



Data Collection and Preparation for Currency Adaptation

Applied Data Science in an Industrial Banknote Image Recognition Framework

Degree course: [BSc Informatik]

Authors: [Sophie Haug]

Constituent: [CI Tech Sensors AG]

Experts: [Prof. Dr. Macha Kurpicz-Briki]

Date: 19.06.2021

Abstract

In this project concepts and methods for collecting and preparing data for Currency Adaption are presented, in the context of the company CI Tech Sensors in Burgdorf. The currency data needed to train and develop algorithms in CI Tech's currency recognition and classification framework should be provided as centralized labelled data sets. These data sets should be generated automatically from the centralized Data Lake. This project proposes a new data container "NoteSet", as well as exemplary workflows to achieve this goal. The implementation was done in Python and using the Jupyter Notebook environment.

Versions

Version	Date	Status	Remarks
1.0	19.06.2021	Final Version	Final corrections

Contents

Abstract	i
1 Project Goals	1
1.1 Initial Position	1
1.2 Project Goals	1
2 Project Methods	3
2.1 Introduce NoteSets as a new note data container	3
2.2 Implement Processes for Automated NoteSet Generation	3
2.3 Introduce NoteSet loading in MCM	4
3 Technical Background	6
3.1 Introduction	6
3.2 The NIF File	6
3.3 The CDF file	6
3.4 MOVEm File Index	7
3.5 MOVEm Data Service	7
3.6 MOVEm Simulator	8
3.7 MOVE Currency Modeller	8
3.8 The Notelist Format	9
4 NoteSet Data Container Specification	10
4.1 NoteSet Specification	10
4.2 NoteSet Annotation Specification	12
4.3 Integrating the NoteSet Data Container in MCM	15
5 Automated generation of NoteSets	17
5.1 Introduction	17
5.2 Technologies used	17
5.3 Exemplary Notebooks	17
5.4 A practical application - analyzing a suspiciously high rate of possible counterfeits for Malaysian Ringgit (MYR)	19
6 Conclusion / Results	22
6.1 Introduction	22
6.2 Conclusions on introducing the NoteSet data container	22
6.3 Conclusions on the automated NoteSet generation using Jupyter Notebooks	22
6.4 Challenges encountered	23
6.5 Outlook	23
List of figures	25

1 Project Goals

1.1 Initial Position

1.1.1 Background

The 80/20 rule of Data Science states that most data scientists spend about 20% of their time on actual data analysis and 80% of their time finding, cleaning and preparing data. The process of reading data for the training, testing and implementation of an algorithm thus is not only a highly important step in the Data Science Lifecycle, it's maybe the costliest.

CI Tech Sensors, based in Burgdorf, has over 30 years experience in the field of banknote image recognition. One of its core competences is the development of currency adaptation software, which is loaded onto the banknote reader, and enables it to identify and classify banknotes for specific currencies, as well as determine their authenticity and fitness. This software is continuously updated and developed by a dedicated team of Currency Adaptation Specialists at CI Tech Sensors, whose work relies on the availability of a Data Lake of Banknote Raw Data.

Over the last three years, there has been ground-breaking shift in raw data management: From folder-based data storage on both local and central file drives - which heavily relied on filenames for identification and was thus prone to ambiguity and duplication of data - towards a centralized Data Lake, where each file was given a unique ID. At the same time a cloud based Microservice architecture (MOVEm Data Management Toolchain) for the management and maintenance of this Data Lake was developed. This development comes with a number of exciting possibilities, some of which to explore is the goal of this project. The focus will be on the collection and preparation of tailored banknote sets.

In earlier days, the collection and preparation of banknote data had to be manually done by the Currency Adaptation Specialist. They would have to navigate to the relevant folder, select the relevant files and sort them by denomination, emission, orientation and other criteria, all information taken from a file's annotation. Not only is this process cumbersome and error-prone in itself, it also solely relies on the file annotation, which is made by a human annotator before the actual banknote recording, and thus can contain errors as well.

Today it's already possible to group banknote raw data into sets, using a text-based file format called *Notelist*. Adaptation specialists will manage and create their own Notelist files locally according to their specific needs. Notelists contain references to individual file IDs and Note IDs as well as additional annotational information and the file path where files can be found. However this format comes with some problems of its own as we will see later.

1.2 Project Goals

1.2.1 Annotated Reference Data Sets

We want to find a way to define, generate and persist annotated note data sets that can be used as reference data sets in the context of currency adaptation. By reference data set we mean a reliable and authoritative set of note data, that can be used for training and development of CDFs. An example for a reference data set could be the set of "all EUR 5 b orientation 1 notes"¹ or "all CNY 100 b counterfeits".

These data sets should fulfil the following criteria:

- Accuracy (with respect to annotation): Data records have a correct and binding annotation.
- Consistency: When regenerating a data set, it should always contain the same notes for a given state of the Data Lake.

¹The concept of orientation refers to the banknote's orientation when transported through the reader. There are 4 possible orientations: frontside upright (1), backside upright (2), frontside upside down (3) and backside upside down (4)

- **Completeness:** In the generation of reference note data sets, generally all data in the Data Lake should be considered.

1.2.2 Mechanism for generating tailored Data Sets according to context-dependent criteria

More generally and based on requirement 1.2.1 of reference note data sets, it should be possible to generate data sets according to user specific criteria. For example, the set of all outliers for algorithm result X within a specific folder, or all Category 3 (suspected counterfeits) within a particular directory of field data. A process should be installed which allows specialist users to generate such datasets. These datasets will have to fulfil the same criteria as the reference data sets.

1.2.3 Persisting Note Data Sets

We want to have a lightweight, cost-friendly way to persist the generated note data sets. Especially, the format in which we persist the data sets should be independent of individual file locations of NIF files and make use of the advantages of the MFX and MDS microservices.

2 Project Methods

2.1 Introduce NoteSets as a new note data container

We will introduce so called *NoteSets* as a new data container for referencing and persisting sets of note data. While these Note Sets will be exported to files, they do not constitute a file format *strictu sensu*, but rather a new paradigm as how to think of collection of notes: Namely no longer as a group of notes that were recorded in the same transaction and thus reside in the same binary file (in earlier times this was the only way one could think of collections of notes), but rather as a collection of IDs that are linked by a common *annotation*.

The note set data container will have the following advantages:

- Compactness: NoteSets will be minimalist in the sense that notes are referenced by ID only. There is no annotational information stored in the note set itself.
- Independency of file locations: With the introduction of MFX the actual location of a file in the Data Lake becomes redundant. Therefore, the note set does not store any file paths.
- Equivalence with respect to annotation: Unlike the Notelist format, NoteSets will no longer have an individual annotation per note. Rather, they have at most one annotation per NoteSet, which is to be persisted in the NoteSet's filename (cf. section 4.2). Thus, NoteSet are considered equivalent with respect to annotation

Meanwhile, the new data container must at least satisfy the following requirements:

- Convertability to and from the existing Notelist format. The Notelist format has been established as the standard way of storing data sets in Currency Adaptation. Even if one were to pursue the goal of replacing the Notelist format by the Noteset format, convertability would be needed at least for a period of transition.
- Operability. Since they are so minimalist, Note Sets are more ideal than previous formats for quick comparison. We need a way to compare and find the difference between Note Sets reliably and efficiently. Furthermore we would like to do other set operations like union or intersection on NoteSets.

The NoteSet data container is introduced in a small Python library providing the abovementioned functionality. This library is deployed as a Python package and can be used in the Currency Adaptation Python toolchain, as well as in Jupyter notebooks. In a later step it remains to analyze the feasibility of introducing this data container in the .NET toolchain as well.

2.2 Implement Processes for Automated NoteSet Generation

In a first phase Jupyter Notebooks are used to generate the NoteSets. Different methods and their results, i.e. the resulting NoteSets, will be compared and presented to the Currency Adaptation Team. Once the NoteSet generation process has been refined on a technical level, it remains to decide how to implement it in the Currency Adaptation Software toolchain.

The NoteSet generation workflow in this development phase can be generalized as follows:

1. Crawl MFX for a chosen directory in the Data Lake to obtain all File IDs.
2. Query each note from MDS and determine its annotation/label based on a chosen labeling method.
3. Group all notes with the same annotation/label into NoteSets (within a NoteSet, all elements have the same label)

It remains thus to choose a labeling method. The possible methods can be outlined as follows:

1. Query MDS to obtain the results calculated by the banknote reader for each note. In this case, we use the banknote reader (and the CDF which was loaded at the time of recording) as the labeller: For example, if the reader classified the note as e "EUR 5 b Orientation 1" at the time of recording, we will give it that label. This is the most straightforward way, since all that is done is query existing information from the MDS REST API. No new information is generated.

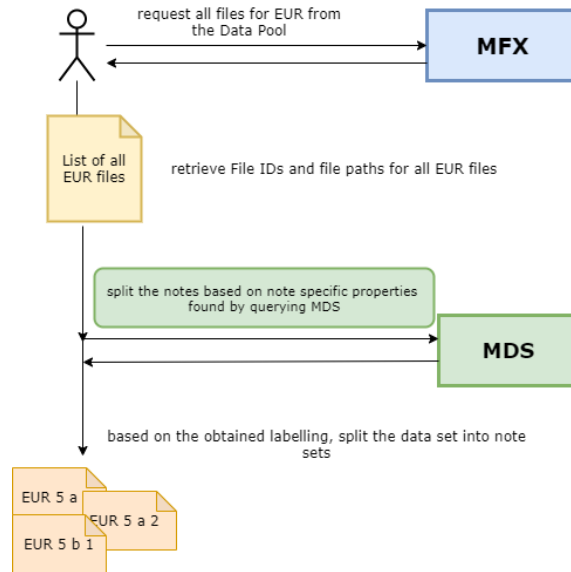
2. Use the MOVEmSimulator to simulate the banknote reader on the chosen set of notes and a chosen CDF. The results generated by the Simulator can then be used to group and label the Note Sets. This method is more complicated and time consuming than the first one, but it's also more generic, because it makes us independent from the CDF which was used in the original recording. For example, the original CDF used in the actual recording might classify a note as Category 3 (suspected counterfeit), but might now be outdated since the latest CDF for the same currency will classify the same note as Category 2 (proven counterfeit). On MDS, the information generated in the original recording is stored, therefore, this note will always be a Category 3 note if we use the first method. If we let the simulator process the chosen set of NIF files first, using the latest CDF, we will get the most up to date information, namely the results a physical reader would calculate using the most recently released CDF.
3. Computationally load all notes into MCM and use algorithm(s) from the MCM algorithmic framework to determine properties of each note based on which the notes can be labelled. In this case use MCM as the labeller. To access the MCM application from a Jupyter Notebook, a Python wrapper library for MCM has to be used.

