



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: 17.04.2021

Management Summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus scelerisque, leo sed iaculis ornare, mi leo semper urna, ac elementum libero est at risus. Donec eget aliquam urna. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc fermentum nunc sollicitudin leo porttitor volutpat. Duis ac enim lectus, quis malesuada lectus. Aenean vestibulum suscipit justo, in suscipit augue venenatis a. Donec interdum nibh ligula. Aliquam vitae dui a odio cursus interdum quis vitae mi. Phasellus ornare tortor fringilla velit accumsan quis tincidunt magna eleifend. Praesent nisl nibh, cursus in mattis ac, ultrices ac nulla. Nulla ante urna, aliquet eu tempus ut, feugiat id nisl. Nunc sit amet mauris vitae turpis scelerisque mattis et sed metus. Aliquam interdum congue odio, sed semper elit ullamcorper vitae. Morbi orci elit, feugiat vel hendrerit nec, sollicitudin non massa. Quisque lacus metus, vulputate id ullamcorper id, consequat eget orci .

Versions

Version	Date	Status	Remarks
0.1	01.08.2013	Draft	Lorem ipsum dolor sit amet
0.2	21.08.2013	Draft	Phasellus scelerisque
0.3	02.09.2013	Draft	Donec eget aliquam urna. Lorem ipsum dolor sit amet
1.0	26.01.2014	Final	Lorem ipsum dolor sit ametPhasellus scelerisque, leo sed iaculis ornare
1.1	31.01.2014	Correction	Layout changed
1.2	07.02.2014	Addition	Chapter 1.1 extended

Contents

Management Summary	i
1. Introduction	1
1.1. Introduction	1
2. Project Goals	2
2.1. Initial Position	2
2.2. Project Goals	3
3. Project Methods	5
3.1. Introduce Note Sets as a new note data container	5
3.2. Implement Processes for Automated NoteSet Generation	5
3.3. Introduce NoteSet loading in MCM	6
4. Technical Background	8
4.1. Introduction	8
4.2. The .NIF File	8
4.3. MOVEm File Index	8
4.4. MOVEm Data Service	9
4.5. MOVEm Simulator	9
4.6. MOVE Currency Modeller	10
5. Note Set Data Container Specification	11
5.1. Note Set Specification	11
5.2. Note Set Annotation Specification	13
6. Automated generation of Note Sets	16
6.1. Introduction	16
6.2. Technologies used	16
6.3. Exemplary Notebooks	16
6.4. A practical application - analyzing a suspicious high rate of possible counterfeits for Malaysian Ringgit (MYR)	18
7. Conclusion / Results	21
7.1. Introduction	21
7.2. Conclusions on introducing the NoteSet data container	21
7.3. Conclusions on Tools and Technologies used	21
7.4. Outlook	22
Glossary	23
List of figures	24
APPENDICES	25
A. The pynoteset library	25
A.1. The NoteSet class	25
Index	25

1.Introduction

1.1. Introduction

1.1.1. Purpose of this document

This document serves as a documentation for the project “Data Collection and Preparation for Currency Adaptation” which is carried out in the scope of the BFH module “Project 2” BTI7302.

The client of the project is the company CI Tech Sensors AG (subsequently referred to as CI Tech) and the project is supervised on the part of CI Tech by Lukas Burger, head of application software.

The project is supervised on the part of BFH by Prof. Mascha Kurpicz-Briki.

1.1.2. Vision

In the scope of this project a process for the data collection and preparation for the currency adaptation workflow should be developed. Namely, 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 can be generated automatically from the centralized Data Pool, they are version controlled and a uniform process for their updating and management will be implemented. With this change, it will be possible to directly track the influence a new banknote recording has on an existing currency recognition software. Also it will be possible to use dedicated sub-data sets for training specific algorithms, for example data sets consisting of banknotes with a dogear.

