Grimm	0.85	Change for inserting a year folder into the folder structure stored on HCP.
4/15/2015	Clifford Grimm	0.8	Flushing out more information to reflect actual implementation. Lots of changes to Verint transformation.
3/25/2015	Clifford Grimm	0.7	Bringing in line with actual implementation. Reuters transform structure changed a little bit. Verint added the region to the input folder name and various placed where updated to reflect this.
3/19/2015	Clifford Grimm	0.6	Substantial clarity on all data sets. However, there are still a few unanswered questions.
3/4/2015	Clifford Grimm	0.5	Flushed out more detail in the Maintenance Monitoring and Runtime Environment sections.
3/3/2015	Clifford Grimm	0.4	Added sections around status reporting and error handling and other refinements.
2/27/2015	Clifford Grimm	0.3	Major revamp. More information coming in.
2/17/2015	Clifford Grimm	0.2	Added some bullet information about content validation.
2/11/2015	Clifford Grimm	0.1	Initial Revision

1 Introduction

Rabobank is a customer of the Hitachi Data Discovery Suite (HDDS) and the Hitachi Content Platform (HCP). These platforms are part of a solution to provide a compliance application with an intuitive search interface focused on locating various communication data based on a comprehensive but specialized criteria.

The data is searchable by this solution consists of:

1) E-mail
2) Reuters Instant Messaging
3) Bloomberg Messages
    a. IB Compliance (Instant Bloomberg)
    b. Message Compliance (Bloomberg Messenger)
4) Verint call recordings
5) Algomi fixed income messaging

The overall solution consists of two parts:

1) Data Ingestion Process Service consisting of custom software that will ingest the Reuters Bloomberg, Verint, and Algomi data into HCP in a form that is consumable by HDDS.
2) Search Interface Service custom built search interface with criteria specification as required by Rabobank.

This document focuses on the data ingestion portion of the solution and will only ingest the Reuters, Bloomberg, Verint, and Algomi content.

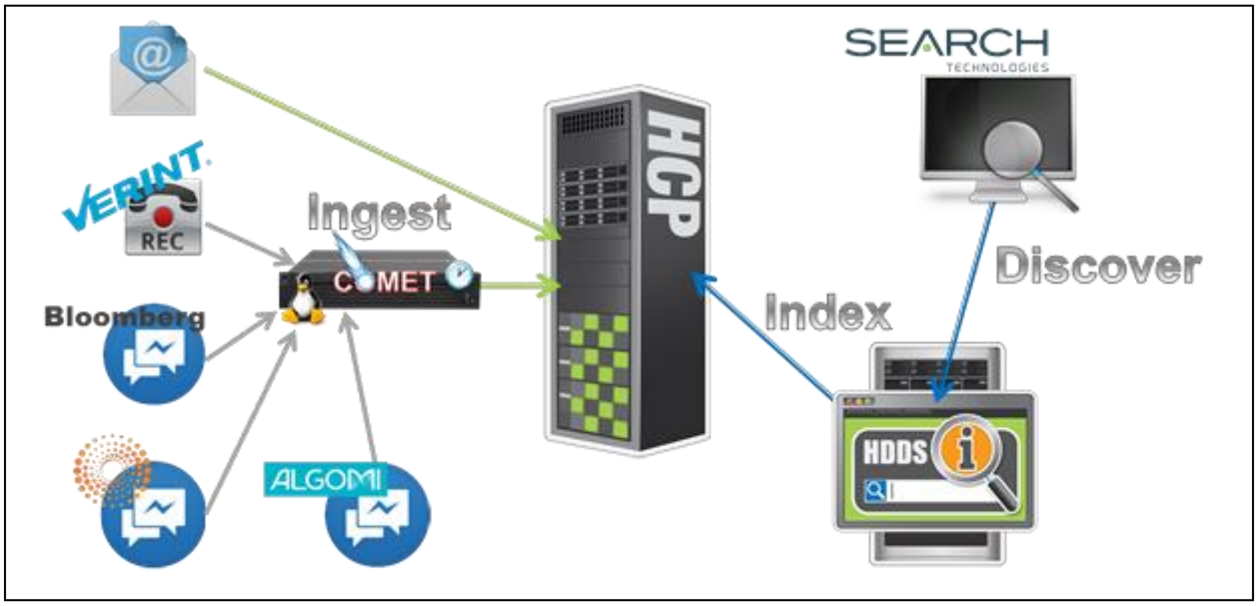


Figure 1: Solution Overview

1.1 Terminology

Term	Definition
Custom Metadata	An HCP concept where detailed information can be attached to HCP objects to be used for providing further value to the HCP objects. This custom metadata is usually in the form of XML to facilitate indexing.
Data Set	Refers to a type of data that is provided to this solution. The types discussed in this document are `Bloomberg`, `Reuters`, `Verint`, and `Algomi`.
Data Set Collection	A Data Set specific group of files/objects for a period of period as designated by the Data Set gathering mechanism. Currently, every data set collection is performed every 24 hours, thus consists of 24 hours of data.
Data Set Item	Refers to the most basic item that is ingested for any given data set. A data set item could be a single message, chat/conversation session, or a call recording wav file.
Quarantine Area	The file system folder structure that is used to place content that cannot be processed due to various reasons. The content stored in this area needs to be manually evaluated for the cause of the failure and manual corrective actions taken.
Raw Data Set	The original form of the content for a given Data Set as provided by external processes for processing by this solution. (a.k.a. `RawData`)
Region Mapping	The action of taking a region code extracted from the Data Sets and mapping it to an HCP Namespace that is to receive the content.
Staging Area	The file system location that Raw Data Set is placed by the external systems and is used as the raw input for this solution.
Transformed Data Set	This solution receives the Raw Data Set and performs various transformation of the content to facilitate usage by HDDS and the Search Console part of the solution. (a.k.a `TransformData`)
Work Area	The file system folder structure that is used as a temporary storage area for assembling and transforming content in preparation for ingestion into HCP. (a.k.a. `WorkArea`)

1.2 Requirements

The core requirements for the ingestion solution are the following:

- Ability to processes multiple data sets of differing layout received by different geographic regions.
- Ability to ingest regional content into separate HCP namespaces to provide secure separation of content between regions.
- Ability to configure an execution schedule with granularity down to the different data sets.
- Ability to tune throughput via parallel ingestion and/or sleep timers to manage highest throughput with acceptable compute/network resource consumption.
    - Parallel ingestion may be obtained through either single process execution or multiple process execution.
- Ability to monitor execution for the purposes of throughput and ingestion failure detection.
- Ability to resubmit failed ingestions for second processing.
- Perform comprehensive data validation of each data set before ingesting into HCP, as once ingested in HCP it can’t be removed.
- Record ingestion processing status into search console database for viewing and reporting.
    - Status will include success/failure statistics and failures will contain hints as to the type of failure.

2 Solution Overview

The ingestion portion of the solution is responsible for accepting the different data set types and ingesting them into the HCP system in a manner that will provide for robust search by the HDDS system. The current types of data set types processed are Reuters, Bloomberg, Verint, and Algomi.

The data sets are written onto a file system of the ingestion system by a mechanism provided by Rabobank. The data sets are organized by the data set type and further will be organized in a manner that will identify the geographic region for which they belong.

The ingestion engine detects the existence of the data sets and process them as appropriate for each data set. This processing may consist of transforming the data prior to ingestion either by reorganizing or structuring in a manner that facilitates ingestion into HCP.

During content ingestion into HCP, the content is organized into namespaces appropriate for that data and the region for which the data is for. In addition, when the content is ingested into HCP, it may be further transformed into appropriate folder structure and/or generate custom metadata attached to the object. Folder structure is important to adhere to HCP best practices around folder and object count. Custom metadata added to the objects will facilitate the indexing and search capabilities of the HDDS product.

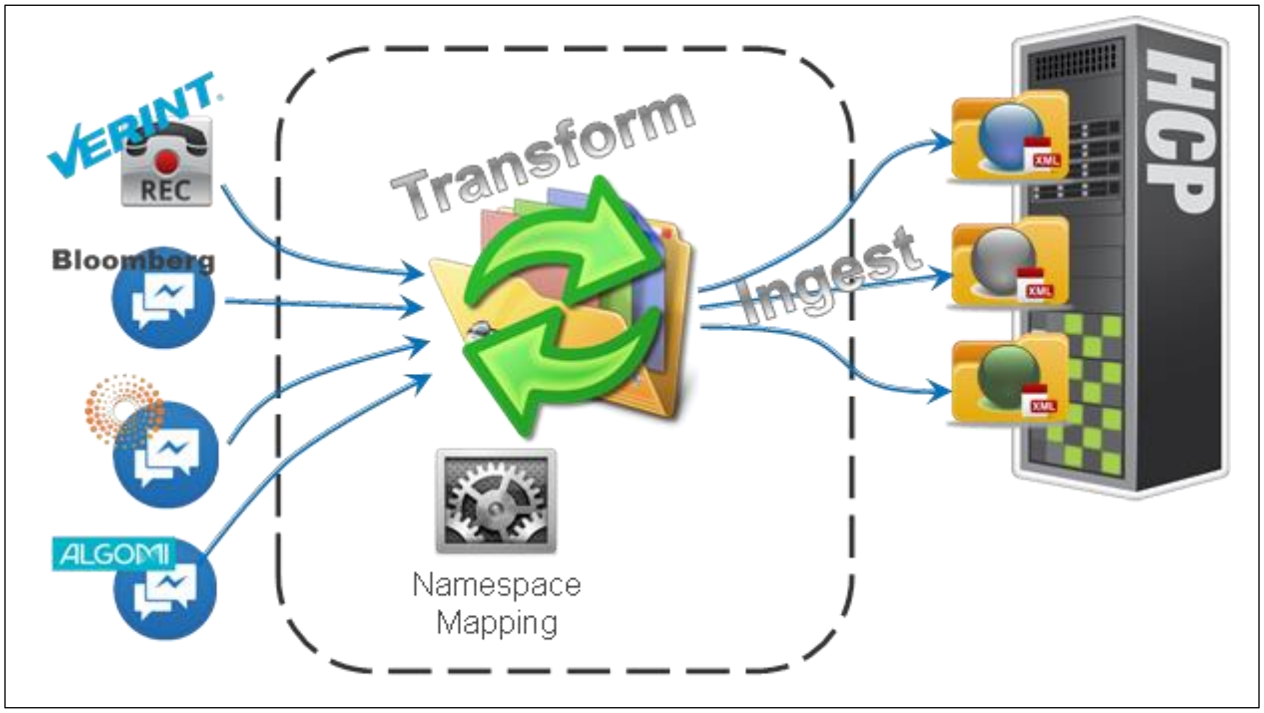


Figure 2: Ingest/Transform Overview

3 Data Set Processing

This section describes the main processing phases for the data sets. Each data set goes through multiple phases to reach the point of content being ingested into HCP in an index-able form. The general phases of processing are the following:

Phase Name	Description
Raw Data Set Capture	Evaluate content in drop zone for complete data set collections. When found, move data set into work area. There is a basic data layout transformation to adjust for Rabobank file transfer mechanism to recreate original raw data set layout.
Data Set Validation	Data Set is validated to ensure it is complete and intact.
Data Set Transformation	Data Set is transformed as needed to produce ingest-able and index-able content form. Some metadata may be prepared during this phase.
Data Set Ingestion	Raw and/or Transformed content is ingested into HCP. Applicable objects will be ingested and custom metadata constructed and attached.
Cleanup and Failure Reporting	Work area is evaluated for failure conditions and cleaned up for next processing cycle.

Table 1: Data Set Processing Phases

The following subsections will describe:

- The form the Input Data Set presented to the ingestion solution as dictated by Rabobank.
- The Input Data Validation performed to ensure only valid data is processed.
- The transformation of the Input Data Set to prepare it for ingestion by HCP and indexing by HDDS.

3.1 Raw Data Set Capture

Rabobank is responsible for defining and presenting the data sources from multiple geography regions onto a file system accessible by the ingest solution host OS.

The main data sources this solution supports are the following:

- Reuters Instant Messages (IM)
- Bloomberg Messages
    - IB Compliant (Instant Bloomberg)
    - Message Compliant (Bloomberg Messenger)
- Verint call recordings.
- Algomi fixed income messages

On the file system, there is a folder structure consisting of a set of top level folders rooted at `/appl/ingest/HCP_Production`, one for each major data set, i.e. Reuters, Bloomberg, Verint, and Algomi. In each of these folders, external processes present the content to be ingested in various forms to be described later.

During this phase of processing, the content is collected from this file system structure and staged in a work area for further processing. For content to be collected, it must meet a certain criteria of files to be a candidate for collection. Incomplete collections are left in the file system structure until such a time all expected files have arrived, or a collection of files remain for a configurable amount of time and thus considered “stale”. Any stale files will be moved from this input file system structure to a quarantine area for further evaluation.

The following sections will describe the files for each data set type.