Figure 2.1a to 2.1c visualizesthese different approaches.

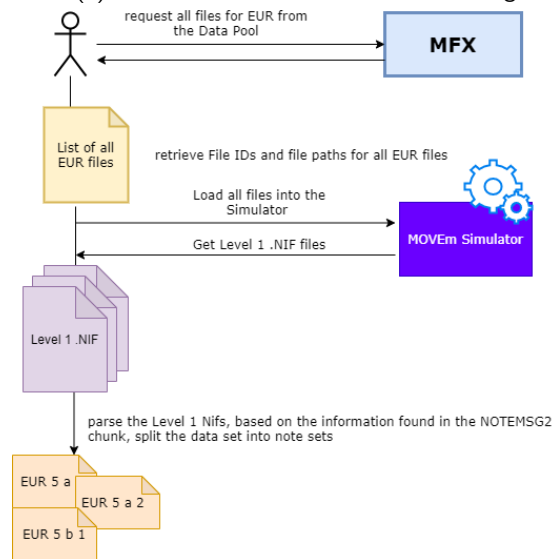
In the course of implementing the project the focus shifted almost entirely towards the method of labeling via MDS on the one hand and the MOVEm Simulator on the other. In a first stage I tried to use MCM as a labeller, but discarded this mainly because of performance reasons. MCM is already a very resource-intensive piece of software and accessing it via a Python wrapper from Jupyter had another negative impact on the performance. In addition, the python wrapper for MCM is quite limited as only a number of MCM functions are accessible and MCM is impossible to debug when accessed via Python.

2.3 Introduce NoteSet loading in MCM

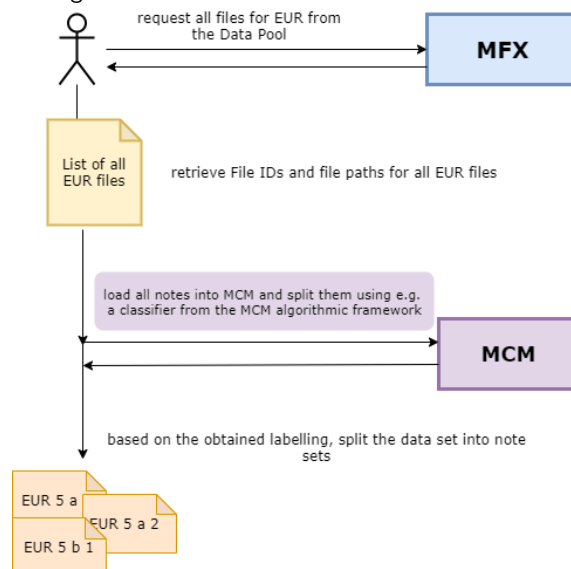
Since MCM already uses MFX, it is able to load NIF files by File ID only, thus nothing stands in the way of loading Notes by ID, i.e. as a set of Note IDs. Therefore we want to extend MCM with a feature that allows loading and exporting of NoteSets. This should be done with as little cost as possible as introducing bigger changes and new models into MCM is always very complex and delicate. The idea is to allow passing NoteSets in the MCM View layer, but converting them into standard MCM Notelist objects before they are passed to deeper layers.



(a) NoteSet Generation with MDS labeling



(b) NoteSet Generation with MOVEm Simulator labeling



(c) NoteSet Generation with MCM labeling

3 Technical Background

3.1 Introduction

This section will provide an overview about existing technologies of CI Tech Sensors, namely the ones which are involved and affected by this project.

3.2 The NIF File

NIF stands for *Note Information File* and is a binary file format which stores note transaction records. In other words a NIF file is what you get when processing a set of notes through a banknote reader. The NIF stores Metadata, Note Record Data and Image Data, it thus contains both raw data and data calculated by the reader. It is encoded in a proprietary binary format, which uses a tree-like structure of so called Chunks, parts of data with a unique ID and a fixed length to store and structure data. An example of a relevant Chunk, which is referenced throughout this documentation is the NOTEMSG2 (Note Message 2, historically because there used to exist a Note Message 1 Chunk) Chunk, which is present once for each note recorded and which contains the most relevant information about a note's recognition and classification by the reader. This information can then be used for labelling our dataset.

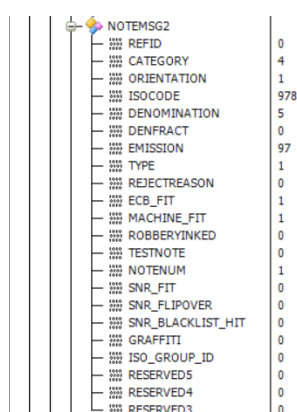
A NIF has a unique ID (the file's MD5 checksum), which is also called File ID. Derived from it, the Note ID can be defined as the concatenation of the File ID and the note's¹ FilePosition, which corresponds to the note's position in the bundle, e.g. the first note has FilePosition 1 etc. Thus

`8AD47001CC4BD7B49783B4D6125F9551 : 1`

is the unique ID of the first note in the NIF file with checksum `8AD47001CC4BD7B49783B4D6125F9551`.

3.3 The CDF file

The CDF (*Currency Data File*) is a binary parameter file, which contains all the required information to enable the banknote reader to process, classify and validate all banknotes of one currency. It stores metadata as well as algorithmic recognition data, which specifies the algorithms used, their order of execution and the respective

A screenshot of a proprietary ChunkFormat parsing software interface. It displays a list of fields for the NOTEMSG2 chunk, each with a small icon, a name, and a numerical value. The fields are listed in a column, with their values aligned to the right. The values range from 0 to 978.

NOTEMSG2	0
REFID	4
CATEGORY	1
ORIENTATION	978
ISOCODE	5
DENOMINATION	0
DENFRACT	97
EMISSION	1
TYPE	0
REJECTREASON	1
ECB_FIT	1
MACHINE_FIT	0
ROBBERYINKED	0
TESTNOTE	1
NOTENUM	0
SNR_FIT	0
SNR_FLIPOVER	0
SNR_BLACKLIST_HIT	0
GRAFFITI	0
ISO_GROUP_ID	0
RESERVED5	0
RESERVED4	0
RESERVED3	0

Figure 3.1: Content of NOTEMSG2 Chunk as displayed in a proprietary ChunkFormat parsing software

¹We informally speak of notes but in this context actually do not mean notes as an abstract concept like "The CHF 10 banknote" but rather *a recording of a physical note*. Simply put: A note is a record of a physical banknote, which was done by a banknote reader and is stored in a uniquely identifiable NIF file. Technically, such a note recording can also be uniquely identified by the note's serial number and the angle with which it passes through the reader (a note will likely never have the exact same angle in different recordings)

parametrization for each algorithm and banknote type. The CDF is released independently of the actual banknote reader software and the development and maintenance of CDFs is one of the main tasks of Currency Adaptation Specialists. Typical reasons why a CDF would need to be adjusted are the release of a new banknote or the emergence of a new type of counterfeit which the CDF needs to be able to detect.

3.4 MOVEm File Index

MFX (MOVEm File Index) is a central index of all files in the Data Lake, allowing the lookup of a file's location in the Lake. The Data Lake is hosted on a network drive in the proprietary Cloud of CI Tech and its partner companies and stores both test data (data which was recorded inhouse with a dedicated recording tool for testing) and field data (data which was recorded by banks or other clients on productive banknote readers) in its raw, unaltered form. It uses hierarchical directory structure, which is however only loosely regulated. There is new data being continuously added to the Data Lake using a dedicated procedure, which we will not delve into here.

MFX is a Flask application with a REST API backend. In addition to allowing the lookup of individual files, it lists the indexed directories as well as all duplicate and defect files in the lake. Moreover, MFX is integrated into MCM, insofar as MCM has access to the MFX API and can thus load directly from the Data Lake. This integration made it possible to load files in MCM by File ID only. The following figure visualizes how MFX is used for path lookup within MCM:

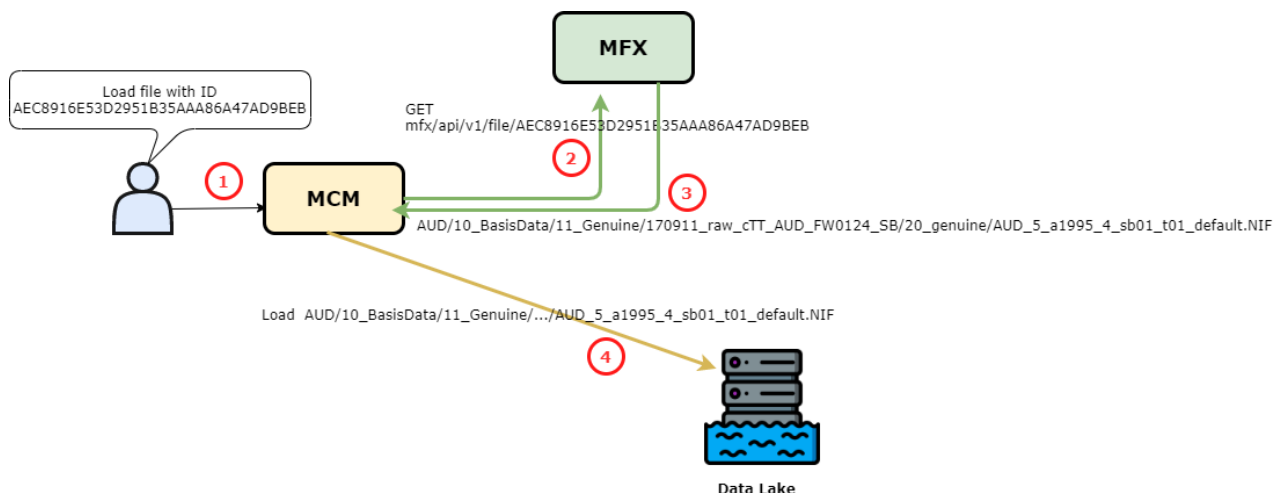


Figure 3.2: MCM uses MFX for lookup of file paths in the Data Lake

3.5 MOVEm Data Service

MDS (MOVEm Data Service) is another REST based Flask web application, with the MDS database in the background. It allows the lookup of NIF data, namely on the levels of NIF (metadata, recording data, device data etc.), Note (all note specific data recorded by the reader, such as algorithm results or the values of the NOTEMSG2 chunk presented above), and Image (the actual note images taken by the reader cameras).