2. Project Goals

2.1. Initial Position

2.1.1. Background

The 80/20 rule of Data Science states that most data scientist 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 has over 30 years experience in the field of Banknote Image recognition. One of its core competences is the development of currency adaptation software. Namely the so-called Currency Data File is a software, which is loaded onto the banknote reader, and enables it to identify and classify banknotes for specific currencies, as well as determine their fitness and authenticity. 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 huge data pool 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 pool, 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 pool 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, an 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 file. Adaptations 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.

2.1.2. Overview of the existing CI Tech adaptation toolchain

In this section a short overview is given over some of the tool in the existing CI Tech adaptation toolchain. This list is by far not exhaustive and only names the tools which are not relevant to the project at hand

- **MCM** (MOVE Currency Data Modeller) is a desktop application and the main tool of currency adaptation specialists in their daily work. It is used to develop, test and adapt so called *Currency Data Files*, the software which is loaded onto the banknote reader and enables it to recognize and classify banknotes. MCM is a software monolith that has grown and extended its functionality over decades, but it doesn't lend itself very easily to modularization. MCM is also the interface to the entire algorithmic framework. Recordings of banknotes can be loaded from the filesystem as .NIF files (binary files) or .nl (XML Format) files. The latter also allows loading notes from the central Data Pool via MFX instead of the local file system.
- **MFX** (MOVE File Index) is a web based microservice that allows the lookup of files from the central Data Pool. Namely, it allows to retrieve the storage path in the pool for a given File ID. Using this service, MCM can load files by File ID instead of a file path.

- *MDS* (MOVE Data Service) is a web based microservice that allows the lookup of note and image data. It allows to query the most relevant note data found in the (binary) .NIF files as well as images via REST api. A future vision which is in ongoing development is that entire sets of notes could be loaded into MCM via this REST interface instead of parsing large amounts of binary data.

2.1.3. Stakeholders

- The Head of Application Software will supervise the project and provide technical advice
- The Head of Currency Adaptation will provide additional support from the perspective of Currency Adaptation specialists.
- The Currency Adaptation team will be the primary user group

2.2. Project Goals

2.2.1. Accurate, consistent and complete reference data sets

The goal is to have accurate and consistent data sets, which are generated and persisted in a way that makes use of the Microservice based MOVEm Data Management Toolchain. Namely these test sets should fulfil the following criteria:

- Accuracy: Data in test sets has been checked for correct labeling and or given a label in case it was missing.
- Consistency: Different adaptations for the same currency should use the same data sets.
- Completeness: In the generation of reference note data sets, all data in the pool should be considered.

2.2.2. Mechanism for subclassifying Data Sets according to context-dependent criteria

It should be possible to generate data sets according to user specific criteria like: "All EUR notes that were classified as fake", "All USD notes for which tape was detected". A process should be installed which allows specialist users to generate such datasets from queries. These datasets will have to fulfil the same criteria as the reference data sets.

2.2.3. Automated process for generating and maintaining Reference Note Data Sets

A process has to be defined for generating Reference Data Sets automatically as well as updating and expanding them when new data is added to the Data Pool. A process has to be defined for generating Reference Data Sets automatically as well as updating them when new data is added to the Data Pool. Part of this process is also the management of Reference Data Sets in a Version Control System.

2.2.4. Technical Analysis of the existing Toolchain

To achieve the previously listed goals, the introduction of a new data container will be necessary to make the existing toolchain compatible with this new paradigm. While this shift can only happen over a longer period, it would be very undesirable to have a new data container or format which will only be deployable for a part of the toolchain. The goal is therefore to have an assessment on the feasibility of implementing the new data container into the existing toolchain.

2.2.5. Introduction of Data Science Workflows in the company

Another less technical goal, is to familiarize users with certain Data Science workflows. This entails showing by presenting this project how the existing microservice can be used to gain knowledge about note data and generate data sets, as well as familiarize expert users (who might want to use similar workflows to generate their own data sets) with the Jupyter Notebook framework.

3. Project Methods

3.1. Introduce Note Sets as a new note data container

We will introduce Note Sets 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 in the narrower sense, but rather a new paradigm as how to think of collection of notes: No longer as notes that were recorded in the same transaction and thus in the same binary file (in earlier times this was the only way one could think of collections of notes), but as a collection of IDs.