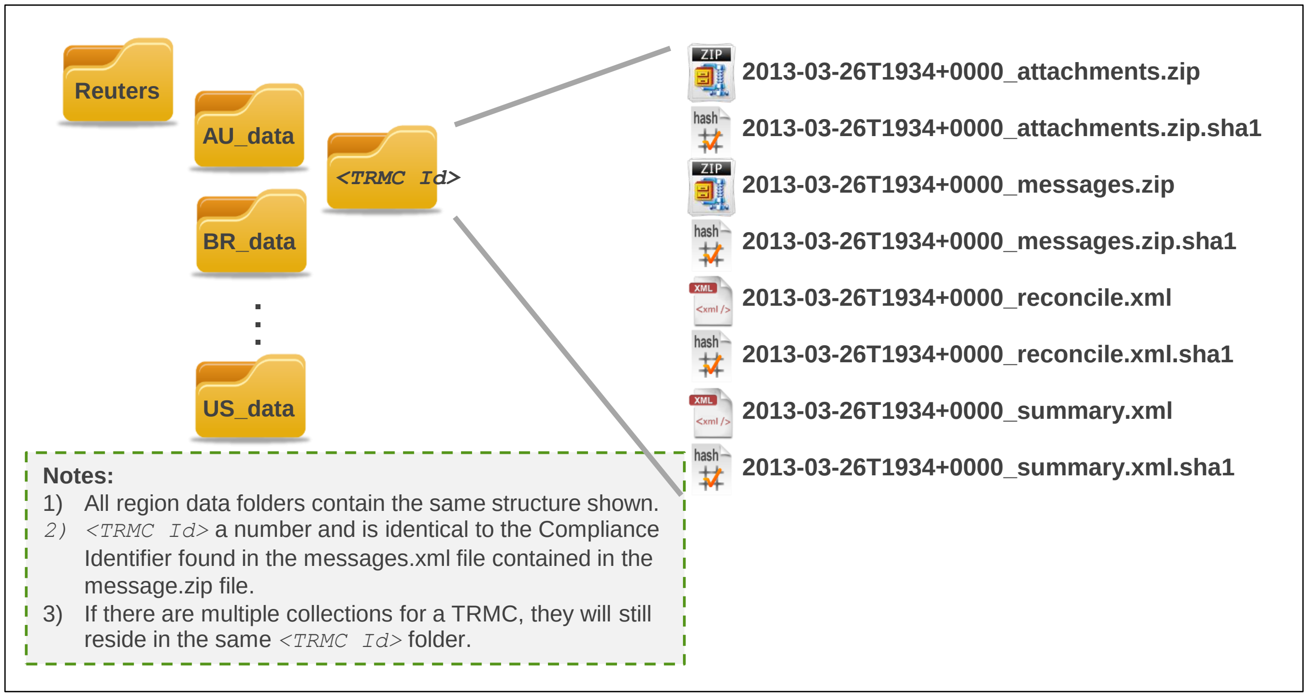
3.1.1 Reuters IM

The Reuters Instant Message (IM) External Feeds is a service provided by Thomson Reuters that collects all IM interactions utilizing their messaging service. These collections are packaged daily and made available for download via FTP by the clients. This service has a very well documented file/folder layout as well as internal field definitions that can be obtained from Thomson Reuters called:

Thomson Reuters Messenger Compliance External Feeds User Guide v1.1

Rabobank is responsible for providing the processing necessary to perform periodic download of the daily collections for all geographic regions and presenting them onto the ingestion solution host OS.

The folder structure consists of a top level of folders that will define the region for the call collections. Under each region data folder will be folder with the name of the Compliance Identifier, *<TRMC Id>*, for which this collection belongs. The Compliance ID folder will contain one or more daily collections. Collections consist of groups of files with the same starting date/time string. The date/time string has the convention of convention *yyyy-MM-ddTHHmmZ*; for example 2013-06-26T1730+0000. The picture below shows the set of files for one daily collection. A daily collection(s) in this folder is a group of files consisting of xml, checksum, and zip files.


Reuters
AU_data
BR_data
US_data
<TRMC Id>
2013-03-26T1934+0000_attachments.zip
2013-03-26T1934+0000_attachments.zip.sha1
2013-03-26T1934+0000_messages.zip
2013-03-26T1934+0000_messages.zip.sha1
2013-03-26T1934+0000_reconcile.xml
2013-03-26T1934+0000_reconcile.xml.sha1
2013-03-26T1934+0000_summary.xml
2013-03-26T1934+0000_summary.xml.sha1

Notes:
1) All region data folders contain the same structure shown.
2) *<TRMC Id>* a number and is identical to the Compliance Identifier found in the messages.xml file contained in the message.zip file.
3) If there are multiple collections for a TRMC, they will still reside in the same *<TRMC Id>* folder.

Figure 3: Reuters Input Folder/File Layout Sample

The following table contains a brief description of the files:

File Name	Description
***_summary.xml**	Summary of Messenger activity over the period
***_summary.xml.sha1**	Checksum for *summary.xml*
***_messages.zip**	All messages exchanged between Messenger users during this period
***_messages.zip.sha1**	Checksum for *messages.zip*
***_attachments.zip**	All attachments exchanged between Messenger users during this period
***_attachments.zip.sha1**	Checksum for *attachments.zip*
***_reconcile.xml**	Reconciliation file for Messenger activity over the period
***_reconcile.xml.sha1**	Checksum for *reconcile.xml*

Table 2: Reuters IM Collection Content

For any given daily collection to be considered complete, there must be a minimum of the `*_summary.*` and `*_reconcile.*` files. If there are messages, both `*_messages.*` files must exist. If there are attachments associated with the messages, both `*_messages.*` and `*_attachments.*` files must exist.

3.1.2 Bloomberg

The Bloomberg collection of files presented to the data ingestion system contains no folder structure and the only distinguishing factor are the names of the individual files. Of these files, there is a daily manifest file that contains the list of all the individual files for the collection. The daily manifest is of two forms: a text file and an html file. The names for the files are `daily_manifest_*.txt` and `daily_manifest_wget_*.html`. Complete manifest files have a number at the end of the name, the number indicates the Collection ID for which the value is the date for the collection. These collections are typically finalized between 3PM EST and 3:30PM EST each day. For the collection on finalized during this period on February 25th, 2015, the manifest file name will be:

```
daily_manifest_150225.txt
```

The download folder may also have a manifest file that contains the word `current` instead of a date. This is the currently collection being built for the current day. This manifest is not considered complete and will not be processed.

The individual files that make up the collection have a file name format of:

```
<Account-Number>.<Content-Type>.<Collection-ID>.<File-Type>
```

where each field is described in the following table:

Field Name	Description
<Account-Number>	Bloomberg assigned account number for the file content. Each account can be mapped to a specific deployment. This mapping of the accounts to regions will need to be obtained from the account owner (i.e. Rabobank). For Bloomberg, there is only one account, f3616, collected and represents the Rabobank firm number 3616.
<Content-Type>	Multi-letter code that indicates the type of content in the file. There are 4 known content types: • `ib` – Instant Bloomberg (Chat rooms) • `msg` – Bloomberg Messenger (IM messages) • `dscl` – Disclaimer collections • `att` – Attachments
<Collection-ID>	Collection ID for which this file belongs. The collection ID is the date for when the collection was finalized in the form of `YYMMdd`.
<File-Type>	Traditional file type of the file indicating the form of the contents of the file. For instance: • xml – eXtensible Markup Language • tar.gz – Tar GnuZip compressed

For each daily collection, there will only be one set of files. That is, all worldwide communication for Bloomberg will be recorded into one collection and tagged with an Account Number that represents the Rabobank firm number of 3616. This point is important to understand since any given collection must be broken up into message interactions that will not allow any geographic region to see messages for which nobody in their region participated.

Each daily collection consists of a daily manifest file and the contents of the file is the list of companion files that exist. The manifest file is examined to obtain the list of expected files for a daily collection. To ensure that any daily collection is complete, the list in the manifest file is used to verify the file existence.

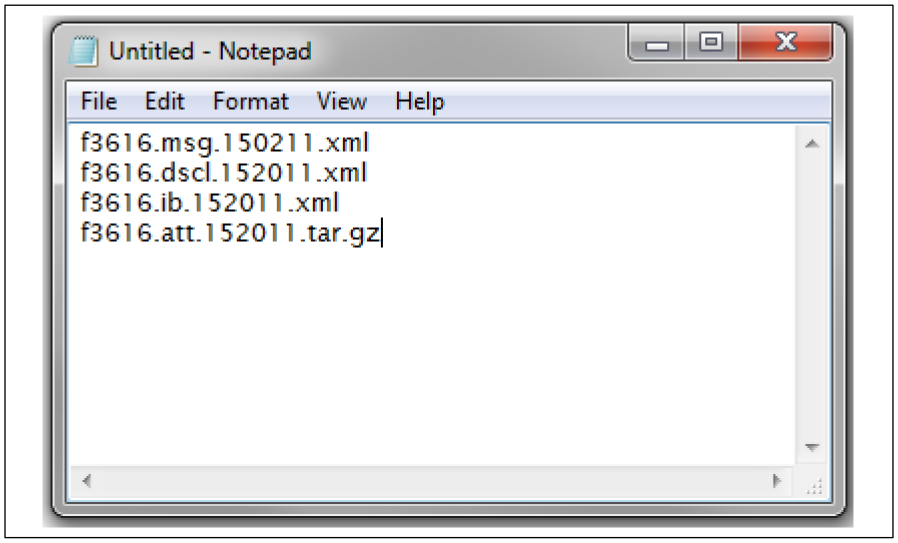
A daily collection may contain one of each of the files (ib, msg, dscl, and att) for the Rabobank firm number (3616). The detailed description of the `ib` and `msg` files can be found in the Bloomberg documents on these topics.

The `dscl` file is a simple enough format to understand, but the purpose of this file is to have all unique disclaimers that accompany a message to be collected in a separate file rather than having the disclaimers filling up the `msg` files. Instead the `dscl` file contains all the unique disclaimers and reference are written to the `msg` files, where applicable.

The `att` files are a compressed set of all file attachments that were part of the Instant Message and Instant Bloomberg conversations contained in the `msg` and `ib` files. Each file in the `att` file consists of the same name and file type as the original. The `msg` and `ib` files will have references to the file(s) contained in the `att` file.

3.1.2.1 Daily Manifest File Contents

For each daily collection, there is a manifest. The manifest is stired both in a text file and html file. The following is example content for the text based manifest file:



```
f3616.msg.150211.xml
f3616.dscl.152011.xml
f3616.ib.152011.xml
f3616.att.152011.tar.gz
```

Figure 4: Sample daily manifest text file

3.1.2.2 IB Compliant

The files with the `<Content-Type>` of `ib` contains the Instant Bloomberg (IB) dump of message conversations. The file is in XML form and contains all IB Conversations for chat rooms for the period. For the full data definition of the contents, see the PDF document titled:

Bloomberg LP - IB Compliance Dump Format v1.3. Dated March 18, 2008.

3.1.2.3 Message Compliant

The files with the <Content-Type> of `msg` contains the Bloomberg Messenger collection dump. The file is in XML form and contains all Bloomberg instant messages for that period. For the full data definition of the contents, see the PDF document titled:

Bloomberg LP - Message Compliance Dump Format v1.6. Dated March 18, 2008.

3.1.3 Verint Call Recordings

The Verint call recordings consists of WAV files and associated metadata that describes the call. Periodically, a process external to the ingestion process extracts these calls and package them up into a folder structure. The top level folder is a collection identifier. The naming of this folder is of the form:

`<Region-Code>.<Job-Name>;<Job-Id>_<Date-Time>`

where

`<Region-Code>` - the Rabobank standard two letter code that indicates the geographic region for the data collection.

`<Job-Name>` - Name assigned by administrator of the Verint Export Manager

`<Job-ID>` - Run number of the <Job-Name>.

`<Date-Time>` is of the form of *`YYYYMMdd_HHmmss`* and represents when the job was run.

In this folder, is a collection description file name `CD_Mappings.xml` and two folders named `WAV` and `IDX`.

The `WAV` folder contains all the individual call recordings in WAV format for the collection period. The name of the WAV file(s) are a unique identifier generated by the software that prepared the call recording collection.

The `IDX` folder contains XML files with the metadata for each call in the `WAV` folder. The mapping between the XML metadata file and the WAV file is the unique identifier that is the as the file name for both the XML and WAV file (less the file extension). The XML schema for the XML content is voice recorder specific information depending on the brand: Avaya, Etrali, Voxsmart, etc.

The WAV and IDX file names are constructed of two pieces of information. The first six digits make up the voice recorder serial number and can be used to map to a specific location; for example `702521` is the serial number for the London/Utrecht system. The following 9 digits make up a unique identifier for the call for a given voice recorder serial number.

The `CD_Mappings.xml` file contains customized metadata from the Verint Extraction Manager deployment.