Currently the MDS database stores about 322 GB of NIF data, whereas the Data Lake stores about 6.67 TB of NIFs, which are about 1110714 individual files. The MDS application also entails a parsing module, which converts the NIF elements into corresponding Python objects, that constitute the JSON responses of the REST APIs.

The big improvement that comes with introducing MDS is that it allows lazy loading of NIF data. Depending on the use case, MDS allows fetching NIF data (metadata on the file, like the time of recording, the reader used etc.), Note data, or Image data i.e. actual images. This is a huge difference to the MCM architecture, where always the entire NIF is loaded and all images cached. In many use cases, adaptation specialists analyze algorithmic data on the level of the note only and don't need all of the images or any images at all. There are even NIF analyzing

tools in the Currency Adaptation Software Toolchain that do not support the displaying of images at all, but still unnecessarily load and cache all images, because they use and underlying MCM core for the loading of NIF data. The promise of MDS in the long run is therefore improved loading time and less use of memory, mainly due to the aforementioned lazy loading of notes and images. However, to achieve this goal, the architecture of MCM would have to be substantially changed.

Figure 3.3 visualizes the MFX-MDS architecture.

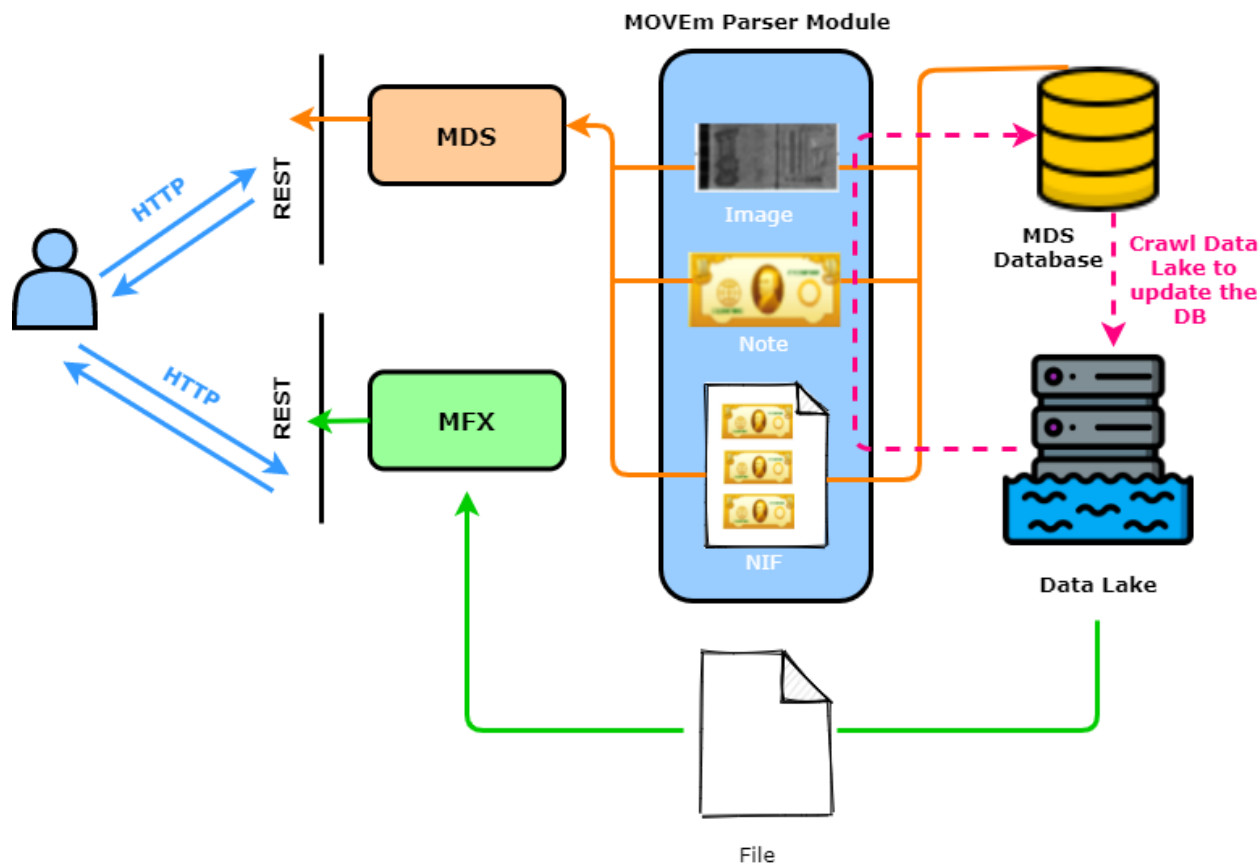


Figure 3.3: MFX and MDS architecture

3.6 MOVEm Simulator

The MOVEm Simulator is a user interface to the core application of the MOVEm firmware, the embedded banknote reader software. It's a desktop application written in Python and Qt and its purpose is to test CDFs without the need of operating a physical banknote reader and actual physical banknotes. Input of the MOVEm simulation are existing NIF files. The simulator then takes the raw data from these files and calculates classification, sorting and algorithm results the same way one can expect from a physical test with real banknote readers. Output of the simulation are so called Level-1 .NIF files. These are valid, minimal NIF files which contain all the computed results, but no image data (which of course the simulator cannot generate).

3.7 MOVE Currency Modeller

MOVE Currency Modeller (MCM) is the adaptation specialist's main tool which they use to create and test Currency Data Files (CDF). Typically, users load raw data (NIF files) into MCM and use this data to develop, test and improve CDFs or train a particular algorithm within a CDF. MCM has an interface to the embedded software

algorithmic framework and thus is able to simulate the individual algorithms used by the banknote reader. MCM is a desktop application which has been developed over years and contains a huge range of viewers and functionality. It's a very powerful tool but also a monolithic one, and as of today not very modularizable. Although MFX is already integrated into MCM as mentioned above, MDS isn't yet. The integration of MDS is in the works and comes with the promise of performance improvements, because it would allow the lazy loading of individual notes or images from the MDS API, as opposed to always having to parse and load the entire NIF, even if one just wants to look at a particular note or image therein.

3.8 The Notelist Format

Since the NIF file is immutable, the need to persist sets of notes requires a dedicated format. The Notelist format is an XML-based format with the file extension ".nl". It is structured in a hierarchical way and consists of the following elements:

- Notelist: A list of NoteContainers
- NoteContainer: A list of NoteItems, typically refers to a NIF file
- NoteItem: refers to a single note, contains its annotation/label

The original purpose of the Notelist format was on the one hand the already mentioned possibility to group notes from different NIFs together. Another feature, is the per-note annotation (NoteItem), which allows the adaptation specialist to annotate notes differently from and independently of the reader's recognition/labeling. For example, a note might be recognized as a Category 4 (genuine) by the current CDF but the adaptation specialist knows it's actually a new type of counterfeit and thus wants to label it as a counterfeit (Category 2) when using it in the training data.

Notelists can be loaded in MCM and today constitute the standard means of persisting sets of notes in the context of currency adaptation.

```
<?xml version="1.0" encoding="utf-8"?>
<Notelist Version="1" BasePath="T:\F+E\WA\700_Work\730_cTT\AUD1003">
  <NoteContainer RelativePath="..\..\..\500_Währungsdatensätze\540_MOVEm\aud\10_BasisData\11_Genuine\190201_raw">
    <NoteItem>
      <FilePosition>3</FilePosition>
      <Guid>7f33fa8b-1970-4ed3-b157-0ac8983bbf4e</Guid>
      <CUR>AUD</CUR>
      <Denomination>20</Denomination>
      <Orientation>One</Orientation>
      <Emission>d</Emission>
      <Quality>New</Quality>
      <Type>Genuine</Type>
      <Bank>unknown</Bank>
      <FileDescription />
      <FileComment />
      <IsWorkingNote>false</IsWorkingNote>
      <WorkingNoteList>negative</WorkingNoteList>
    </NoteItem>
    <NoteItem>
      <FilePosition>7</FilePosition>
      <Guid>343a7820-a5a4-44a7-8e2a-749fcf4a425f</Guid>
      <CUR>AUD</CUR>
      <Denomination>20</Denomination>
      <Orientation>One</Orientation>
      <Emission>d</Emission>
      <Quality>New</Quality>
      <Type>Genuine</Type>
      <Bank>unknown</Bank>
      <FileDescription />
      <FileComment />
      <IsWorkingNote>false</IsWorkingNote>
      <WorkingNoteList>negative</WorkingNoteList>
    </NoteItem>
  </NoteContainer>
</Notelist>
```

Figure 3.4: Extract of a Notelist file

4 NoteSet Data Container Specification

4.1 NoteSet Specification

4.1.1 Motivation

By introducing the NoteSet as a new data container the primary goal is not to introduce a new data format. Firstly, because this would entail a longer process of defining a format schema and approval of various departments, the potential users of the new format, as well as a long introduction phase. Secondly, a data format is always restrictive in that it imposes an absolute way of how data should be thought of. The goal of introducing the NoteSet is exactly the opposite: We want users to think of a data collection as exactly that and not more: A collection of data elements, that at least for now does not come with a set of restrictions and rules of its own, or with as little as possible. In our case, this means that a NoteSet is just a set of notes which can be found in our Data Lake, or put differently: a set of Note IDs. In section 2.1 we listed the advantages and requirements for the new data container. In this section the steps undertaken to reach them will be specified.