The note set data container will have the following advantages:

- Compactness: Note Sets will be minimalist in the sense that notes are references by ID only. There is no additional information like annotation stored in the note set format.
- Independency of file locations: With the introduction of MFX the actual location of a file in the pool becomes redundant. Therefore, the note set does not store any file paths.
- Equivalence with respect to annotation: Unlike the Notelist format, Note Sets will no longer have an individual annotation per note. Rather, they have at most one annotation per note set, which is to be persisted in the note set's filename. Thus, note sets 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. The reasons for that are that a) Noteset import is supported by MCM however in a yet limited way. b) the Python MCM wrapper library mcmp, which is used to load notes and run algorithms from outside MCM, does currently not support the loading of Notesets.
- Differentiability. 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.

The Note Set 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 python toolchain, including MDS and MFX, 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.

3.2. Implement Processes for Automated NoteSet Generation

Using different methods and for a set of exemplary currencies a first version of NoteSets are generated from the Data Pool, In the course of this existing client software of MDS and MFX are extended to allow for more complex queries.

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 include it in the adaptation specialist's workflow.

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

1. Crawl MFX for a chosen currency to obtain all file paths of .NIF files for that currency
2. With a list of all FileIDs and corresponding filepaths, use a labeling method to label all the notes in all the files
3. Based on the obtained labeling, split the set of all Notes into labelled NoteSets (within a NoteSet, all elements have the same label)

It remains thus to choose a labeling methods. I extracted the following four methods which can be outlined as follows:

1. For each note in the set of all notes, query MDS to obtain the recognition and classification done by the reader for this note. We use the banknote reader (and the CDF which was loaded at the time of recording) as the labeller. 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 reader on the chosen set of notes and a chosen .CDF. Use the information found in the .NIFs generated by the Simulator 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 classifies 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. Load all the notes from the set of all notes into MCM and use algorithm(s) from the algorithmic framework of MCM to determine properties of each note based on which the notes can be labelled. We use MCM as the labeller. To do this in an automated way, a Python wrapper for MCM has to be used, which unfortunately only allows restricted use of MCM.
4. Use an algorithm external to MCM as a kind of plug-in to process and label the notes. This is the most experimental method, but maybe also the most interesting one. It has the advantage that we are free of MCMs monolithic architecture in our labeling process, but also the challenge that the MCM algorithmic framework is highly developed and refined and can hardly be bested by an external classifier.

Figure 3.1 to 3.4 visualize these different approaches.

In the course of implementing the project the focus shifted almost entirely on the method of labeling via MDS and the MOVEm Simulator. 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.

The usage of a plugin algorithm is still a promising option especially if one would like to challenge an existing algorithm (e.g. classifier) by comparing its result with a generic image classifier. It turned out however that the users' main need at this point in time is not to gather new knowledge from the existing data but rather to just find ways of generating useful datasets. Given that the need of introducing a new algorithm was less urgent.

3.3. 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 a feature is added in MCM which 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.