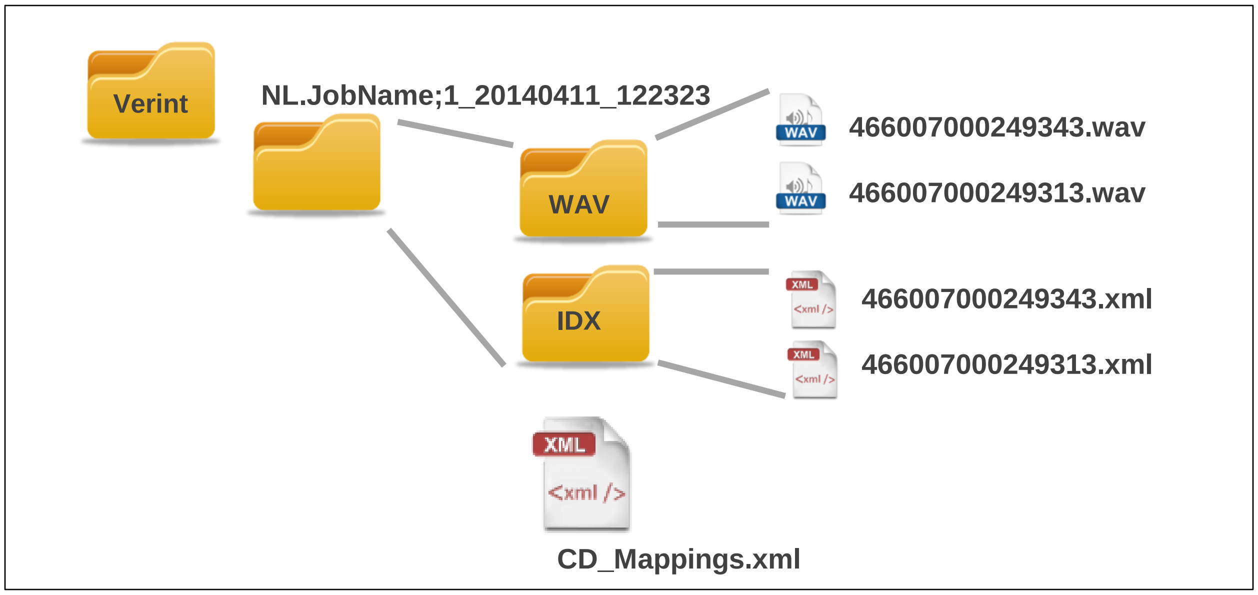


Figure 5: Verint Input Folder/File Layout Sample

Each folder is examined to ensure it is complete by looking to see if the `CD_Mappings.xml` file has been created. If it has this file, collection will be moved to the work area for processing. If any folders and its contents are older than a configurable threshold and does not have the `CD_Mappings.xml` file, it will be moved to a quarantine area.

3.1.4 Algomi

Algomi is a third party software application that constructs a fixed-income (bond) communication network. The Algomi product suite includes two types of messaging components:

- **Synchronicity** : Internal communication between fixed-income trading team members within Rabonbank.
- **Honeycomb** : External communication between Rabobank trading team members and fixed-income trading teams at other financial institutions.

Rabobank is building a software solution to extract these communications from the Algomi system for ingestion into this overall solution. Data collections will occur once a day and will be presented it to the ingestion system rooted at:

    /appl/ingest/HCP_Production/Algomi

The following figure shows a sample input data collection.

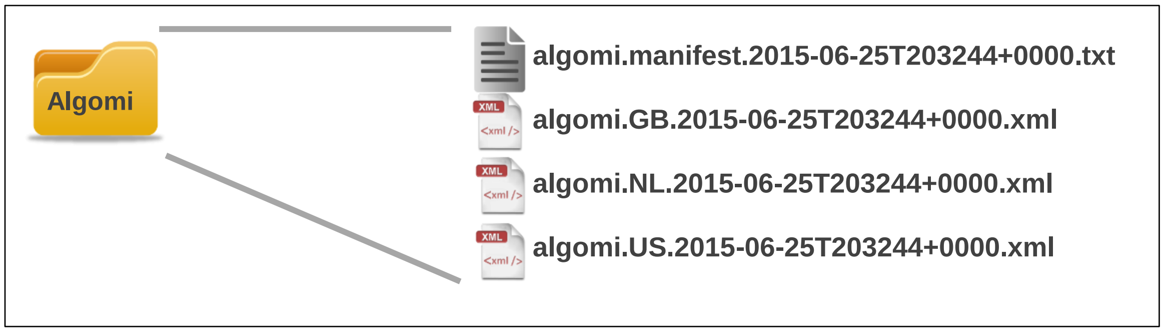

Algomi
algomi.manifest.2015-06-25T203244+0000.txt
algomi.GB.2015-06-25T203244+0000.xml
algomi.NL.2015-06-25T203244+0000.xml
algomi.US.2015-06-25T203244+0000.xml

Figure 6: Algomi Input Data Collection

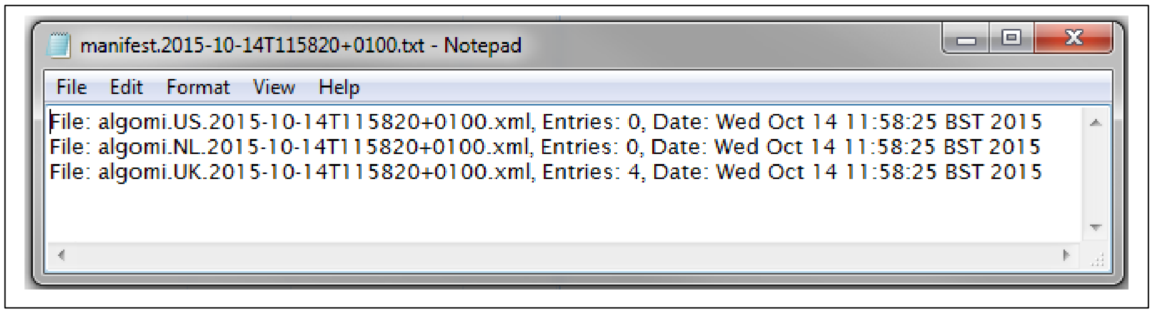
Each daily data collection consists of a group of files. The first file is the manifest file with the naming convention of:

`algomi.manifest.`*`yyyy-MM-ddTHHmmssZ`*`.txt`

, for example:

algomi.`manifest.2015-06-25T203244+0000.txt`

Within this manifest file is a list of the accompanying files that make up the daily data collection. The following is an example file manifest content:


manifest.2015-10-14T115820+0100.txt - Notepad
File Edit Format View Help

```
File: algomi.US.2015-10-14T115820+0100.xml, Entries: 0, Date: Wed Oct 14 11:58:25 BST 2015
File: algomi.NL.2015-10-14T115820+0100.xml, Entries: 0, Date: Wed Oct 14 11:58:25 BST 2015
File: algomi.UK.2015-10-14T115820+0100.xml, Entries: 4, Date: Wed Oct 14 11:58:25 BST 2015
```

Figure 7: Algomi Manifest File Contents

Each file listed contains messages formulated in XML form for which a user in the region participated or initiated.

3.2 Data Set Validation

Once content has been identified as available to be captured and moved to the work area, it is necessary to ensure the content is completely valid and usable before ingestion into HCP. Once validated for completeness and authenticity, the content is ingested into HCP.

There are two reasons for comprehensive validation checking of received content:

1) Legal regulators require there be proof that content has not changed from the original source and content is valid.
2) To ensure content is not removed, HCP will be set up with retention to ensure content cannot be removed. Once content is ingested, it needs to be valid.

The validation consists of both data source agnostic as well as data source specific validation. The two data source agnostic checks will be:

1) Compute a general hash of a complete collection for the data source content and lookup in a 7 day history to see is the same hash has already been seen. The details of how the hash will be computed will be different for each data source and described in the following sections.
2) Compare the HCP computed hash during ingestion for each piece of content to the ingestion tool computed hash for the content. If they do not match, the content will remain on HCP (as it cannot be removed), but the content will be flagged as a failure and will be moved to the Quarantine failure area for further inspection.

Aside from these two validation checks, there will be comprehensive validations for each type of data source. Any data source collections found to be invalid will be moved in its entirety into the Quarantine storage area defined in the configuration.

The following sections describe the data source specific validations performed.

3.2.1 Reuters

The Reuters IM content is delivered in the form of multiple files. These files are examined to ensure they have not been tampered with or was provided as corrupted from the source. The following is the list of checks that will be performed on the content before ingestion:

- Validate the data source provided hashes in the `*.sha1` files with their corresponding files by re-computing the hash from the original content. As part of this check, it ensures that a full set of hash files exist.
- The `messages.xml` file is extracted from the `messages.zip` file. This check will also ensure that the `messages.zip` file is a valid zip file. This `messages.xml` file is passed through an XML parser to ensure it is well formed XML.
- The Compliance ID found in the `messages.xml` file under the XML tag `/Headers/ComplianceSetup/ID` is compared to the parent folder identifying the `TRMC_ID` for the content. This ensures that every collection was placed into the correct compliance setup.
- Perform an unzip test on the `attachments.zip` to ensure it is a valid zip file.

- Verify the number of attachments in the `attachments.zip` file matches the count recorded in the `summary.xml` file.
- Verify the number of messages in the `messages.xml` matches the count recorded in the `summary.xml` file.
- Verify the `reconcile.xml` file contents with the `messages.xml` file content. The specific items validated are:
    - The number of expected chat sessions exist with the recorded chat room identifiers.
    - All chat sessions contain all the same message count and identifiers.
- Content collection has not previously been processed in the last 7 days. This will be accomplished by taking all the hashes available for individual content and computing a hash of those hashes. This hash will be used to verify this content has not been previously seen.

If any of the validation items fails, the raw content will be moved to a quarantine area for further investigation as to the cause.

3.2.2 Bloomberg

The Bloomberg content does not contain hashes generated by Bloomberg, quick hash validation is not an option. Therefore, the following operations are performed to ensure some level of confidence that the content is intact before ingestion into HCP:

- Verify that all files listed in the daily manifest exist, and there are no extra files.
- Each daily file collection contains no more than one `ib`, `msg`, `dscl`, and `att` file.
- Verify that the `att` file is a valid compressed tarball file by attempting content listing.
- The `ib, msg,` and `dscl` files are confirmed to be valid XML by performing some basic XML parsing on the file.
- Perform a cross reference to ensure that there exactly a one-to-one mapping between the attachment files and the references in the Bloomberg message content.
- Content collection has not previously been processed in the last 7 days. This will be accomplished computing a hash of the string consisting of the region, account, and collection date/time combined with the content of the `daily_manifest_*.txt` file. This hash will be used to verify this content has not been previously seen.

If any of these validation checks fail, the whole collection will be considered invalid and in its raw form will be moved to quarantine area for further evaluation.

3.2.3 Verint

The Verint data set is very simple in that it only consists of WAV files with accompanying metadata contained in a sibling XML file. There is no manifest or hashes to validate all the content exists. While it is possible to utilize modules to validate the WAV files by perhaps attempting extraction of the duration, there are likely to be too many WAV files per collection to make this feasible.

Therefore, the following is the validation performed on Verint collections:

- Generate of a list of the WAV files and the sibling XML files. The two lists are compared to ensure every file has a peer.
- Instead of taking the overhead of validating every sibling XML file for valid XML, during ingestion, the sibling XML file will be validated as valid XML by HCP during the ingestion process. If not

valid XML, neither the WAV file or the XML file will be ingested. This will be accomplished by using HCP Whole I/O feature and configuring the namespace to only accept valid XML content for custom metadata.

- Content collection has not previously been processed in the last 7 days. This will be accomplished computing a hash of the string consisting of the region, and collection date/time. This hash will be used to verify this content has not been previously seen.

With these behaviors any mismatched WAV and/or XML files, or WAV files with invalidly formatted XML for metadata will not be ingested in HCP. As part of the final processing of the ingestion process, this remaining content will be considered failed ingestions and moved to the Quarantine area for further manual evaluation.

3.2.4 Algomi

The Algomi data collection simply consists of a manifest file and companion files as listed in the manifest. The manifest contains information that is used for validation to ensure the data collection is fully in tact before ingesting the content into HCP.

If any validation checks fail, the full data collection will be moved to the Quarantine area for manual investigation as to the cause.

The following is a list of the validation checks to be performed against a data collection's content:

- For every file listed in the manifest file, the file will be checked for existence. If not all files exist, the data collection will not be collected. If the data collection fails to become complete for the configured threshold time, it will be moved to the Quarantine area.
- For each file listed, the following checks are performed against the content
    - The file contains valid XML.
    - The `Entries` count recorded in the manifest file matches the number of `ChatRecords` elements found in the file.
    - At least one of the `ChatMember` structure's `countryCode` field for each `ChatRecord` structure found in the file is the same as region code portion of the file name.

Once the data set collection is validated for completeness, there are a couple of additional checks that will be performed on the data sets. This will consist of detecting duplicate data collections and network transmission failures.

First, the data collection will be determined if the content has been seen previously, thus considered a duplicate. Duplicity will be determined by comparing a SHA-1 hash of the data collection with a list of previously processed checksums stored for the last configurable number of days. If there is a match, the data collection will be flagged as a duplicate and moved to the Quarantine area.

The Algomi data set does not have a supplied checksum to utilize, therefore the SHA-1 checksum will be generated based on the content by generating SHA-1 checksum of the SHA-1 checksum of all the files that exist for the data collection.

The second validation is when each individual objects (i.e. Algomi ChatRecord) is ingested in HCP. During the ingestion, a hash will be generated for the HCP object prior to ingestion. The object will then be ingested and part of the return status from HCP is a hash of the object as stored on HCP. If the computed and the returned hash do not match, the object will be left behind and considered a residual file failure. The file will be moved to the Quarantine area for manual investigation as to the cause of the failure, and perhaps re-introduced into the processing to retry the ingestion.