4.1.2 Problems with the Notelist format

Even though the Notelist format (see section 3.8) has been established as a de facto standard, there are some disadvantages that come with it and that the introduction of the NoteSet data container hopes to address:

- Too much flexibility: Since Notelists are annotatable on a per-note basis and not on a per-set basis, there is no limit to mixing notes from different currencies, denomination etc. within the same list. This can lead to the temptation to store all notes a user works with over a timespan of months or longer in one single big Notelist. However this defeats one of the Notelist's original purposes namely creating more clarity by storing notes into use-case-specific sets that can be re-used by others.
- Readability: Large notelists can get hard to read (and even harder to compare), due to the bulk of annotational information which in most cases is redundant.
- Insufficient congruity with MOVEm Data Service APIs: Since the Notelist Format requires the presence of a file path, it is not possible to directly generate Notelists from MDS queries. If we query notes from MDS we get back a JSON structure containing the note data stored in the NIF, but no file location (The NIF does not store file location). Thus, we have to get the file path for each File ID with an additional MFX query and consequently increase the amount of queries. This is still feasible, but it slows down the generation of data sets.¹

Furthermore, the Notelist format requires the presence of an outdated GUID for each note. In other words, while the Note ID is the unique note identifier in the universe of MDS and MFX, the Notelist always requires an additional identifier.

4.1.3 NoteSets as equivalence classes

One distinguishing feature of NotSets is that they will not have a per-note labelling but one labelling for the entire set, the NoteSet annotation. This is in the spirit of the adaptation specialist's way of working with data, insofar that they are generally not interested in the individual notes as such, but rather in an algorithm's performance on a particular set of – correctly labelled – notes. Thus we can think of NoteSets as equivalence classes where the equivalence relation is defined by the NoteSet's annotation. The NoteSet Annotation is specified below in section NoteSet Annotation Specification

¹The necessity of the file path in the Notelist hails from a time where the MFX service had not yet been established. Therefore the Notelist needed to know the file location of each NIF on the local file system, in order to be able to load it. Today, where MFX is used to load the notes from the Data Lake, the NIF's File ID is enough information for locating and loading it.

4.1.4 Serialization

Since NoteSets are basically just a collection of IDs, we will store them as text files in ASCII format. In a generic NoteSet, each line will represent a note. We do allow the following rules and elisions, some of whose usage will make the serialized form more compact, if parsing slightly more complicated:

1. Notes within the same file can be listed on the same line, e.g.:

```
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 1
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 2
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 3
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 4
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 5
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 12
```

is equivalent to:

```
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 1 – 5, 12
```

2. If we mean to include all the notes in a file, we can reference them just by the File ID. Thus, assuming that a file contains 100 note recordings:

```
374DB8C13B9B1DB8C42C1B109DE3BBD5
```

is equivalent to:

```
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 1 – 100
```

and also equivalent to:

```
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 1
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 2
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 3
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 4
...
374DB8C13B9B1DB8C42C1B109DE3BBD5 : 100
```

3. We will allow comments, preceded by #
4. Any software with a NoteSet export feature will use the generic way of just listing the individual Note IDs and not use the abbreviated forms presented above

4.1.5 The pynoteset library

Since there is already a comprehensive Python toolchain for Adaptation Software, it was an obvious choice to implement a library *pynoteset*- This small library can generate NoteSet object from ASCII-files or lists of FileIDs/Notelds, and perform the classical set-theory operations on NoteSets. The features of the pynoteset library can be summarized as follows:

1. generate NoteSet objects from files
2. export NoteSet objects to ASCII files
3. annotate NoteSet objects and parse string annotations
4. merge NoteSet objects: for two NoteSets A and B generate the NoteSet $A \cup B$.
5. intersect NoteSet objects: for two NoteSets A and B generate the NoteSet $A \cap B$.

```
260AF80EB470D72AE73B374B572583C4:1
260AF80EB470D72AE73B374B572583C4:2
260AF80EB470D72AE73B374B572583C4:3
260AF80EB470D72AE73B374B572583C4:4
260AF80EB470D72AE73B374B572583C4:5
260AF80EB470D72AE73B374B572583C4:6
260AF80EB470D72AE73B374B572583C4:7
260AF80EB470D72AE73B374B572583C4:8
260AF80EB470D72AE73B374B572583C4:9
260AF80EB470D72AE73B374B572583C4:10
260AF80EB470D72AE73B374B572583C4:11
260AF80EB470D72AE73B374B572583C4:12
260AF80EB470D72AE73B374B572583C4:13
260AF80EB470D72AE73B374B572583C4:14
260AF80EB470D72AE73B374B572583C4:15
```

Figure 4.1: Extract of a generic NoteSet file

```
BC7CFE0DFC4285F8C080CB1AA1D8828B #this is a comment
AEC8916E53D2951B35AAA86A47AD98EB
8AD47001CC4BD7B49783B4D6125F9551:1,2-3
3AEA78D2F8629E6F8A31E727CE16C6F7:4 #here is another comment
0431CA32E68C1611E653CAA39FA538B:1-10
374DB8C13B9B1DB8C42C18109DE38B05
006C14DD69769451E20F63811D:68003
```

Figure 4.2: Extract of a NoteSet file using abbreviations and comments

6. compare NoteSet objects: for two NoteSets A and B generate the NoteSet $A \setminus B$ and $B \setminus A$.

The library consists of three main classes, the NoteSet class, the NoteSetAnnotation class and the NoteSetConverter class. The latter's purpose is to generate NoteSet objects from files and lists of strings, a functionality that was outsourced into a converter class because of a dependency on MDS, which I didn't want to have in the NoteSet class itself² See the pynoteset class diagram figure 4.3 for reference. Furthermore, a converter class NoteSet2Notelistfile has been added to the library *notelistfile*, which allows the conversion of NoteSets to Notelist, the standard XML based format currently in use (cf. section 3.8). This functionality is particularly powerful since a majority of the Python toolchain already uses a Notelist class internally. The converter thus provides an interface for the NoteSet object to be used in other tools.

For deployment of this library a Bitbucket CI/CD pipeline is used, which deploys the package to a software repository manager (JFrog Artifactory). Thus registered Artifactory users can install the pynoteset package like any other using e.g. *pip*.

4.2 NoteSet Annotation Specification

4.2.1 Motivation

As already mentioned, we introduce NoteSets as a container for notes which are equivalent with respect to a yet to be defined annotation, the labeling of the set. Since the NoteSet itself is basically just a set of IDs or keys, all other substantial information will be in the annotation. The first challenge is to select the properties which should be part of the annotation. The second is the question of how to persist the annotation within the NoteSet. Since we do not want to introduce a new format definition, it's not possible to add the annotation as a sort of header in an XML or JSON format. This would mean that a new format has been introduced and some kind of schema would have to be defined for it.

The only option that remains is thus to store the set's annotation within the filename itself when exporting a NoteSet. For this a special file format string has to be introduced (see below).

²The reason for this dependency is the generosity of the NoteSet serialization syntax as specified in section 4.1.4: Remember that if we mean to include all the notes in a file, we can reference them just by the File ID. Internally however, the NoteSet object has to know the number of notes in a file. Therefore, in the case where just a File ID is given, we have to query the number of notes (property "note_count") from the MDS API.