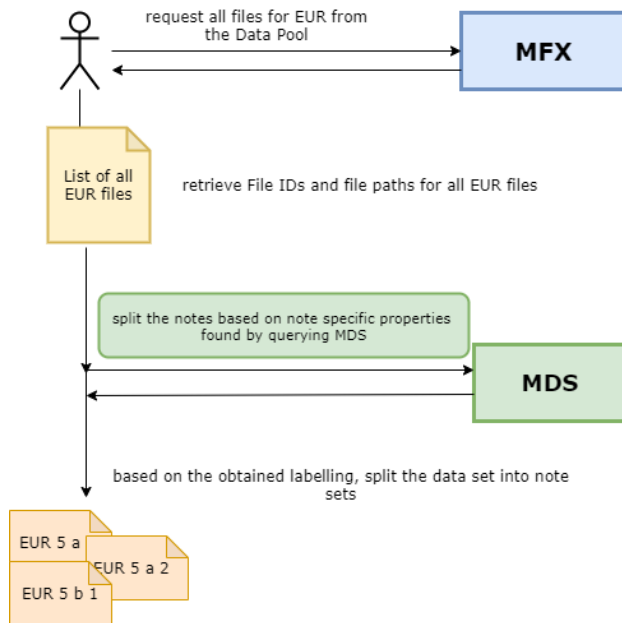


Figure 3.1.: NoteSet Generation with MDS labeling

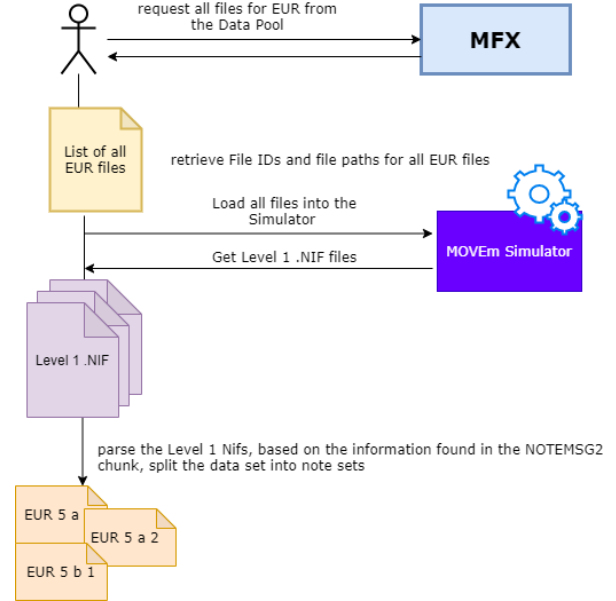


Figure 3.2.: NoteSet Generation with MOVEm Simulator labeling

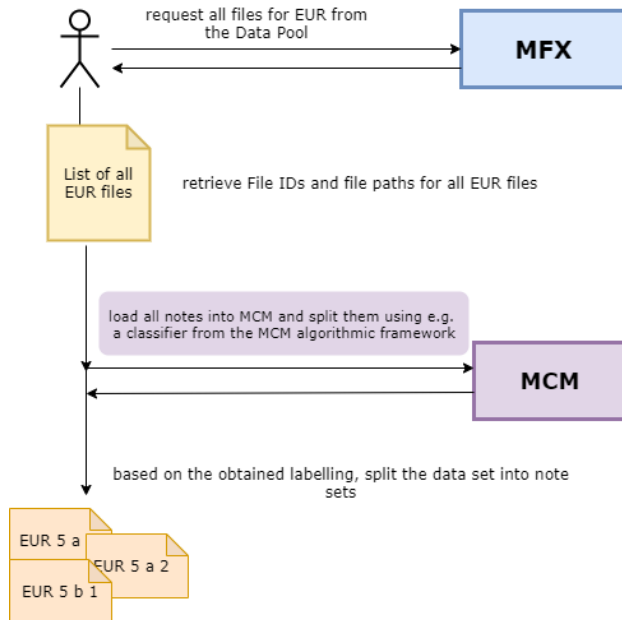


Figure 3.3.: NoteSet Generation with MCM labeling

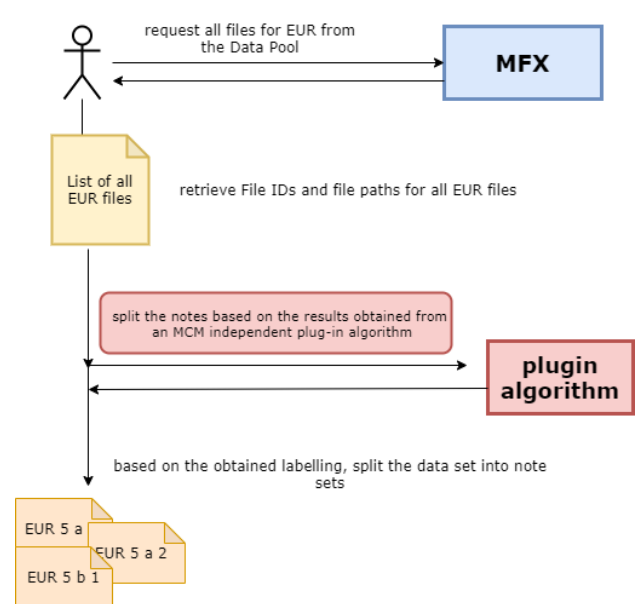


Figure 3.4.: NoteSet Generation with Plugin Algorithm

4. Technical Background

4.1. 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.

4.2. 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 processed data calculated by the reader. It uses the *BEB Chunk Format*, CI Tech's 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. 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 exactly once for each note recorded and which contains the most relevant information about a note's classification by the sensor. This information can then be used for labelling our dataset.

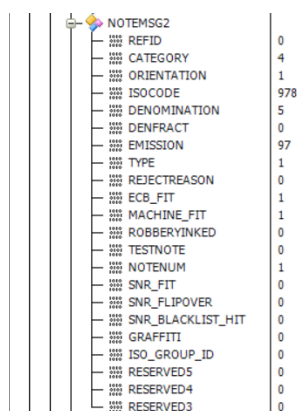
A NIF has a unique ID (the file's MD5 checksum), which is also called FileID. With the help of the FileID, the NoteID can be defined as the concatenation of the FileID and the note's¹ FilePosition, which corresponds to the order in which the note's were recorded. Thus