3.3 Data Set Transformation/Layout

This section discusses how the content will be stored in HCP for indexing by HDDS. The transformation will consist of namespace distribution as well as data transformation with metadata.

For namespace distribution, there is a set of HCP tenants and namespaces. This namespace structure is required to adhere to security requirements between different geographic and data set boundaries.

Tenants are created using two attributes for the tenant name. The general naming convention can be described as the following:

```
<System><Data-Type><Environment-Type>
```

All tenant names start with “HCP” string for `<System>`. The *`<Data-Type>`* is a three letter code of REU (Reuters), BLM (Bloomberg), and VER (Verint). The *`<Environment-Type>`* is a 3 digit code of 400 (Production) and 500 (Test).

Thus for Reuters messages for the production environment the tenant is named:

```
HCPREU400
```

Within a tenant contain namespaces with names fully identifying content contained within it. The general naming convention for namespaces can be described as the following:

```
<Data-Type><Region-Code><Legal-Entity><Environment-Type>
```

Some attributes from the containing tenant name will be repeated in the namespace name; for instance, *`<Data-Type>`* and *`<Environment-Type>`*. The namespace name will also contain additional identifying attributes of *`<Region-Code>`* and *`<Legal-Entity>`*. The *`<Region-Code>`* is a 2 alpha ISO country code. The *`<Legal-Entity>`* will be `RAB` for Rabobank, or 3 letter code for legal entity for the given country.

For example, namespaces in the `HCPREU400` tenant containing Netherlands (NL) content and for the Rabobank Netherlands (RAB) legal entity, the namespace name will be:

```
REUNLRAB400
```

For each data set, there is mapping configuration to indicate all valid `<Region-Code>`’s and the namespace content should be ingested. In addition, the HCP connection information can be specified for each namespace. Details of this configuration is discussed in the *Rabobank – HCP Ingestion Engine Administrative Guide*.

For a full list of tenants and their respective namespaces, see the Appendix A at the end of this document.

Within each of these namespaces, content will be transformed to facilitate indexing and/or more effective organization for retrieval. Each of the following sections will focus on each core data set to be ingested.

3.3.1 Reuters IM

The Reuters IM content requires both data transformation for effective indexing as well as storing the original content for regulatory purposes to provide the content has not been modified from the original.

Both these forms will be stored in a namespace with differing root folders. The unchanged content will be stored in the top level folder `RawData`. The transformed data will be stored in the top level folder `TransformData`.

The `RawData` layout is a slight variation of the original content as described in *Figure 3: Reuters Input Folder/File Layout*. The difference is the files will have the date/time stamp removed and be placed in a folder structure with the year used as the top level folder, and the remaining date/time stamp as the sub-folder name. The following shows an example layout.

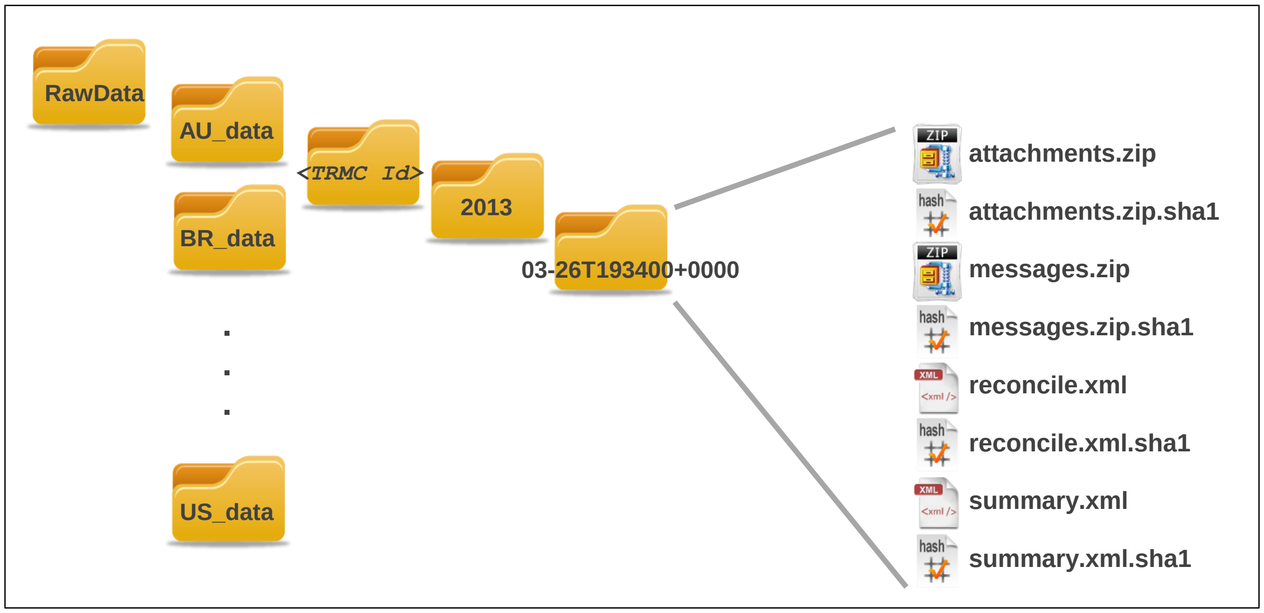


Figure 8: Reuters RawData Folder/File Layout

Transformation of the Reuters IM content is required to solve two basic problems with the content. First, the content is packaged in zip files and retrieving portions of the content will require unnecessary processing by the consumer of the content. Second, there is one IM message XML file that contains all the chat sessions for the collection period. This message XML file needs to be broken up into individual chat sessions to facilitate identifying chat sessions that fit the criteria instead of the whole message XML , which can be rather large. Each chat session has a naming convention of:

```
TRM_<Collection-Date><Index>.xml
```

where,

`<Collection-Date>` is the format of 2 digit year, 2 digit month, and 2 digit day for the date of the collection.
`<Index>` is an assigned daily collection unique number.

In the raw data form, there is an `attachments.zip` file that contains all the attachments for the chat sessions for the collection period. This zip file will be expanded and stored in an `attachments` folder.

In the raw data, there is a `messages.zip` file that contains the one `messages.xml` . Instead of just unzipping the `message.xml`, the file is further broken down into its top level XML tags into their own files. The XML tags extracted consists of 3 singleton tags: `Headers`, `Version`, and `Users`. In addition, the multiple `Chats` tags representing the chat sessions are extracted into individual files with unique names provided via combination of collection date and numbering as to the order found in the `messages.xml` file. These files are written to the `instant_messages` folder.

The following diagram outlines the layout that will be available on HCP:

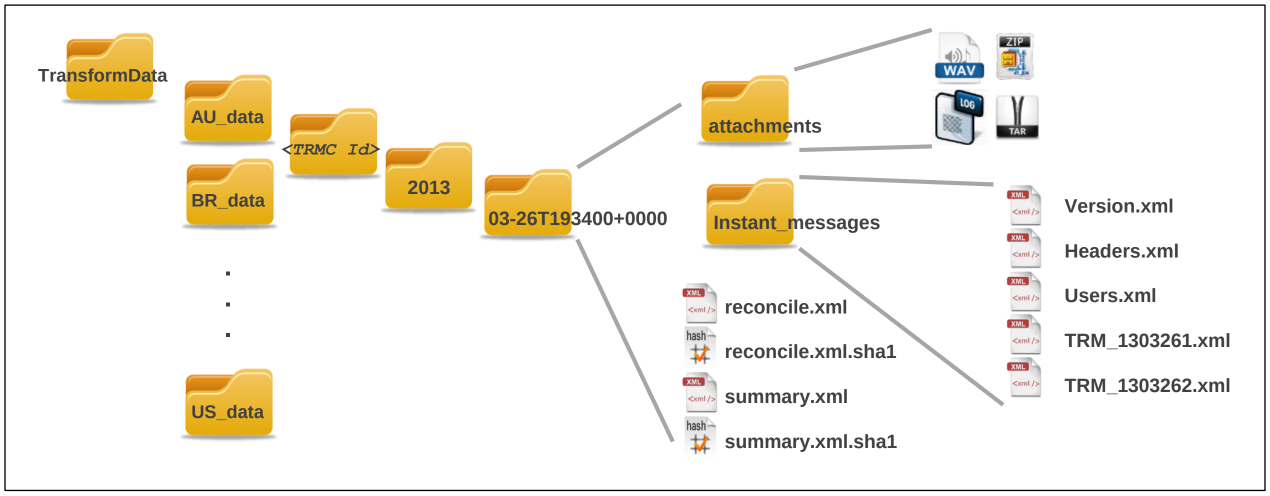


Figure 9: Reuters Transformed Layout

In addition, to the transformed data layout, each chat session ingested will have HCP custom metadata attached to facilitate searching for content based on the original messages headers. The custom metadata will be XML structure consisting of the some core collection information, `Headers`, `Versions`, and `UserInfo` tags from the original `messages.xml` file. The `UserInfo` information will contain only the user details for those users that participated in the chat session.

The following figure shows an example of the custom metadata to be attached to the chat sessions.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
  <CollectionInfo>
    <Region>NL</Region>
    <DateTime>2013-03-26T19:34:00+0000</DateTime>
    <Account>406811</Account>
  </CollectionInfo>
  <Version>1.0</Version>
  <Headers>
    <ComplianceSetup>
      <ID>406811</ID>
      <Name>SampleCompany1</Name>
    </ComplianceSetup>
    <DataPeriod>
      <From>2013-03-25T19:34:17.600Z</From>
      <To>2013-03-26T19:34:17.600Z</To>
    </DataPeriod>
  </Headers>
  <UserInfo>
    <Identifier>John@thomsonreuters.com</Identifier>
    <Email>John@thomsonreuters.com</Email>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
    <ComplianceSetup>
      <ID>406811</ID>
      <Name>SampleCompany1</Name>
    </ComplianceSetup>
    <SiteLocation>
      <ID>SampleCompany1</ID>
      <Name>SampleCompany1</Name>
    </SiteLocation>
  </UserInfo>
          .
          .
          .
</TransformData>
```

Figure 10: Sample Reuters Chat Custom Metadata

3.3.2 Bloomberg

The Bloomberg content will require both a data transformation for effective indexing as well as storing the original content for regulatory purposes to provide the content has not been modified from the original.

Both these forms will be stored in a namespace with differing root folders. The unchanged content will be stored in the top level folder `RawData`. The transformed data will be stored in the top level folder `TransformData`.

The `RawData` layout is a slight variation of the original content as described in section *3.1.2 Bloomberg*. The difference is the files will have the date/time stamp removed and be placed in a folder structure with the year used as the top level folder, and the remaining date/time stamp as the sub-folder name. The following shows an example layout.

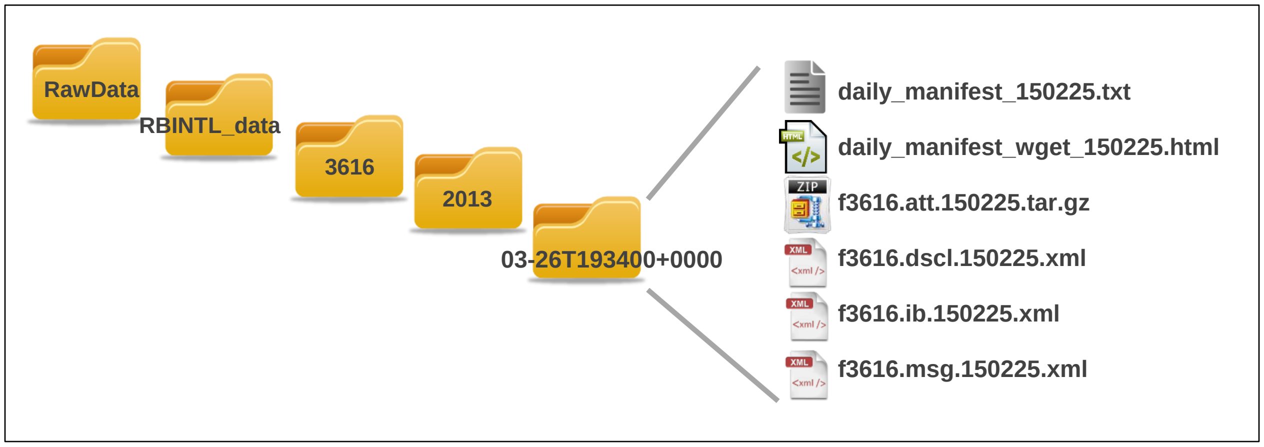


Figure 11: Bloomberg RawData Folder/File Layout

The transformation of the content is conditional based on the individual files. For the `ib` (Instant Bloomberg), the single file is broken down into individual chat conversations. The `msg` (Bloomberg Messenger) is broken down into individual message files. The `att` (Attachments) tar file will be expanded to expose the individual files contained. The `dscl` (Disclaimer) file content will remain changed.

For the `ib` (Instant Bloomberg) content, the transformation consists of creating a folder in the collection named `conversations`. The folder will contain individual files for each `Conversation` tag in the original file. The name of each conversation will have the name in the form of:

```
IB_<CollectionDate><Index>.xml
```

where `<CollectionDate>` is the `YYMMdd` format of the collection and `<Index>` is a number to provide unique names to each conversation object stored in HCP. The `<Index>` will start at 1 and will be an indicator as to the order the conversation appeared in the `ib` file. Also, contained in the `conversations` folder is a `Version.xml` file that contains the `Version` tag contained from the original `ib` file.

For the msg (Bloomberg Messenger) content, the transformation consists of creating a folder in the collection named messages and will contain individual files for each Message tag in the original file. The name of each conversation will have the name in the form of:

```
BM_<CollectionDate><Index>.xml
```

where <CollectionDate> is the YYMMdd format of the collection and <Index> is a number to provide unique names to each message object stored in HCP. The <Index> will start at 1 and will be an indicator as to the order the message appeared in the msg file. Also, contained in the messages folder is a Version.xml file that contains the Version tag contained from the original msg file.

The att (Attachments) file is a tar gzip compressed format file and contains all files that were attached messages in either the ib or msg files. For the collection, a folder named attachments is created in the collection folder and all files along with whatever folder structure exists in the att file will be expanded into that folder.

Along with any file content transformations, the names of the files are normalized to remove the account name and the collection id/date as this information has been transformed as parent folders of the content.

The following figure shows the folder and file layout for the transformed Bloomberg content.

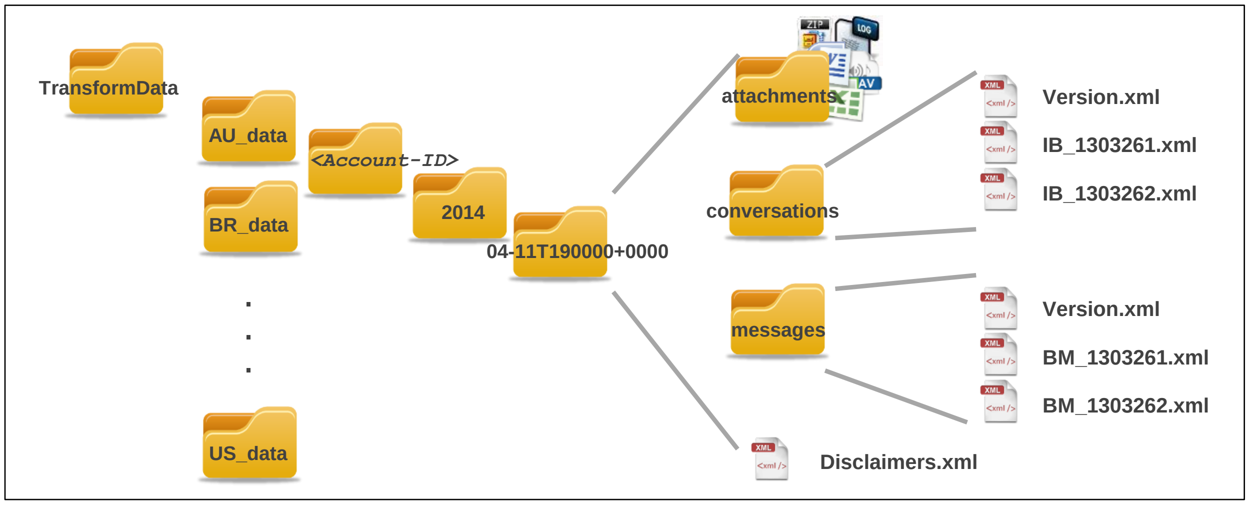


Figure 12: Bloomberg Transformed Layout

Each conversation or message object in HCP contains custom metadata associated with them to facilitate search and content extraction. There is core collection information followed by conversation and message specific information.

The following is a sample of the core custom metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
  <CollectionInfo>
    <Region>NL</Region>
    <DateTime>2013-03-26T19:34:00+0000</DateTime>
    <Account>406811</Account>
  </CollectionInfo>
  <Version>type-specific-version</Version>
        .
        .
        .
</TransformData>
```

Figure 13: Sample Bloomberg Core Custom Metadata

In addition to the core metadata, each type of content (conversation, message), contains its own metadata. The subsections describe each type.

3.3.2.1 Bloomberg Messenger Metadata

Each message object contains its own custom metadata. Along with the core metadata described in the previous section, there is also some message specific information. For messages, there is a `Messages` structure that provides the number of messages in the object, and the earliest starting time and latest ending time for the messages. Following that is a list of `Participants`, that consists of one or more `Participant` tags that contains the corporate and Bloomberg e-mail addresses for those that are included in the messages for this object.

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
        .
        .
        .
  <Messages count="1">
    <StartDateTime>2008-03-11T12:18:17+0000</StartDateTime>
    <EndDateTime>2008-03-11T12:18:17+0000</EndDateTime>
    <Participants count=="1">
      <Participant>
        <CorporateEmailAddress>cg@myco.com</CorporateEmailAddress>
        <BloombergEmailAddress>cg@bloomberg.net</BloombergEmailAddress>
      </Participant>
    </Participants>
  </Messages>
</TransformData>
```

Figure 14: Sample Bloomberg Messsenger Custom Metadata

3.3.2.2 *Instant Bloomberg Metadata*

Each conversation object contains its own custom metadata. Along with the core metadata described in a previous section, there is also some conversation specific information. For conversations, there is the `RoomID` of the conversation. There is a `StartDateTime` of the earliest interaction and `EndDateTime` of the latest interaction in this conversation. Following this basic information is a group of `Activities` structures that provides a list of various activity statistics that appear in the conversation, for example: `Attachment`, `Invite`, `ParticipantEntered`, `ParticipantLeft`, `Message`, `History`, and `SystemMessage`. These activity structures consist of a `count` and `StartDateTime` and `EndDateTime` for the specific activity.

Included is a list of `Participants`, that consists of one or more `Participant` tags that contains the corporate and Bloomberg e-mail addresses for those that are included in the conversation for this object.

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
        .
        .
        .
  <Conversation>
    <RoomID>CHAT-2847039-2373595-113959354479953</RoomID>
    <StartDateTime>2006-02-10T17:52:27+0000</StartDateTime>
    <EndDateTime>2008-03-12T02:15:56+0000</EndDateTime>
    <Activities count="9">
      <Attachment count="1">
        <StartDateTime>2008-03-12T01:19:22+0000</StartDateTime>
        <EndDateTime>2008-03-12T01:19:22+0000</EndDateTime>
      </Attachment>
      <Invite count="1">
        <StartDateTime>2008-03-11T18:25:34+0000</StartDateTime>
        <EndDateTime>2008-03-11T18:25:34+0000</EndDateTime>
      </Invite>
      <ParticipantEntered count="2">
        <StartDateTime>2008-03-11T18:25:34+0000</StartDateTime>
        <EndDateTime>2008-03-11T18:25:37+0000</EndDateTime>
      </ParticipantEntered>
      <ParticipantLeft count="2">
        <StartDateTime>2008-03-12T02:08:00+0000</StartDateTime>
        <EndDateTime>2008-03-12T02:15:56+0000</EndDateTime>
      </ParticipantLeft>
      <Message count="3">
        <StartDateTime>2006-02-10T17:52:27+0000</StartDateTime>
        <EndDateTime>2008-03-11T18:31:37+0000</EndDateTime>
      </Message>
    </Activities>
    <Participants count=”1”>
      <Participant>
        <EmailAddress>cg@bloomberg.net</EmailAddress>
        <CorporateEmailAddress>cg@myco.com</CorporateEmailAddress>
     </Participant>
  </Conversation>
</TransformData>
```

Figure 15: Sample Instant Bloomberg Custom Metadata

3.3.3 Verint Call Recording

The Verint Call Recording content requires both a metadata transformation for effective indexing as well as storing the original content for regulatory purposes to provide the content has not been modified from the original.

Both these forms will be stored in a namespace with differing root folders. The unchanged content will be stored in the top level folder `RawData`. The transformed data will be stored in the top level folder `TransformData`.

The `RawData` layout is a slight variation of the original content as shown in *Figure 5: Verint Input Folder/File Layout Sample*. The core difference is the content will be separated into folders based on the region code that was part of the Verint Job Name used as a folder name in the original file layout. The following shows an example layout.

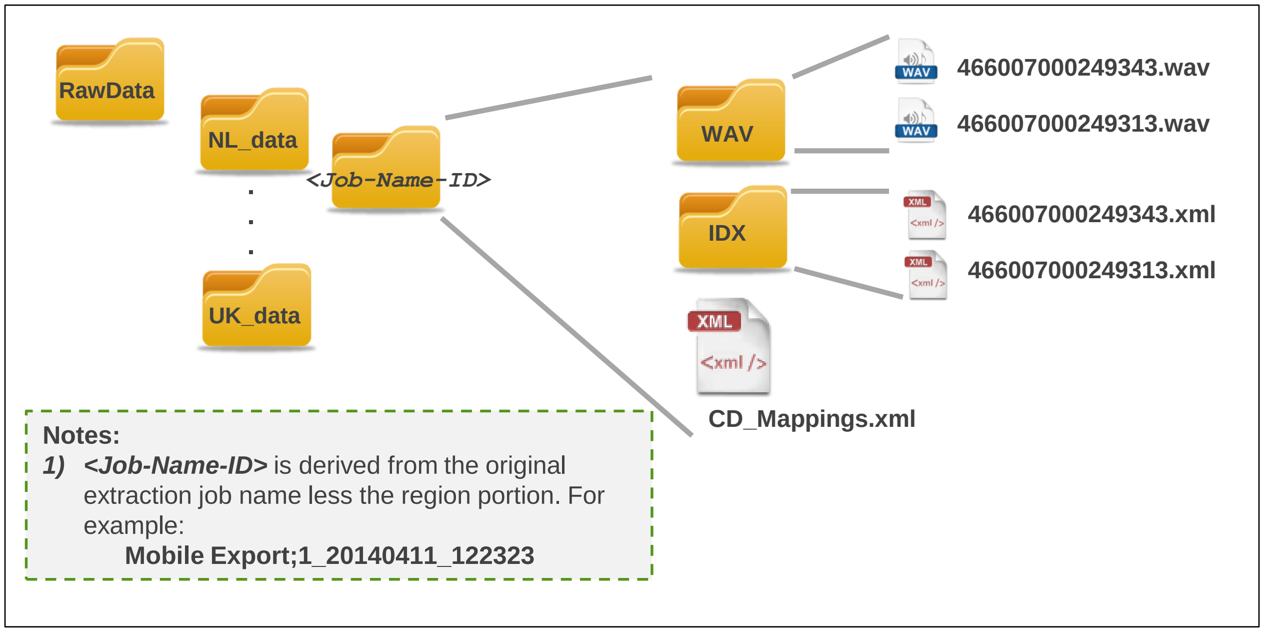


Figure 16: Verint RawData Folder/File Layout

The Verint call recordings consist of the WAV file recordings and associated XML metadata. To facilitate search for call recordings based on the associated metadata, the metadata is ingested into HCP as custom metadata attachments to the WAV files for which they describe; thus the individual call recording XML metadata files will not be ingested into HCP as separate objects. This will help provide a strong mapping between the WAV recording files and the metadata by HCP. Thus the search application will only need to search based on metadata values stored in the HCP custom metadata. The result set will return a URL to the WAV object and the search application can retrieve the WAV file and/or the full custom metadata, if required.

The source data has one top level folder that contains 4 pieces of information:

```
<Region-Code>.<Job-Name>;<Job-Id>_<Date-Time>
```

Instead of keeping this folder name, this is broken up and used to construct a multi-level folder structure. The `<Region-Code>` will be used to construct the top-level folder; for example, with a region code of `NL` in the name, the top level folder will have the name of `NL_data`. The `<Job-Name>` will be a subfolder. This subfolder with `<Job-Name>` will have a subfolder with the `<Date-Time>` value converted into a folder indicating the year, and the subfolder with the formatted remaining portion of date time of the form:

`MM-ddTHHmmssZ` – For example `04-11T122330+0000`

The time zone of the `<Date-Time>` is determined based on information in a region mapping properties file. The default is GMT time zone, but can be overridden in the configuration.

The `<Date-Time>` folder will contain the `CD_Mappings.xml`, and WAV files. The WAV files will have the index XML files attached as custom metadata. There will not be subfolders of `WAV` or `IDX`.