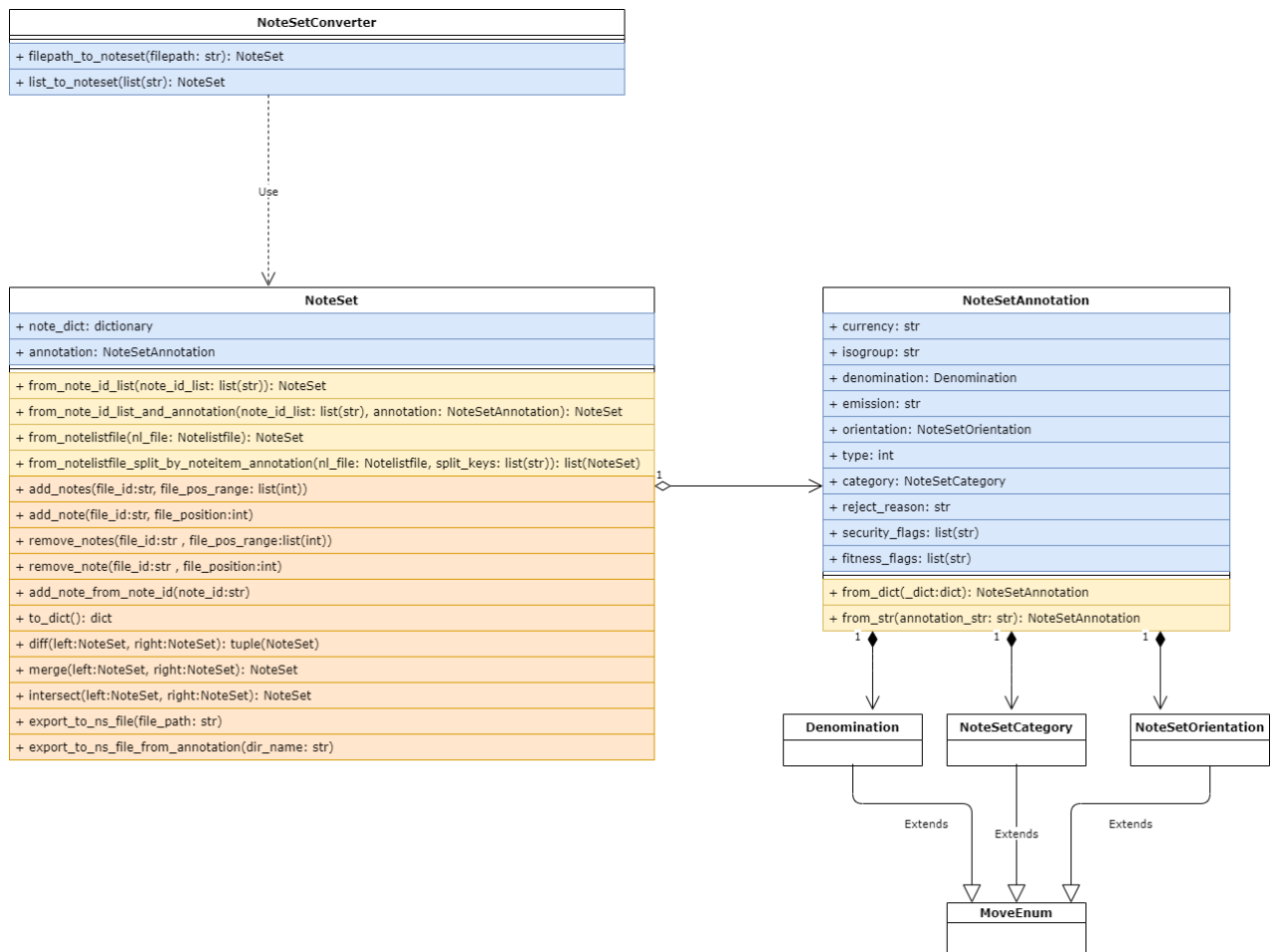


Figure 4.3: Class Diagram of *pynoteset*

4.2.2 Criteria for selecting annotational elements

The selection of the elements which should be part of the annotation was overall not that easy, on the one hand because there are hundreds of properties in a NIF file and we want to be as specific as possible, on the other hand, a NoteSet's annotation should be as simple as possible, namely because we want to persist the annotation in the filename, which ideally should be a readable and not too long string. Finally after having consulted with the adaptation specialist team, the selection was done by the following criteria: The mandatory properties are properties which are needed to identify a particular type of note in MCM. The optional properties (with the exception of comment) are criteria for which are not mandatory to identify a note, but for which it is likely that a user wants to split or filter a set of notes by them when analyzing or training algorithms.

4.2.3 Mandatory Elements of the NoteSet Annotation

The following attributes form the mandatory part of the NoteSet's annotation. These are all derived from properties set in the NOTEMSG2 chunk of the NIF file.

Currency Corresponds to the ISO string matching the the numeric ISOCODE value in the NOTEMSG2 Chunk.

IsoGroup A sub-specification of a currency that is mainly used for the currency GBP (Great Britain Pound), which has different banknotes and issuers for the groups England (BOE), Scotland (SCO), Northern Ireland (IRL), Guernsey (GUE), Jersey (JER) and Isle of Man (MAN). For example, the England GBP would be abbreviated as GBPBOE, the Irish one as GBPIRL.

Emission Emission of the notes in the NoteSet, corresponds to the value of the Chunk EMISSION in NOTEMSG2

Orientation Orientation of the notes in the NoteSet, corresponds to to ORIENTATION in NOTEMSG2

Type NoteType, a numeric value to identify the CDF reference note this note is matched to. In some cases there is more than one reference note for the same denomination, emission and orientation (for example when a currency has different issuers/printers that each print their own distinctive banknotes)

4.2.4 Optional Elements of the NoteSet Annotation

The following attributes are optional elements of a NoteSet Annotation. They are not identifying properties but allow to specify a NoteSet.

Class Corresponds to the value CATEGORY in NOTEMSG2 Chunk and is an attribute for authenticity and fitness of a banknote, for example, Category 4a notes are authentic and fit notes, Category 2 are counterfeits. It's a likely use case that a user would want to split notes across a currency by category to train and analyze fitness and security algorithms.

Security and Fitness flags **Flags** Security and Fitness Flags are set by certain algorithms and indicate that a note might be classified as unfit or counterfeit. For example, the fitness flag STAIN indicates that the note has some kind of smudging, the security flag ROBBERY_INK indicates that the presence of robbery ink was detected.

Reject Reason This is an enum value found corresponding to the value REJECT_REASON in NOTEMSG2 chunk. It indicates the reason why a note was rejected or that it was not rejected.

Comment This is an optional comment added by the user and the only value within the NoteSet annotation which does not come from the NIF file

4.2.5 Specifying the file format string

As already stated, the NoteSet's annotation is to be converted into a filename when exporting a NoteSet to ASCII. The file format string to be constructed has to be readable, complete and indicate the optionality of properties, i.e. distinguish the optional properties from the mandatory ones. Finally, the goal is to codify the format into a regular expression and implement an annotation parser in the python noteset library *pynoteset*.

Preliminary Decisions

- The file format string must be defined in a regular expression. Thus any tool which would later adopt the paradigm of NoteSets could easily validate NoteSet annotations independent of the programming language.
- Generic placeholder for non-specified mandatory properties: Since we allow NoteSets to be merged, we have to anticipate the scenario where e.g. a NoteSet contains notes of various denominations. Therefore we need a placeholder for all mandatory properties indicating the state of 'Undefined' or mixed. This placeholder is 'XXX' for currencies and isogroups and 'X' otherwise, a practice which is already in use in the recording of note data and thus one that users are already familiar with.
- Allowing of both upper- and lowercase spelling: While it is common to use uppercase notation for currency ISO codes and isogroups and lowercase spelling for emissions, we will allow both upper- and lowercase notation to make the already fairly complicated string format less restrictive.

Finally, from these requirements such a regular expression could be defined as follows:

```
([A-Z]{3}){1,2}_((([0-9]*[.])?[0-9]|X)*_([a-z]|X)_([1-4]|X)_([1-9]+|X)(_CL[1-4])?(_SF\((([A-Z_0-9]+)([A-Z_0-9]+)*\))?(_FF\((([A-Z_0-9]+)([A-Z_0-9]+)*\))?(_[a-zA-Z0-9]*)?)
```

The examples in figure NoteSet Annotation Examples illustrate the application of this regular expression, i.e. some valid NoteSet Annotation file strings

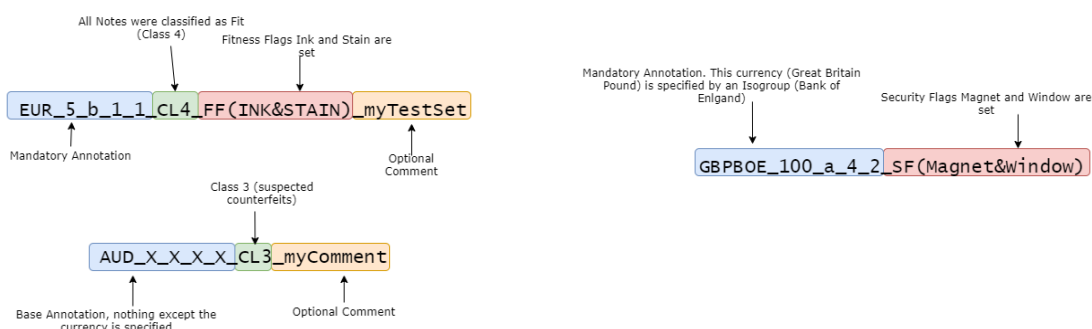


Figure 4.4: NoteSet Annotation Examples

A parser for annotation strings was included in the pynoteset library and since we have defined a regex for it, it would be relatively easy to implement this feature in other tools as well.

4.3 Integrating the NoteSet Data Container in MCM

In order to enable MCM to load NoteSets, a minimal viewer was added to one of the main viewers, which allows to copy-paste and load NoteSets, following the syntax specification described in section 4.1.4. Internally, during loading these NoteSets were converted into Notelists, the standard Note data container object used in MCM, thus there was no major MCM refactoring to be done in the course of implementing this feature. Albeit very simple, this viewer was positively received by developers and adaptation specialists likewise. Later on I added a small feature allowing to export MCM Notelists as NoteSets (ASCII files).