`8AD47001CC4BD7B49783B4D6125F9551 : 1`

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

4.3. MOVEm File Index

MFx (MOVEm File Index) is a central index of all files in the Data Pool, allowing the lookup of a file's path in the central Data Lake. 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 duplicate NIF files in the data lake and defect NIF files. Moreover, MFx is integrated into MCM, insofar as MCM has access to the MFx API and can thus load directly from the Data Lake. The following figure visualizes how MFx is used for path lookup within MCM.



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

Figure 4.1.: 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 *recordings of 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 NIF file.

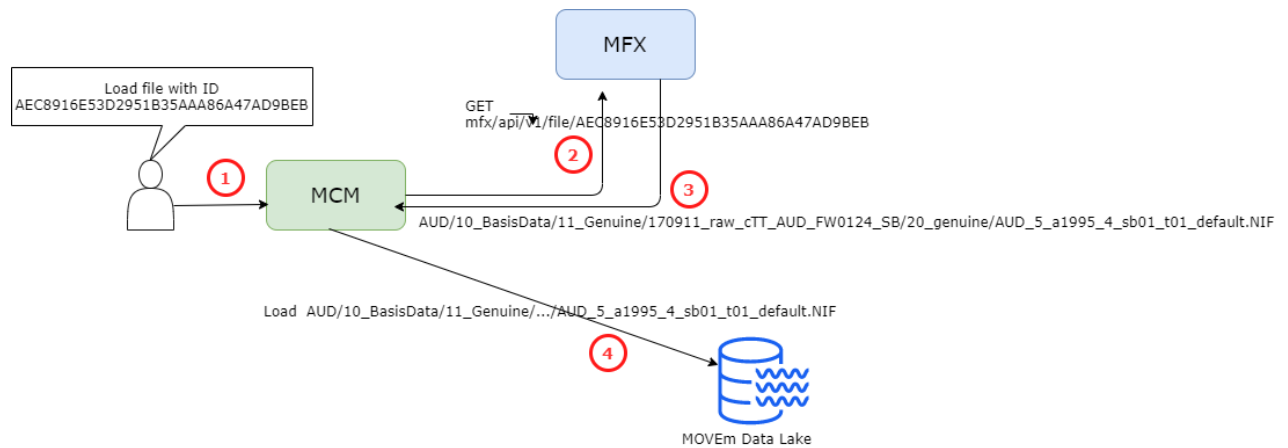


Figure 4.2.: MCM uses MFX for lookup of file paths in the Data Lake

4.4. 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 recorded by the reader).