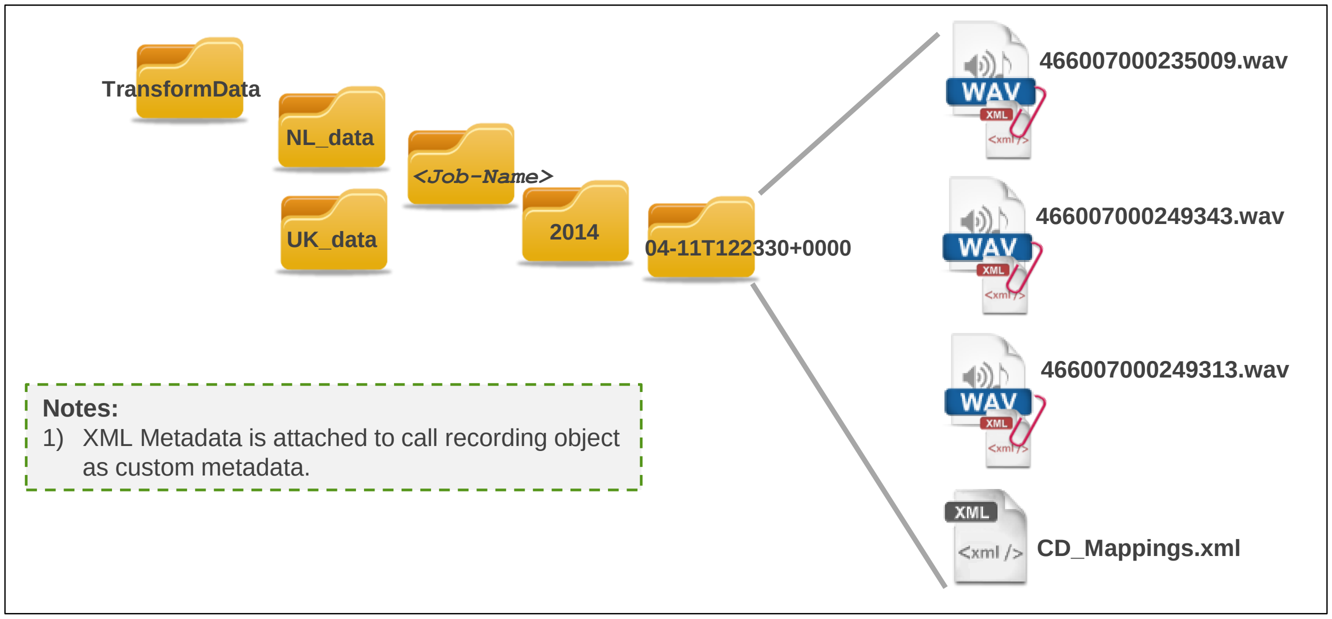


Figure 17: Verint Transformed Layout

The custom metadata attached to the wav file will consist of a transformed form of the index XML files for any given call. The transformation will consist of 3 main areas:

- All metadata will be surrounded with a `TransformData` tag.
- The Verint Extraction Manager job information that was encoded in the job run folder will be encoded into XML structure called `CollectionInfo`.
- The index XML content will be transformed to provide for index/search capabilities of HDDS. The root tag is `CAudioFile`. Within this tag is a child `PrivateData` section that requires some data transformation to facilitate HDDS search against configuration specific data values.

The basic structure of the XML metadata is as follows:

```
<TransformData>
    <CollectionInfo>
        . . . details-omitted
    </CollectionInfo>
    <CAudioFile>
        . . . details-omitted
    </CAudioFile>
</TransformData>
```

Figure 18: Verint Metadata Structure

The following sections will describe the subsections of this metadata.

3.3.3.1 Collection Information Structure

The `CollectionInfo` structure consists of metadata that is provided in the Verint raw content as a folder name generated by the Verint Extraction manager. The folder consists of fields that define <ID>, <Region-Code>, `<Job-Name>`, `<Job-Id>`, and `<Date-Time>`. This information will be stored as custom metadata and included as part of the metadata transformation mentioned in the previously.

The form of the custom metadata will be:

```
<CollectionInfo>
   <ID>id-num</ID>
   <Region>region-code</Region>
   <Name>name-here</Name>
   <ID>number-here</ID>
   <DateTime>YYYY-MM-ddTHH:mm:ssZ</DateTime>
</CollectionInfo>
```

Figure 19: Verint JobInfo Structure

3.3.3.2 Index XML Transformation

The Verint Extraction Manager provides a robust set of metadata in the form of XML. This is stored in a sibling XML file for each WAV file extracted. The details of this structure is beyond the scope of this document; however, the `DictionaryEntry` elements of the `PrivateData` subsection of the XML metadata is transformed to support custom metadata search capabilities of HDDS. The orginal form of the metadata that is a candidate of for transformation is the following:

```
<CAudioFile>
            . . . details-omitted
   <PrivateData>
       <DictionaryEntry>
          <Key>CD1</Key>
          <Value>Outbound</Value>
       </DictionaryEntry>
       <DictionaryEntry>
          <Key>CD3</Key>
          <Value>902432</Value>
       </DictionaryEntry>
             . . . details-omitted
   </PrivateData>
            . . . details-omitted
</CAudioFile>
```

Figure 20: Verint Call Data Metadata Basic Structure

In order to perform searches for all values for a certain key, this section is transformed in two ways:

- Provide mapping of the `Key` value from the generic `CD#` form, to something more content specific.
- Change the overall structure where the `Key` value is integrated into an XML tag name and value of this new tag is the `Value` tag content.

With these goals in mind, for illustration purpose assume that `CD1` is a custom data field for the call "Direction" and the direction is "Outbound". Also assume that `CD3` does not have any specific mapping. The basic structure of the XML will be transformed to look like the following:

```
<CAudioFile>
            . . . details-omitted
  <PrivateData>
     <DictionaryEntry_Direction>Outbound<DictionaryEntry_Direction>
     <DictionaryEntry_CD3>902432<DictionaryEntry_CD3>
             . . . details-omitted
   </PrivateData>
            . . . details-omitted
</CAudioFile>
```

Figure 21: Transformed Verint Call Data Metadata Basic Structure

Each of the 3 types of call recordings (Avaya, Etradeal, and Mobile) has their own mapping from `CD#` to a more meaningful name. Thus each type of call recording performs a different transformation of the XML metadata.

3.3.4 Algomi

The Algomi data collection consists of a manifest file and a group of XML files containing the chat records. These files are not in a form that will provide granular enough information required for the solution search and retrieval needs and will require both a file and metadata transformation for effective indexing and retrieval, as well as storing the original content for regulatory purposes to provide the content has not been modified from the original.

Both these forms will be stored in a namespace with differing root folders. The unchanged content will be stored in the top level folder `RawData`. The transformed data will be stored in the top level folder `TransformData`.

The `RawData` layout is a slight variation of the original content as shown in *Figure 22: Algomi RawData Folder/File Layout*. The core difference is the date information for the data collection will be removed from the files and will be used to construct a folder structure as shown below.

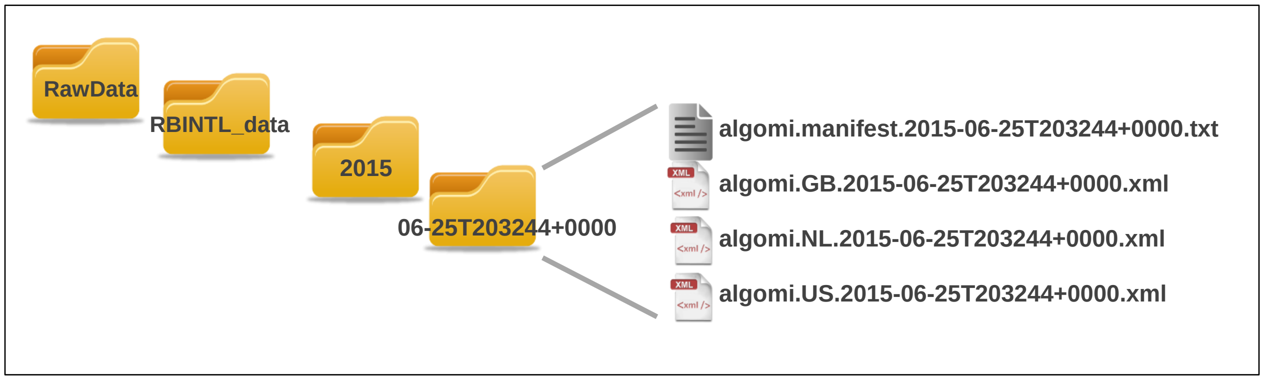

RawData
RBINTL_data
2015
06-25T203244+0000
algomi.manifest.2015-06-25T203244+0000.txt
algomi.GB.2015-06-25T203244+0000.xml
algomi.NL.2015-06-25T203244+0000.xml
algomi.US.2015-06-25T203244+0000.xml

Figure 22: Algomi RawData Folder/File Layout

The data collection content will be transformed into a consumable form into the top-level `TransformData` folder. A data collection consists of XML files where each file has a region specific set of messages. For all messages in a given region, a region folder is created of the form:

```
<RegionCode>_data
```

for example, `NL_data`. A region specific message file consists of all the `ChatRecord` XML elements for the region. Under the region folder will be folder structure generated from the data collection date/time and containing the individual `ChatRecord` XML elements.

The following figure shows an example folder layout for the transformed content.

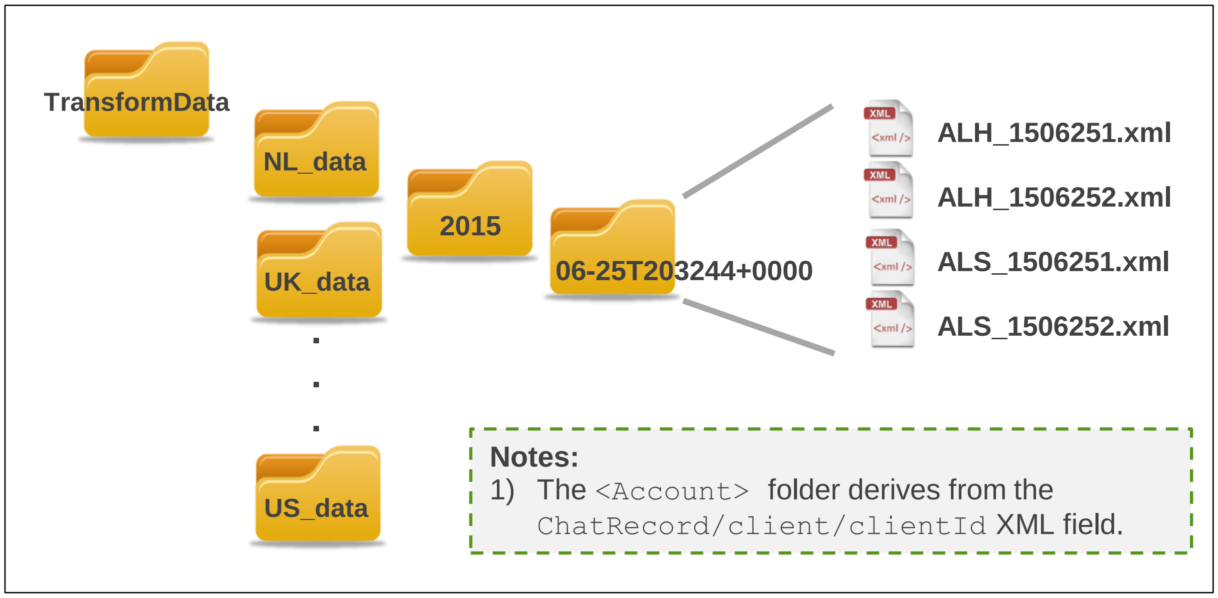


Figure 23: Algomi TransformData Folder/File Layout

Each `ChatRecord` element is extracted and written into individual files. A given `ChatRecord` is of one of the two types: Synchronicity and Honeycomb. The type of record is indicated by the value of the child tag named, `chatType`.

The individual files for each `ChatRecord` will have the form of:

`<TypeCode>_<CollectionDate><Index>``.xml`

where,

<TypeCode> is a 3 letter code consisting of `ALS` for Syncronicity and `ALH` for Honeycomb,

`<CollectionDate>` is the `YYMMdd` format of the collection, and

`<Index>` is a number to provide unique names to each `ChatRecord` object stored in HCP. The `<Index>` will start at 1 and will be an indicator as to the order the specific `ChatRecord` type appeared in the raw file. *Note*: Both the ALS and the ALH files will start with and contain the value of 1.

3.3.4.1 Object Data Transformation

As previously described, each object ingested in HCP will consist of an individual `ChatRecord` XML element. This object will be indexed as XML by the HDDS product. For XML objects HDDS needs to have a hint as to the data type of the XML tag; otherwise, it is assumed to be a string. To provide HDDS the hint, a `dataType` XML attribute needs to be added to those XML tag values that are not strings.

Before content ingestion, a transformation will take place to add the `dataType` attribute with appropriate data type specification to the following fields:

XML Tag Path	Data Type
`/ChatRecord/quantity`	`number`
`/ChatRecord/ChatMessages/ChatMessage/createdAt`	`date`

Table 3: Algomi XML dataType Transform

3.3.4.2 Content Metadata

All HCP objects ingested, both `RawData` and `TransformData`, will contain custom metadata. There will be a core level of metadata consisting of the collection information for all objects. The following is a sample of the core custom metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
  <CollectionInfo>
    <Region>NL</Region>
    <DateTime>2013-03-26T19:34:00+0000</DateTime>
  </CollectionInfo>
        .
        .
        .
</TransformData>
```

Figure 24: Sample Algomi Core Custom Metadata

The values for the `CollectionInfo` child XML tags will derive from the folder structure for which the object resides.

For those objects that represent individual `ChatRecord` XML elements, additional HCP custom metadata is provided. This metadata is a subset/summary of the content in the object itself. The extraction of this information is required to take advantage of the HDDS feature for custom metadata fields that allow for returning the values of these fields with the query result, thus removing the requirement to read the data to find out the field values.

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformData>
  <CollectionInfo>
        . . .
  </CollectionInfo>
    <ChatRecord>
      <chatType>…</chatType>
      <orderId>…</orderId>
      <isinCode>…</isinCode>
      <isinDescription>…</isinDescription>
      <clientId>…</clientId>
      <clientName>…</clientName>
      <quantity>…<quantity>
      <ChatMembers count=”999”>
        <userId>…</userId>
        <userId>…</userId>
        <userId>…</userId>
      </ChatMembers>
      <ChatMessages count=”9999”>
        <startDateTime>yyyy-MM-ddTHH:mm:ssZ</startDateTime>
        <endDateTime>yyyy-MM-ddTHH:mm:ssZ</endDateTime>
      </ChatMessages>
   </ChatRecord>
```

Figure 25: Algomi TransformData Metadata Structure

The following table describes the tag/attribute name and the source of its value. The source specification may contain XPath like syntax of the elements in the object for which it is attached.