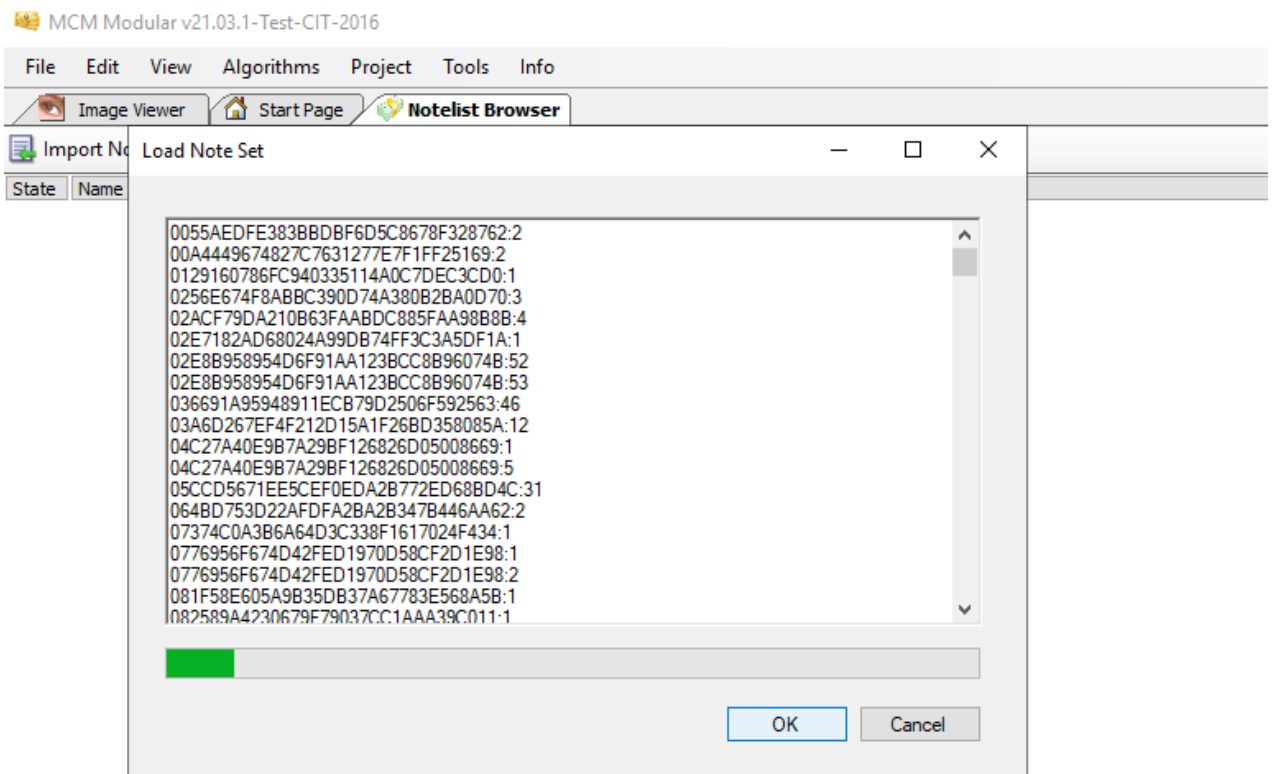


Figure 4.5: NoteSet Loading Viewer in MCM

5 Automated generation of NoteSets

5.1 Introduction

5.2 Technologies used

To develop an automated NoteSet generation process, access to MDS and MFX services is of course required. To ensure availability of these services outside the proprietary Adaptation Cloud (CAPaaS-Cloud), my team leader helped me to set up a local Kubernetes cluster on which these services run using *kind* (Kubernetes in Docker). Like this it was possible to use MFX and MDS services on a local machine, which was very helpful as the CAPaaS-Cloud was sometimes down during the weekend for maintenance reasons.

For the NoteSet generation software itself the decision was made to use Jupyter Notebooks in the phase of this project. The reasons for this decision are as follows:

1. Adaptation specialists are expert users, technically speaking one could say they are Data Scientists. Therefore it could be beneficial for them to learn to use and possibly extend Jupyter Notebooks, where the programming logic is not hidden from them. It would allow them to see themselves in the role of a developer and Data Scientist and not just an end user. Several adaptation specialists have recently presented themselves to be interested in learning to work with Jupyter Notebooks.
2. As long as the NoteSet data container is not the standard data container across the entire toolchain, developing a standalone NoteSet generation software comes with a risk of being rejected by users who are not used to working with this data container. Jupyter Notebooks however can in this case be used as a prototype for (expert) users to get feedback on this new approach.

5.3 Exemplary Notebooks

To familiarize users with the new data container and to start a dialogue about data collection and data set generation, I developed a number of exemplary Jupyter Notebooks, each for a use case, that seemed realistic and typical. The use cases are the following:

5.3.1 Crawl a directory and generate NoteSets automatically

Splitting a chosen directory into NoteSets. The user has a directory of files in the Data Lake which need to be analyzed. For example, this could be field data recorded on banknote reader systems that are in industrial use and might be under suspicion to be faulty in some way. After the user has specified their directory, all NIF files in that directory will be registered and queries made on MDS for each note within that NIF file. The notes are then splitted based on their annotational information found in NOTEMSG2 chunk, authenticity and possible reject reason. The generated NoteSet files are stored in a directory of choice. Optionally they can be exported into NotelistFiles.

During the time I worked on this project, a problem fitting this use case actually came up and the Notebook designed for it could be put into practice (see below).

5.3.2 Get all notes for a currency by filtering

This use case is strongly related to the first one. A user would like to get all notes from a particular currency but filtered by certain criteria. These could be again the “classic” criteria specified in NOTEMSG2 (denomination, orientation, emission etc.) and Security/Fitness flags. In this case the user’s query could e.g. be: “Give me all EUR 5 b orientation 2 notes”, or “Give me all GBPBOE notes with the TAPE flag set” (Notes for which this flag is set have been found to have a tape region and would therefore be considered unfit). Depending on the use case a more exotic query would also be possible, e.g. “Give me all EUR notes which were recorded on the device with

a particular serial number“. In actual fact, we could filter by any property which is part of the MDS note object's JSON structure. This fact was received with great interest by the adaptation team. Again, it is possible to convert the generated NoteSets into NotelistFiles directly from the Notebook.

This is the procedure one would consider when generating reference NoteSets for all currencies.

5.3.3 Perform Data analysis on a chosen NoteSet

In this use case the user has an existing NoteSet persisted in .ns file format or a a Notelist in a .nl file. This NoteSet is then loaded and converted into a pandas dataframe. Using this dataframe, it is possible to quickly analyze data in jupyter, for example, show results for a particular algorithm in a plot, thereby circumventing the necessity to load notes into MCM for analysis. The challenge here was to find a good way to flatten the nested JSON structure of the MDS note object into a pandas dataframe.

I finally used a generic, recursive method to do this and then renamed some columns for better readability. Once the Note object was flattened into a DataFrame I could use the power of pandas on it. For example, I could very easily generate NoteSets for quantiles, e.g. generate a NoteSet for the lower and upper 5% -quantile of the Note Dimensions or NoteSets for outliers, using e.g. the $z - Score$. This process would allow an adaptation specialist to quickly generate a NoteSet for any outliers on a chosen property or algorithm and load these notes immediately into MCM. Of course the property to be analyzed and quantile values can be freely chosen.

5.3.4 Intersecting NoteSets from MFX and MDS

Since MDS only represents the binary NIF data and no filepath is stored in the NIF, the information where a file is located cannot be obtained from it – that is by definition MFX's task. This has in the past proven to be inconvenient: In a typical scenario a user gets a fresh directory of field data saved somewhere in the Data Lake. Imagine they want to find all the Category 3 Notes in this directory, using MDS. MDS however does not know about the Data Lake's directory structure it can only return all Category 3 notes in the Data Lake or the Category 3 notes in an individual file. One way to solve this is to first crawl all File IDs from the directory using MFX and then search these files for Category 3 notes using MDS – this is exactly what was done in 5.3.1. Another possible way is to convert the set of File IDs in the directory under analysis into a NoteSet (remember that a File ID alone already constitutes a NoteSet: namely the set of all notes within that file), then take the set of all Category 3 notes (obtained from MDS) and finally intersect the two sets. The beauty of this approach is that it does not require us to look at the actual note objects' properties, but is technically achieved purely by comparing two sets of Note IDs. The disadvantage is that it can take quite long to find all notes matching a particular query in the entire pool (using Jupyter it took about 27 minutes to get all Category 3 notes).

5.3.5 Scripting the MOVEmSimulator and generating NoteSets from its output

When exchanging my results with the adaptation team, a recurring feedback from their side is that they need to be able to change note records in order to use them in further development of CDFs. The notes as they are recorded in the NIF (and stored in MDS database), reflect the state or version of a CDF at the time of recording. Oftentimes however, adaptation specialists would like to know what results already existing raw data would yield if one were to use a different, more recent CDF. For this the MOVEmSimulator can be used. Since the MOVEmSimulator is a scriptable application, I could add a Notebook for this use case.

The steps are as follows:

- The user specifies a directory with NIF files and a CDF.
- For all the NIF files in this directory, a SimulationInput object is generated and passed to an instance of MOVEmSimulator, on which the chosen CDF has been loaded
- The simulator generates a SimulationResult object for each NIF, which can then be exported to a Level-1 NIF, i.e. a NIF containing reader results but no images

- This set of NIFs can then be analyzed and split up into NoteSets as previously done, with the only difference that the NIFs are parsed locally (these are adhoc generated Level-1 NIFs, not data from the Data Lake) instead of via MDS.
- With the set of NoteSets we obtain from this we can do various things: Compare each NoteSet with the corresponding NoteSet from the MDS data: For example, one might ask how does the set “BOB_20_1_1_CL4_FF(Tape)”¹ I obtained from the simulator differ from the same set I got from MDS? Alternatively, one could repeat the simulation procedure one more time with the same directory of NIFs and a different CDF and compare the resulting NoteSets.

5.4 A practical application - analyzing a suspiciously high rate of possible counterfeits for Malaysian Ringgit (MYR)

In the course of this project, a real-life problem which occurred in the context of Currency Adaptation allowed me to put my existing findings into use: Reports from the field indicated that there was a suspiciously high number of Category 3 notes (notes, which are reported based on a security feature but have not been proven to be counterfeits) for Malaysian Ringgit (MYR) notes. To analyze this problem, a number of banknote reader systems in Malaysia recorded note data for a couple of days, resulting in about 1 TB of note data. The reason for falsely elevated Category 3 rate could theoretically be a problem in the CDF or a hardware fault on a series of devices.

The problem facing adaptation specialists now was how to filter and analyze this big block of NIF data. 1 TB is too much data for MCM to process, so data would have to be split into chunks. However the bigger problem was that there is currently no way to “pick and process” notes based on certain criteria in MCM. Ideally one would like to load a set of notes into MCM and automatically generate a new notelist for say all Class 3 notes within that set. Since this feature does not exist however, more cumbersome ways were needed to split a large set of notes according to users’ needs.

Namely, in the past a dedicated person from the Algorithm Development Team would process a large set of NIF files using MATLAB scripts. They would parse the NIFs, find the relevant notes, chop the binary NIF files up and back together, thus generating artificial NIF files containing only the notes relevant, in this case, Category 3 notes (see Figure 5.1). This process is not only cumbersome (a huge amount of binary data has to be parsed, analyzed, chopped up and rearranged, thus analysis of 1 TB like this would take several days), it is also highly problematic from a Data Science point of view: Since raw data is cut up and rearranged, not only is the principle of immutable raw data violated, but maybe even more problematic the raw data format is used for something that is not actually raw data anymore and which really should be handled by some kind of meta data format. Adaptators took care to not accidentally put these “artificial raw data files” into the Data Lake (they obviously do not belong there since they are not NIFs recorded by an actual device), but since they are—by filename and extension—indistinguishable from actual raw data, it would be hard to track this mistake were it to happen.