Currently the MDS database stores about 322 GB of NIF data, whereas the Data Lakes stores about 6.67 TB of NIFs, which are about 1110714 (status March 2021) individual files. The MDS application also entails a NIF parsing module (movem), which on user request will crawl the entire MFX, parse each NIF file indexed in MFX and store the relevant information in the MDS database.

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 and only need to look at images in special cases. There are even NIF analyzing tools 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 lazy loading of images. However, to achieve this goal, the architecture of the majority of the toolchain would have to be substantially changed.

The following figure visualizes the MDS architecture, as well as the power of the MDS API².

4.5. 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 currency data files (.cdf files) 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).

²This image was created by Lukas Burger and he kindly let me borrow it for this documentation

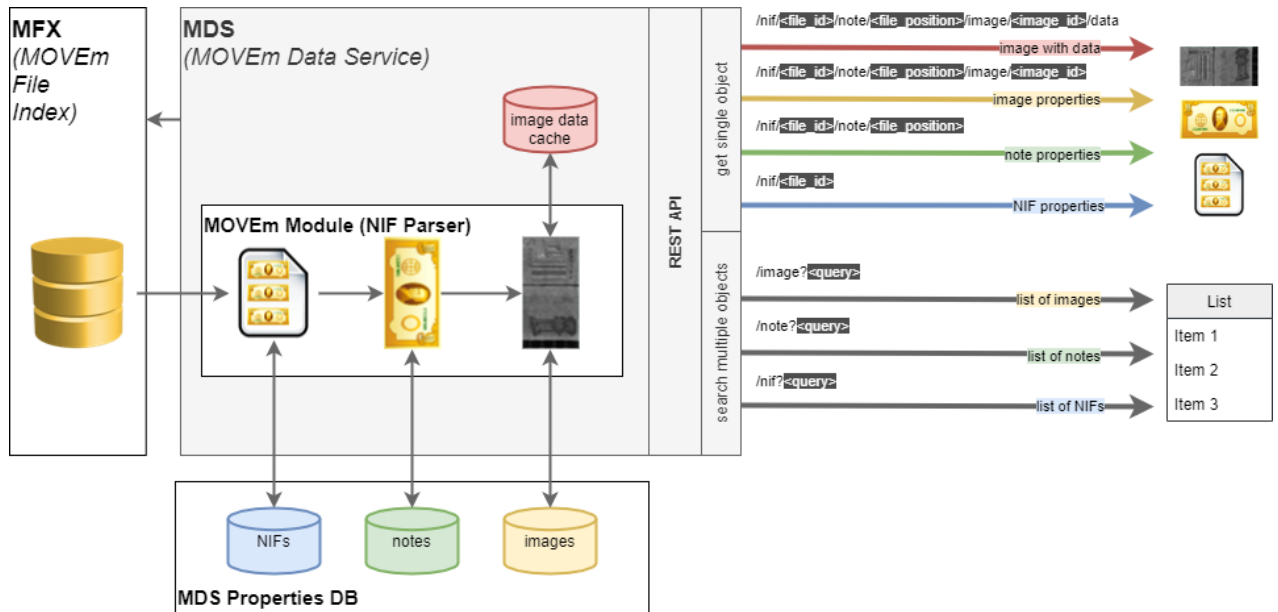


Figure 4.3.: MDS architecture and API

4.6. MOVE Currency Modeller

MOVE Currency Modeller (MCM) is the adaptation specialist's main tool which they use to create Currency Data Files (CDF), the currency recognition software for the banknote reader. Basically, users load raw data (NIF files) into MCM and use this data to develop and improve CDFs or parts thereof. A typical scenario where a CDF would need to be adapted is the release of a new banknote or the discovery of a new type of counterfeit which the CDF has to be able to handle. 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 monolith, 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.