Tag/Attribute	Description/Source
`chatType`	Value at `ChatRecord/chatType`
`orderId`	Value at `ChatRecord/orderId`
`isinCode`	Value at `ChatRecord/isin/code`
`isinDescription`	Value at `ChatRecord/isin/description`
`clientId`	Value at `ChatRecord/client/clientId`
`clientName`	Value at `ChatRecord/client/clientName`
`quantity`	Value at `ChatRecord/chatType`
`ChatMembers/@count`	Count of the number of `ChatMember XML` structures in the `ChatMembers` tag.
`userId`	One instance for each unique `ChatMember/userId` value. There will be a list of these XML elements.
`ChatMessages/@count`	Count of the number of `ChatMessage XML` structures in the `ChatMessages` tag.

`startDateTime`	Earliest `createdAt` tag value in all the `ChatMessage` XML structures.
`endDateTime`	Latest `createdAt` tag value in all the `ChatMessage` XML structures.

Table 4: Algomi Custom Metadata Field Definitions

4 Run-Time Environment

Ingestion is performed as standalone software customized for this solution. This section and it's subsections will discuss:

- Linux based execution environment. This environment consists of a system running the Linux OS, solution deployment environment, the execution scripts, and data source configuration.
- The Custom Metadata Enhancement Tool (COMET) is the core ingestion engine to place content on HCP.
- The results management around completion reporting and failure handling.

4.1 Linux Execution Environment

The solution is written to be executed on a Linux OS environment. This Linux OS environment will require the following basic requirements for the solution:

- 64-bit flavor of Linux.
- Bash shell environment for executing the core wrapper scripts.
- Cron (or equivalent) for scheduling jobs for execution.
- Java JRE 1.7 or newer installed for execution of the COMET code.
- SSH access into the Linux environment for monitoring and failure evaluation.
- Linux interactive user to be used both for execution and management of the solution.
- HTTP/HTTPS Network connectivity to the HCP system used in the overall solution.

The execution environment is used to provide the following high level functions:

- Organized deployment environment.
- Execution scripts to encapsulation full processing of any given data type.
- Solution specific configuration for all data sources.
- Processing control.

4.1.1 Deployment Environment

The solution deployment consists of a simple packaging into tar files to be expanded into an appropriate location on the customer supplied system. The deployment consists of core folder structure to organize the overall content with folders to contain run-time libraries, configuration files, and run-time areas for log files, lock files, and minor scratch files.

Any execution scripts described in the next section will be available on the top folder level and will be the main entry points for the all the data types to be processed.

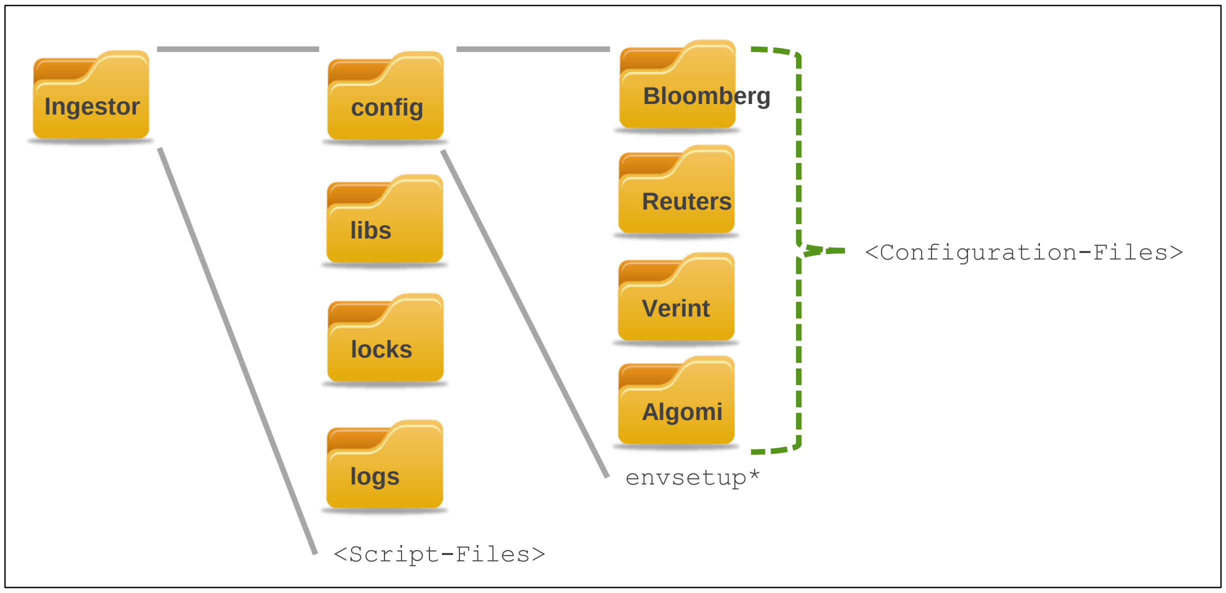


Figure 26: Deployment File/Folder Layout

4.1.2 Execution Scripts

For each data set type, there is a separate high level bash script. This script has all the logic to validate and transform the content found in the input location followed by ingestion into HCP. Each data set has enough differences that each script is only for a specific data set type.

In general, the scripts for each data set type considers the following actions against the data set for which they are written:

- Identify complete data set collections and move to a work area.
- Perform input content validation in the form of hash checking or any other required means.
- Perform any necessary transformations of content in preparation for ingestion into HCP.
- Run the COMET instances required for ingesting the content into HCP.
- Identifying content that could not be successfully processed and move them into a quarantine area.
- Report the completion status to the Search Console database.

4.1.3 Configuration

Each data set has its own configuration. Organized in subfolder in `config` folder. Then may be multiples of one type to process Raw and Transform.

The following table describes the main classes of configuration files:

File	Description
`envsetup_*`	Contains configuration specific environment variables to either supplement or override values set in the general `envsetup` file.
`comet.properties`	Contains core COMET configuration as well as processor configuration. For instance, input sources, threading and

	throughput tuning.
`scanner_*.properties`	Contains scanner specific configuration for item inclusion and cleanup directives.
`generator_*.properties`	Contains generator specific configuration to provide direction on the metadata to be created for items.
`RegionMapper_*.properties`	Contains the configuration for mapping individual items to HCP namespaces.
`log4j2.xml`	Contains configuration for the log4j v2 facility used by COMET. Configuration includes parameters around retention rules for log files.

For a more complete description of the configuration files and their content, refer to the document:

Rabobank – HCP Ingestion Engine Administration Guide

4.1.4 Processing Control

Processing initiation is managed by using Linux cron or equivalent scheduling software. Each data type can be scheduled separately to help spread load on the data ingestion environment and/or avoid quiet periods.

The cron entries will simply be execution of a script residing in the deployment top level folder and any output and error information directed to a log file in the `logs` folder of the deployment.

To trigger manual processing of any data type or maintenance task, this can be accomplished simply by running the applicable script.

4.2 COMET Framework

COMET is a framework written in Java that consists of a pluggable architecture to handle different input sources and produce different metadata for different content types for ingestion into HCP. COMET also has many tuning configuration for adjust throughput for any given environment.

The 3 pluggable modules that can be created for a COMET deployment are the following:

1. Scanner – Contains the knowledge of how to identify content for processing.
2. Generator – Creates the metadata (custom and system) metadata that will be used for ingesting content into HCP.
3. Processor – Contains the logic to ingest content into HCP via the REST protocol.

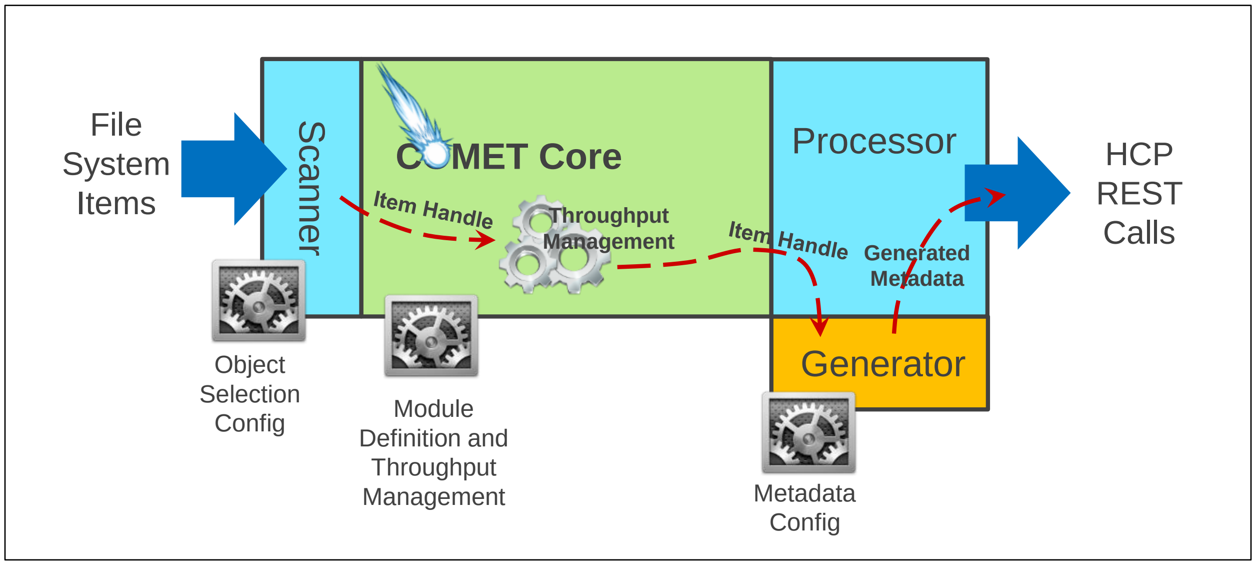


Figure 27: Basic COMET Architecture

For this solution, the Scanner and Processors are or based off of standard modules included with the COMET core. The Scanner will be the based off the standard file system scanner that walks folder structures with configurable filtering capabilities.

The Processor has the core logic for ingesting content into HCP using the REST API. Additionally, it can be configured to compute and verify the hash of the content ingested to ensure chain of custody of content into HCP.

The bulk of the customization is creating the Generators required to generate the metadata for the 3 different data types: Reuters, Bloomberg, Verint, and Algomi. The following section will describe the generator implementations.

4.2.1 Generator Implementations

There is one generator implementation for each type of data to be ingested into HCP. This will provide with simplicity by having separate code for different data types instead of complex code that handles multiple data types.

All generators have the ability to perform mapping of regional data to the appropriate HCP destination information consisting of tenant, namespace, and required credentials. This is accomplished by examining the original files path for a parent folder indicating the region code. One parent folder has the form of `<Region-Code>_data`. The `<Region-Code>` will be used to lookup the HCP location and required location for the content.

In addition, each generator contains the logic to build the custom metadata for the appropriate objects. The metadata consist of information from the source file path, files generated during the transformation phase, and deeper analysis of the object content itself.

The following sections outline the basic approach for each type of Generator.

4.2.1.1 Reuters IM

Reuters IM consists of multiple generators. One will be for the raw content which is simply the ingestion of the content as it was received from the source. This simply will be to provide destination information via the common Region mapping configuration.

The second generator is for the transformed content. This generator will also have the same mapping functionality as the raw data; however, it will also be responsible for creating the custom metadata that will be attached to the Chat session XML files. To generate the Chat session XML, it will collect the `Version.xml` and `Headers.xml` files for the message data collection for which the Chat session belongs. In addition, it will examine the Chat session XML to determine the ID of the users that participated in the Chat session. It will then extract the detailed User information for each participant from the `Users.xml` file. These three pieces will be formulated into valid XML and attached to the Chat session file. An example of this structure can be found in *Figure 10: Sample Reuters Chat Custom Metadata*.

Any other files are just ingested without custom metadata to the appropriate location.

4.2.1.2 Bloomberg Messages

Reuters IM consists of multiple generators. One is for the raw content which is simply the ingestion of the content as it was received from the source. This generator simply is to provide destination information via the common Region mapping configuration.

The second generator is for the transformed content. This generator contains the same region mapping functionality as the raw data; however, it is also responsible for attaching the custom metadata to the message and conversation objects. Before the generator is called, all content was transformed into either multiple files or with different file names with appropriate folder structure.

4.2.1.3 Verint Call Recordings

The Verint Call Recordings are ingested in multiple forms thus will require multiple generators. One is for the raw content which is simply the ingestion of the content as it was received from the source. This generator simply is to provide destination information via the common Region mapping configuration.

The second generator is for the transformed content. The generator contains the same region mapping functionality as the raw data; however, it will also be responsible for generating the transformed folder structure and custom metadata described in section *3.3.3 Verint Call Recording*.