This situation allowed me to put one of the Jupyter Notebooks I designed as part of this project into use.

The 1 TB field data was split into 10 different directories based on the recording device and adaptation specialists wished that this be preserved (since the faultiness could of course be dependent on the recording device). Therefore I processed all 10 directories separately and generated automatically annotated NoteSets for all Category 1 (rejects), Category 2 (counterfeits) and Category 3 (suspected counterfeits). Category 4 (fit notes) were not of interest in this scenario. This reduced the data load, since the majority of notes were Category 4. Simply by looking at the generated NoteSets, one could see that the problem was concentrated on one type of banknote, MYR 50 d, and one particular security feature (IR_Feature). This observation already narrowed down the domain of the problem. Finally the NoteSets were converted into Notelists and given to adaptation specialists for further analysis in MCM.

¹The annotation reads as follows: BOB 20 Emission a, orientation 1, type 1 (there is only one type in this case), Category 4 (genuine note), Fitness Flag TAPE is set (there is likely a tape area on that note). Such a set could be interesting when analyzing or investigating a tape recognition algorithm, or changing parameters connected to it in a CDF.

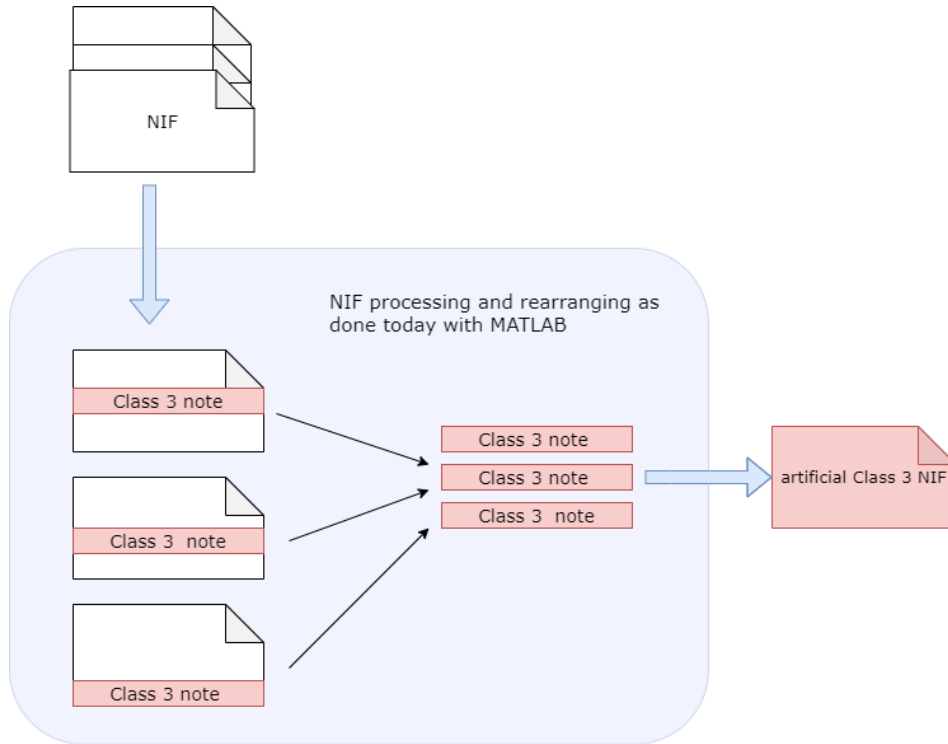


Figure 5.1: Obtaining specified data sets for critical MYR data by splitting binary data as done today

MYR_0_X_X_X_CL1_RR(REJ_ALGORITHM_NOT_SUCCESSFUL).ns	26.05.2021 09:19	NS File	3 KB
MYR_0_X_X_X_CL1_RR(REJ_ANGLE_TOO_LARGE).ns	26.05.2021 09:21	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_INTER_NOTE_GAP_TOO_SMALL).ns	26.05.2021 09:19	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_ISO_DISCR_FAILED).ns	26.05.2021 09:05	NS File	12 KB
MYR_0_X_X_X_CL1_RR(REJ_MECH_INFO_INVALID).ns	26.05.2021 09:22	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_NOTE_OUT_OF_WINDOW).ns	26.05.2021 09:22	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_NOTE_TOO_LONG).ns	26.05.2021 09:22	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_NOTE_TOO_SHORT).ns	26.05.2021 09:19	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_NOTEPROCESSING_ABORTED).ns	26.05.2021 09:15	NS File	13 KB
MYR_0_X_X_X_CL1_RR(REJ_OVERLOAD).ns	26.05.2021 09:22	NS File	1 KB
MYR_0_X_X_X_CL1_RR(REJ_SIG_OVERFLOW).ns	26.05.2021 09:22	NS File	1 KB
MYR_5_c_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:22	NS File	1 KB
MYR_5_d_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:22	NS File	1 KB
MYR_10_d_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:21	NS File	2 KB
MYR_10_d_X_X_CL3_SF(Magnet).ns	26.05.2021 09:21	NS File	1 KB
MYR_20_d_X_X_CL3.ns	26.05.2021 09:22	NS File	1 KB
MYR_20_d_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:20	NS File	1 KB
MYR_20_d_X_X_CL3_SF(IR_Thread).ns	26.05.2021 09:19	NS File	1 KB
MYR_50_d_X_X_CL1_RR(REJ_NOTEPROCESSING_ABORTED).ns	26.05.2021 09:22	NS File	1 KB
MYR_50_d_X_X_CL2_SF(IR_Feature&Composed).ns	26.05.2021 09:22	NS File	1 KB
MYR_50_d_X_X_CL2_SF(IR_Feature).ns	26.05.2021 09:09	NS File	1 KB
MYR_50_d_X_X_CL3.ns	26.05.2021 09:21	NS File	1 KB
MYR_50_d_X_X_CL3_SF(Composed).ns	26.05.2021 09:04	NS File	3 KB
MYR_50_d_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:09	NS File	29 KB
MYR_50_d_X_X_CL3_SF(IR_Thread).ns	26.05.2021 09:20	NS File	4 KB
MYR_50_d_X_X_CL3_SF(Magnet).ns	26.05.2021 09:22	NS File	1 KB
MYR_100_c_X_X_CL2_SF(IR_Feature&Composed).ns	26.05.2021 09:22	NS File	1 KB
MYR_100_c_X_X_CL3_SF(Composed).ns	26.05.2021 09:21	NS File	1 KB
MYR_100_c_X_X_CL3_SF(IR_Thread).ns	26.05.2021 09:22	NS File	1 KB
MYR_100_d_X_X_CL2_SF(IR_Feature&Composed).ns	26.05.2021 09:22	NS File	1 KB
MYR_100_d_X_X_CL3.ns	26.05.2021 09:21	NS File	1 KB
MYR_100_d_X_X_CL3_SF(Composed).ns	26.05.2021 09:22	NS File	1 KB
MYR_100_d_X_X_CL3_SF(IR_Feature).ns	26.05.2021 09:15	NS File	2 KB
MYR_100_d_X_X_CL3_SF(IR_Thread).ns	26.05.2021 09:07	NS File	5 KB
MYR_100_d_X_X_CL3_SF(Magnet).ns	26.05.2021 09:22	NS File	1 KB

Figure 5.2: Automatically generated and labeled NoteSets

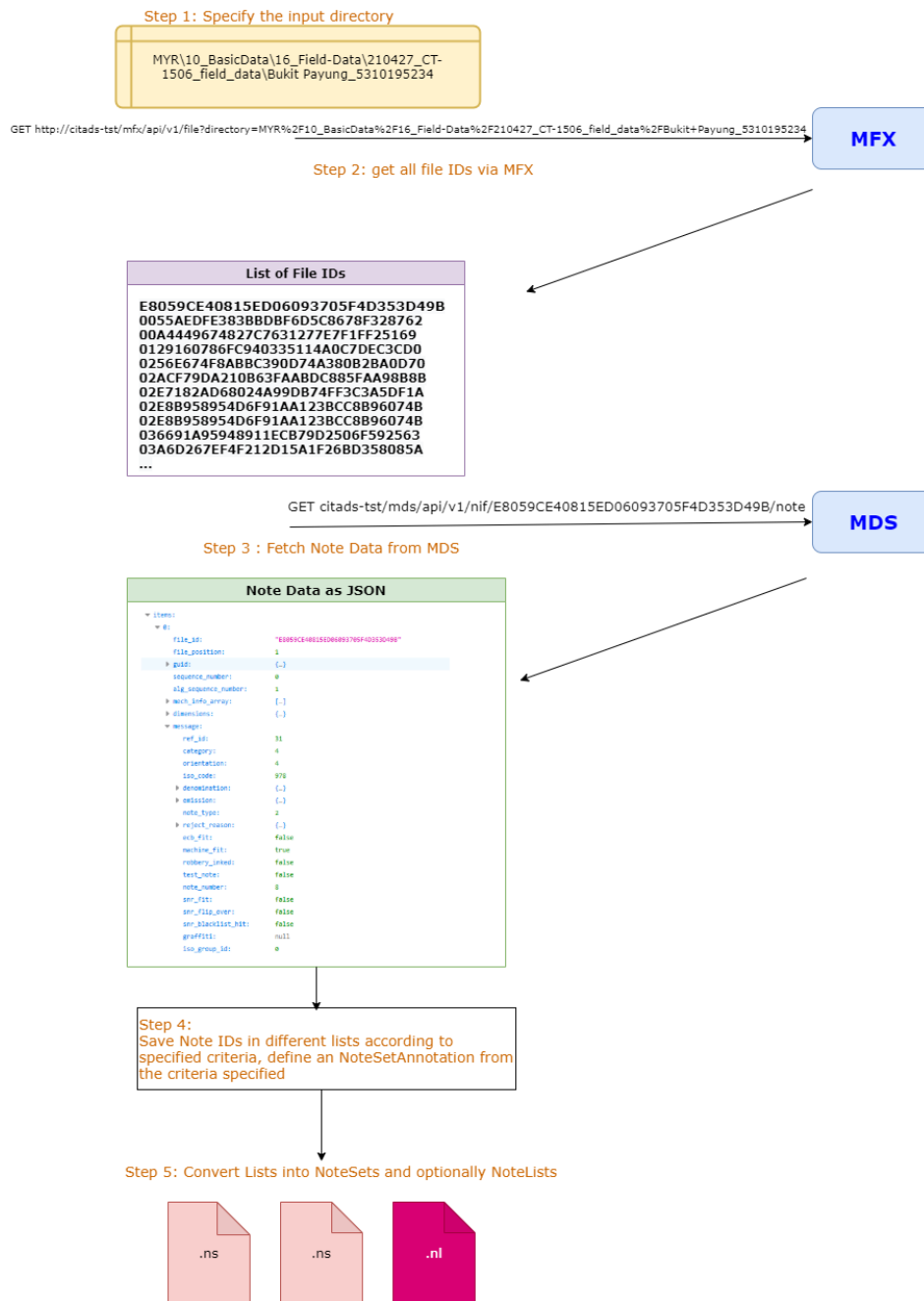


Figure 5.3: Obtaining specified data sets for critical MYR data with the help of Move Data Services MFX and MDS as proposed in this project