5. Note Set Data Container Specification

5.1. Note Set Specification

5.1.1. Motivation

By introducing the Note Set 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 implies that this is the only correct way that data should and can be collected. The goal of introducing the Note Set is exactly the opposite: We want users to think of a data collection as exactly that: A collection of data elements, that at least for now does not come with a set of restrictions and rules of its own. In section Introduce Note Sets as a new note data container we listed the advantages and requirements for the new data container. In this section the steps undertaken to reach them will be specified.

5.1.2. Note Sets as equivalence classes

One distinguishing feature of Note Sets is that they will not have a per-note labelling but one labelling for the entire set, the Note Set 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 Note Sets as equivalence classes where the equivalence relation is defined by the Note Set's annotation. The Note Set Annotation is specified below in section Note Set Annotation Specification

5.1.3. Serialization

Since Note Sets are basically just a collection of IDs, we will store them as text files in ASCII format. Each line will ideally 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 FileId. 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 in the ASCII file, preceded by #
4. Any software with a Note Set export feature will for now for reasons of transparency use the generic way of just listing the individual NoteIDs and not use the abbreviated forms presented above

```
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 5.1.: Extract of a generic Note Set file

```
BC7CFE0DFC4285F8C0BDCB1AA108828B #this is a comment
AEC8916E53D2951B35AAA86A47AD98EB
8AD47001CC4B07B49783B4D6125F9551:1,2-3
5AEA7802F8629E6F8A31E727CE16C6F7:4 #here is another comment
8431CA32E68C1611E653CAA39FA538B:1-10
374DB8C13B9B1DB8C42C1B109DE3BBD5
006C140D69769451E20F63811D:68003
```

Figure 5.2.: Extract of a Note Set file using abbreviations and comments

5.1.4. 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 Filelds/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 $A \cap B$.
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. See the *pynoteset* class diagram figure 5.3 for reference. Furthermore, a converter class NoteSet2Notelistfile has been added to the library *notelistfile*, which allows the conversion of NoteSets to Notelistfile, the standard XML based format currently in use. This functionality is particularly powerful since a majority of the python toolchain already uses the Notelistfile model internally. This converter thus provides an interface for the NoteSet object to be used in other tools.

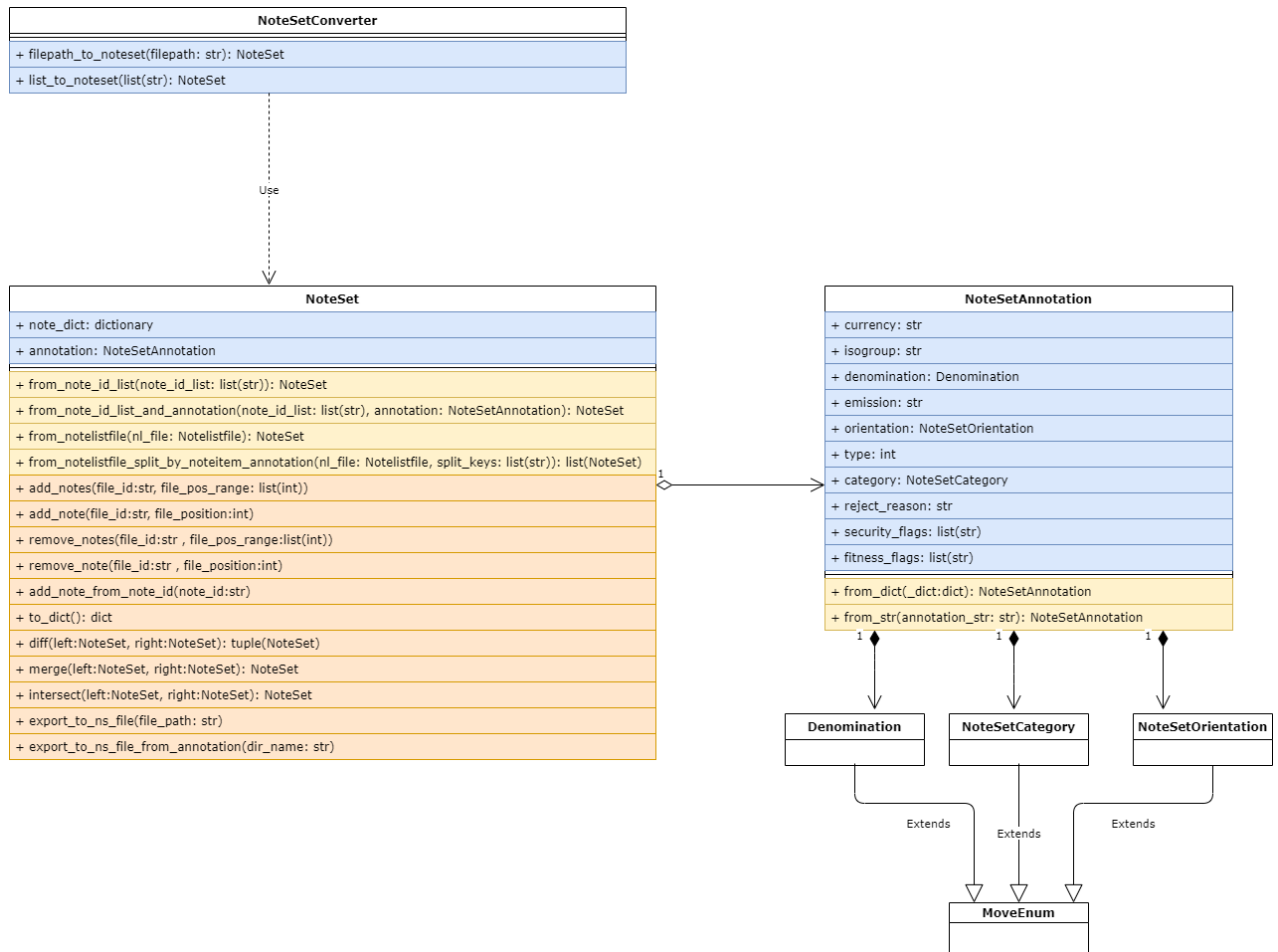


Figure 5.3.: Class Diagram of *pynoteset*