4.2.1.4 Algomi Fixed Income Messages

The Algomi messages are ingested in multiple forms thus will require multiple generators. One is for the raw content which is simply the ingestion of the content as it was received from the source. This generator simply is to provide destination information via the common Region mapping configuration as shown in *Figure 24: Sample Algomi Core Custom Metadata*.

The second generator is for the transformed content. The generator contains the same region mapping functionality as the raw data; however, it will also be responsible for generating the transformed folder structure and custom metadata described in section *3.3.4.2 Content Metadata*.

4.3 Results Management

The Data Ingestion system contains a robust environment for handling failures and allowing for analysis of the failures to rectify the situation. The results management consists of overall completion reporting that is integrated with the search console. The failure/recovery process consists of analysis of log files and performing manual operations to re-insert content for ingestion into HCP.

The overall goal of this is to:

1. Enable ability to show regulators progress and status of items that require tracking. The level of granularity for Reuters is Chat Sessions, Bloomberg is messages and conversation sessions, Verint is call recordings, and Algomi is chat session.
2. Provide solution integrated indicators of success/failures of ingestion. Upon failure detection, manual intervention is required to determine detailed cause.

The detailed failure/recover process is described in the document:

Rabobank – HCP Ingestion Engine Administrative Guide.

4.3.1 Completion Reporting

In the simplest form, the main operating scripts write to standard output of the shell. The output contains the overall summary of items successfully ingested and counts of other kinds of failures. The output of the script will be stored in a log files when configured for automated execution via Linux cron utility. See the next section on Failure/Recovery for more details on failure analysis.

This overview of the completion status can either be sent to e-mail recipients, and/or submitted to the search console to store in the user interaction database table for unified reporting/monitoring. The information is not exhaustive, but instead is just used as a history indicator of the health of ingestion. Any failures need to be evaluated via the detailed log files available on the system.

Submission is performed by issuing an HTTP POST request to the search console the following content main fields will be provided to the search console:

Field Name	Description	Value
Data Set Name	Specifies type of data that was ingested	`Verint, Reuters, Bloomberg, Algomi`
Completion Status	The high level completion status of the run.	`SUCCESS, ERROR, WARNING`
Completion Message	A user consumable text string to provide details about the completion status.	
Start Time	Start date/time of the data ingestion	`Date/Time value`
End Time	Ending date/time of the data ingestion	`Date/Time value`
Processing Statistics	Type and count of number of items processed. Processed does not mean success, but instead how many items were detected.	`Object type string. Count of that number of objects.`
Failure Statistics	Various counters for failure conditions.	`Counts for Collect File`

		`Warning, Collect File Failure, Validation Failure, Duplicate Failure, Ingest Failure, Residual File Failure`

Table 5: Search Console Interation Table Fields

4.3.1.1 Ingestion JSON Reporting Object

This section describes the JSON reporting object that will be submitted to the search console. This object will be submitted as an HTTP POST operation with the data payload being the JSON report object .

The following is the general JSON form of the object:

```
{
        "dataSetName": <data-set-name>,
        "completionStatus": <status-type>,
        "completionMessage": <message-string>,
        "startTime": <date-time>,
        "endTime": <date-time>,
        "processedStats": [
             { "type": <object-type>, "count": <number> },
              …
        ]
        "failureStats": [
             { "type": <general-failure-text>, "count": <number> },
              …
        ],
        "user": <user-name>,
        "password": <user-password>
}
```

The data fields are described in the following table:

Value	Description
`<data-set-name>`	The high level name of the data ingested: Bloomberg, Reuters, Verint, Algomi.
`<status-type>`	Overall status of the ingestion processing: `Success` : All items ingested `Warning`: A recoverable error occurred, but should be understood. `Failure`: One or more items failed to be ingestion
`<message-string>`	A free form string that provides additional information about the status reported.
`<date-time>`	Start/End time of the ingestion processing in the form of: `yyyy-MM-ddTHH:mm:ssZ`
`<object-type>`	Indicates what basic type of object(s) for the `successStats count` represents.

	Values are: `ChatSession`, `Message`, `CallRecording`

`<general-failure-text>`	Category name for failures encountered.
	Values are:
	`Ingest Failure`
	`Validation Failure`
	`Duplicate Failure`
	`Residual File Failure`
	`Collect File Warning`
	`Collect File Failure`

Table 6: Ingestion Report JSON Object Field Definitions

NOTE: `FailureStats` array will only be provided if there are any failures.

4.3.2 Failure and Recovery Procedure

The data ingestion process involves a handful of processing phases to transition from receipt of the raw content to the point of having content ingested into HCP. The main phases are described in *Table 1: Data Set Processing Phases*. During any of these phases there can be many different types of failures that occur.

During any of these phases if a failure occurs, the end result will be that any non-processed content will be set aside into a quarantine area. The quarantine area is rooted at the path:

```
/appl/ingest/Quarantine/HCP_Production
```

Under this folder will be two subfolders named `RawData` and `TransformData`. The organization under this folder will be identical to the work area structures described in *section 3.2.4*

To evaluate the reason for content being placed in the quarantine area, the log files can be analyzed to root out the details of the failure. Appropriate manual actions can be taken to resolve the issue (be it environmental or unexpected data set form), re-inserting the current state of the data set collection into the processing work flow for retry, then re-initiate the data ingestion processing.

The following subsections will discuss the possible failures in each phase of processing, where to look for failure details, typical actions to resolve the issues, and how to appropriately re-introduce the failed content into the multi-phased processing.

4.3.2.1 Raw Data Set Capture

This phase of processing consists of identifying content placed into appropriate data set and region folders rooted at the path:

```
/appl/ingest/HCP_Production
```

Raw input files will be placed into a data set appropriate subfolder by the file transfer mechanism provided by Rabobank. This phase will evaluate the files placed in the folder and determine when there is a full data set collection that can be processed. Once a full data set collection is identified, it will be moved to the data ingestion `RawData` work area described in section 3.2.43.3 Data Set Transformation/Layout. The move operation will consist of copy of all relevant files to the data ingestion work area and upon success will remove files from raw input file folder.

Until a full data set collection is identified, any files in the folder will remain untouched. The main processing scripts will not detect stale files remaining. A maintenance task that runs outside of the main data ingestion will detect stale files and report the condition.

During this phase processing, it is possible for disk space to become exhausted or un-writable for the data ingestion work area and/or the quarantine area. Under these conditions, it may be the case that some files have been successfully copied to the work area. If this occurs, it will make a best attempt at cleaning up the files that were copied. The next attempt at execution will copy over any previously copied files.

Although one exception here could be that someone cleaned out the raw input files and then the partially copied files will remain in the work area and evaluated in the next validation phase.

4.3.2.2 Data Set Validation

The data set validation phase consists of looking at the content constructed in the data ingestion `RawData` work area. The validation checks are described in section *3.2 Data Set Validation*. If any of the validation fails, the content cannot be processed further. The full data set collection will be moved to the quarantine area under the `RawData` folder structure for the data set it belongs.

Evaluation of the log files should indicate which validation rule was validated. This content should be evaluated for the cause of failure and identify how the content was made invalid. It may require regenerating the content from the source resulting in the data set collection being re-inserted into the raw input file area.

If content is fixed in the form stored in the `RawData` folder in the quarantine area, this content can be copied back into the appropriate folder location in the data ingestion `RawData` work area. It will then be processed on the next data ingestion execution.

4.3.2.3 Data Set Transformation

The data set transformation phase consists of using the validated content existing in the data ingestion `RawData` work area and performing all transformations as described in section *3.3 Data Set Transformation/Layout*. The operations could consist of activities like unzipping files, creating additional XML files, etc. The resulting transformed content will be written to the `TransformData` folder structure.

During this processing, the likely failures are due to unexpected command failures due to either disk space issues and/or unexpected corrupt files not evaluated during the data set validation. If any failures occur, the content constructed in the `TransformData` folder will be deleted and the `RawData` folder for the data collection will be moved to the quarantine area.

Upon evaluating the log files for the cause of the failure, the appropriate corrective action can be taken either with the execution environment or the content itself. If the content is corrupted, this should be tracked back to the source of the content to determine how and why it was corrupted. If appropriate resolution is to manually fix the content, the `RawData` content can be repaired by appropriate means and re-inserted into the processing stream by placing it in the appropriate folder in the data ingest `RawData` work area.

4.3.2.4 Data Set Ingestion

The data set ingestion phase consists of executing an instance of COMET against the content residing in the data ingestion work area. The execution will be against both the `RawData` and `TransformData` contents for a given data set. The final expected result of the COMET execution is that all content is ingested into HCP and removed from the data ingestion work area.

There are a number of possible causes for COMET not ingesting content into HCP and thus leaving files behind. The following are some possible causes:

- HCP unreachable
- HCP in read/only state.
- Misconfigured HCP
    - Permissions issue
    - Tenant/Namespaces missing
- Individual object already exists
- COMET execution environment misconfigured

Any details of failures will be recorded in the COMET logs stored in the logs folder. Upon any failed item will remain in the data ingestion work area. It will be dealt with during the final cleanup and failure reporting phase.

4.3.2.5 Cleanup and Failure Reporting

During this final phase, the data ingestion work area will be evaluated for any residual files found. If there are any files, these will be copied to the quarantine area in the form they were found. The content left behind could be in one of these forms along with its likely resolution path.

- Full data set collections consisting of both its `RawData` and `TransformData` forms. Can consist of multiple regions depending on what data sets were received.
- Various individual files likely caused by already existing content from prior partial failed attempts.

Regardless of the failure, the files will be moved into the quarantine folder structure in the same structure they resided on the data set work area.

Recovery will depend on the state of the content and the cause of the failure. If there were no problems with the data itself and caused by an environmental failure, it can be reintroduced into the data ingestion by moving them into the data set work area maintaining the same folder and file layout.

For any individual files, the cause is likely be either from content already existing in HCP or the files are not candidates for ingestion based on the current COMET configuration.

5 Out of Scope Items

The overall solution has a requirement to integrate with HP ArcSight SIEM. The data ingestion will not integrate with Arcsight directly, but instead will record ingestion processing completion status in the search console database.   If required to be reported in ArcSight, it will be the responsibility of the search console.

The data ingestion is not subject to security testing as requested via the Pen Test Partner testing. Any security into the data ingestion system must be managed by the Rabobank supplied Linux system. The data ingestion system is a backend processing engine and the only communication into this system would be through content transform via the Rabobank supplied Message Queuing system. All other communication is outbound using standard protocols like HTTP/HTTPS into HCP system and Search Console system.

Appendix A

HCP Tenant/Namespace Mapping

The tables below represent the tenants and namespaces for each data source type. For the full mapping and additional information see the document *HCP Data Ingestor: Directory, Tenant, and Namespace Naming Standards* created by Rabobank.

Bloomberg

The Bloomberg tenant for production will be named `HCPBLM400`. Within this tenant there will be the following namespaces created and the owning country:

Namespace	Customer IDs	Country
BLMARRAB400	548478	Argentina
BLMAURAB400	30146097 624225	Australia
BLMBERAB400	30204878	Belgium
BLMBRRAB400	43673	Brazil
BLMCLRAB400	30086012	Chile
BLMCNRAB400	30167443 30167694	China
BLMHKRAB400	30114181	Hong Kong
BLMINRAB400	30214976	India
BLMIDRAB400	213002	Indonesia
BLMIERAB400	30082955	Ireland
	30122731	Japan
BLMMERAB400	30009312	Mexico
BLMNLRAB400	30537 30009756 30197429 196970 30184577 30140309 554332 196976 217426 30208420 179739 128915 3616 201308 748220 30135751	Netherlands

	102834 707527	
BLMSGRAB400	117481 117589	Singapore
BLMTRRAB400	30218535	Turkey
BLMGBRAB400	30217083 73828 206856 30211910 30218690	United Kingdom
BLMUSNAW400	30219040 220733 8229 319647 30052667 585552 30050391 30201800 30216818 30087853	United States (North America Wholesale)

Reuters

The Reuters tenant for production will be named `HCPREU400`. Within this tenant there will be the following namespaces created and the owning country:

Namespace	TRMC ID	Country
REUAURAB400	52100	Australia
REUBRRAB400	58010	Brazil
REUCLRAB400	319686	Chile
REUCNRAB400	54760	China
REUHKRAB400	56686	Hong Kong
REUINRAB400	472643	India
REUIDRAB400	141416	Indonesia
REUIERAB400	9456	Ireland
REUNLRAB400	499212	Netherlands
REUSGRAB400	82665	Singapore
REUESRAB400	347910	Spain
REUTRRAB400	470343	Turkey
REUGBRAB400	237	United Kingdom
REUUSNAW400	56769	United States (North America Wholesale)

Verint

The Verint tenant for production will be named `HCPVER400`. Within this tenant there will be the following namespaces created and the owning country:

Namespace	Country
VERNLRAB400	Netherlands
VERGBRAB400	United Kingdom

Algomi

The Algomi tenant for production will be named `HCPALG400`. Within this tenant there will be the following namespaces created and the owning country:

Namespace	Country
ALGNLRAB400	Netherlands
ALGGBRAB400	United Kingdom
ALGUSRAB400	United States
ALGRDRAB400	Corporate Raw Data