6 Conclusion / Results

6.1 Introduction

In the scope of this project I introduced a new Data Container for storing collections of banknote recordings and proposed different automated workflows for generating such NoteSets. I applied these workflows to some real-life use cases that came up in the daily work of the currency adaptation team. In the following section I will present my conclusions and results.

6.2 Conclusions on introducing the NoteSet data container

- By introducing the NoteSet data container a foundational element for the process of Data Collection and Preparation could be established. This project was concurrent to other developments in the Adaptation Data Management and a general shift towards realizing the importance of a “note annotation”, which ideally should be:
 - independent of the label/classification in the NIF: The classification done by the reader is CDF and firmware dependent and sometimes these very elements are under testing.
 - independent of the recording tool annotation¹. This is because a. annotational mistakes could have been made at the time of recording and b. field data generally were not recorded using a dedicated recording tool and are lacking this annotation altogether.
- An advantage of the filename-based NoteSet annotation introduced here is that it is very flexible and comes with little maintenance cost (no additional database for managing annotations is needed, reference NoteSets could be maintained as ASCII files in a simple Git repository). Additionally, the set-based annotation introduces the concept of data equivalence classes, which shifts the focus away from a set-in-stone annotation for each note towards annotation being something more flexible, namely a collection of mappings between a nominal label and a set of elements.
- NoteSets are small, readable and can be shared easily, even via e-Mail or copy-pasted into a JIRA-Issue. They address the problem of impeded readability and comparability that Notelists have.
- NoteSets lend themselves extremely well to the Microservice Toolchain and can be generated very fast from MDS and MFX. There is even an ongoing discussion of improving this compatibility by introducing dedicated NoteSet endpoints on MDS (cf. Section 6.5)

6.3 Conclusions on the automated NoteSet generation using Jupyter Notebooks

- The Python Jupyter Notebooks for automated NoteSet Generation provide a way to quickly and flexibly assemble MCM-loadable NoteSets from the MDS API, according to user-specific criteria. While technically this code does not use any high-level algorithms, it has proven to be valuable in a hands-on use case already in the course of this project. Its strength is not algorithmic innovation but rather the fact that it tries to shape a process for something that previously wasn't consciously seen as a process at all on the one hand, and the automation of actions which manually are very cumbersome and time-intensive on the other.

¹Inhouse banknote recording tools enable the user to define one annotation per recorded bundle. This information is then stored in a dedicated Chunk called NOTEINFO in the NIF file. However field data i.e. data recorded on an in-production reader say at a National Bank, does not have this additional annotation chunk

- Finding a way to flatten the note object from the MDS API into a pandas DataFrame opened up the possibility to use standard Python frameworks used in Data Science contexts on the note data such as pandas, NumPy or SciPy. This also allowed to bring statistics into play, something that on the level of data collection had not been done before, e.g. generate an MCM-loadable NoteSet of outliers for a particular algorithm result or measured note dimension, using the *z-Score*.
- By intersecting NoteSets from MDS and MFX the knowledge gap between the two services, namely MDS not knowing about the Data Lake's internal directory structure, could be closed.
- The proposed process of using the MOVEmSimulator to generate MCM-loadable NoteSets constituted an important step for the future Data Collection process because it provides an efficient way for CDF regression testing (cf. Section 6.5)

6.4 Challenges encountered

- Users experienced substantial performance problems when working with automatically generated NoteSets in MCM. The reason for this is the MCM architecture which does not allow the lazy loading of individual notes: Normally, when loading a Notelist in MCM, usually 100 Notes per NIF are loaded. So the NIF to note ratio of a Notelist is about $\frac{1}{100}$. This is because the way users normally generate Notelists is that they export a selection of complete NIFs into one Notelist. However, the way I introduced NoteSets to the users was in the context of generating tailored data sets for them, selecting dedicated notes based on certain criteria from the entire Data Lake. Unsurprisingly, there is usually not an entire NIF with notes satisfying the criteria specified by the user. Taking the example of finding Category 3 (suspected counterfeit notes), there are normally at most 1 or maybe 2 such notes in a NIF with 100 notes. As a consequence, a set of 100 Category 3 notes would be expected to originate from a similar amount of NIFs, so the NIF to note ratio in the resulting set would be close to 1. Since Notes within a NIF cannot be loaded lazily in MCM, loading the 100 notes in our hypothetical Category 3 set would mean loading and parsing close to 100 NIFs, as opposed to 1 NIF for loading 100 notes in a standard Notelist. This not only means a much higher loading time, but also, it seems that MCM is caching unnecessary image data from all the NIFs loaded, which results in high memory usage.

It's worth to point out that this problem is of course not a fault in the NoteSet data container – we would run into the exact same issues had we stored the 100 Category 3 notes in a Notelist. Rather it's an issue that likely has just not occurred before, because there had previously not been any way or process to generate and load tailored data sets which aggregate notes from a large number of individual containers (i.e. with a NIF to note ratio close to 1). On the side of application software developers, this problem impressively showed up the necessity to adapt the MCM architecture to allow lazy loading of notes with the help of MDS.

- At the beginning I was hopeful to use the algorithmic framework of MCM with the help of an existing MCM Python wrapper library called *mcmp*. My vision was to use these algorithms in a modular way, i.e. just run one chosen algorithm, say the algorithm for Tape detection, over a dataset and then process the data further based on the results. However this proved to be infeasible due to limitation of the Python wrapper library and the complexity of MCM: The wrapper library only contained a limited number of the MCM functionality and using it in Jupyter Notebook turned MCM into a total blackbox. I therefore had to discard this approach and focused on the MOVEmSimulator, which was itself written in Python and could be scripted directly in Jupyter, at the price of not having the individual available in a modular, "plug-in" kind of way.

6.5 Outlook

The following points are currently under discussion for development in a next stage:

- Establish the NoteSet data container across the toolchain, specifically in a first step: allow the loading of NoteSets in the MOVEmSimulator. Currently, the MOVEmSimulator allows the loading of NIFs and Notelist files. Allowing the loading of NoteSets would technically not be a very difficult change and could be an incentive for users to use this data container in their daily work.

- Introduce new endpoints on the MDS API which would allow to get notes directly as NoteSets i.e. directly as a list of just Note IDs instead of a list of entire JSON note objects. This would result in smaller HTTP payloads and a smaller number of requests, and thus allow to generate NoteSets even faster.
- Version controlled NoteSets in the adaptation team. This is in my opinion a very promising idea. Different reference NoteSets could be managed in a Git Repository (one repository per currency), together with the CDF under development. As the CDF changes between releases, the NoteSets might change as well: a note that was formerly a Category 3 note could now suddenly be in the Category 2 set. These changes could be very easily traceable using Git, since the NoteSet uses an ASCII format. So one could see immediately which adjustment in the CDF caused a note to land in a different set.

As of today, CDFs are not developed using any kind of version control system. One reason for this is that being binary files they don't really lend themselves to be maintained with Git, another one a general hesitance on the side of the adaptation team to introduce Git in their workflows. However, introducing Git for CDF development is intensely discussed at the moment and from this it would only be a small step to introduce NoteSet-based reference data sets as well.

- CDF regression tests using NoteSets generated by the MOVEm Simulator. Using the *pynoteset* library, NoteSets can be compared very fast. One area of application of this is the testing of two different CDFs using the same reader and firmware. The idea is as follows: The user chooses a reference CDF(*REF*) and a "device under test" CDF(*DUT*), as well as a set of NIF files. The NIFs are then processed by the Simulator and the respective NoteSets generated (a procedure that this project proposes and which is set out in one of the Jupyter Notebooks). NoteSets with the same annotation then could be compared quickly and we could answer questions such as: "Which notes are in the AUD_5_a_1_1_CL4_FF(Tape) ² for *REF* and *DUT* respectively and how do these sets differ? "

²The annotation reads as follows: genuine AUD (Australian Dollar) 5 a notes, orientation 1, for which the TAPE flag is set

List of Figures

3.1	Content of NOTEMSG2 Chunk as displayed in a proprietary ChunkFormat parsing software	6
3.2	MCM uses MFX for lookup of file paths in the Data Lake	7
3.3	MFX and MDS architecture	8
3.4	Extract of a Notelist file	9
4.1	Extract of a generic NoteSet file	12
4.2	Extract of a NoteSet file using abbreviations and comments	12
4.3	Class Diagram of <i>pynoteset</i>	13
4.4	NoteSet Annotation Examples	15
4.5	NoteSet Loading Viewer in MCM	16
5.1	Obtaining specified data sets for critical MYR data by splitting binary data as done today	20
5.2	Automatically generated and labeled NoteSets	20
5.3	Obtaining specified data sets for critical MYR data with the help of Move Data Services MFX and MDS as proposed in this project	21