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 pip.

5.2. Note Set Annotation Specification

5.2.1. Motivation

As already mentioned, we introduce Note Sets as a container for notes which are equivalent with respect to a yet to be defined annotation, the labeling of the set. Since the Note Set itself is basically just a set of IDs or keys, all 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 Note Set. Since we do not want to introduce a new Note container format, 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 defined for it.

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

5.2.2. 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 Note Set's annotation should be as simple as possible, namely because we want to persist the annotation in a Note Set's 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 notes in MCM, the same criteria needed when mapping a loaded note to a reference note in a CDF. The optional properties (with the exception of comment) are criteria for which it is very likely that a user wants to split or filter a set of notes by when analyzing or training algorithms.

5.2.3. Mandatory Elements of the Note Set Annotation

The following attributes form the mandatory part of the Note Set'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 Note Set, corresponds to the value of the Chunk EMISSION in NOTEMSG2

Orientation Orientation of the notes in the Note Set, corresponds to to ORIENTATION in NOTEMSG2

Type NoteType (indicating the reference note this note is matched to by the CDF which was used in recording). For currencies like, GBP with different printers (for example HKD¹ or GBP²), this type is necessary to identify a note. For other currencies with only one printer, like CHF, denomination and emission is usually enough to identify a note.

5.2.4. Optional Elements of the Note Set Annotation

The following attributes are optional elements of a Note Set Annotation. They are not identifying properties but qualify a note (set).

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 which does not come from the NIF file

¹Hong Kong Dollar

²Great Britain Pound

5.2.5. Specifying the file format string

As already stated, the Note set's annotation is to be converted into a filename when exporting a Note Set 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 Not Sets could easily validate Note Set annotations independent of the programming language.
- Generic placeholder for non-specified mandatory properties: Since we allow Note Sets to be merged, we have to anticipate the scenario where e.g. a Note Set 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, 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 Note Set Annotation Examples illustrate the application of this regular expression, i.e. some valid Note Set Annotation file strings

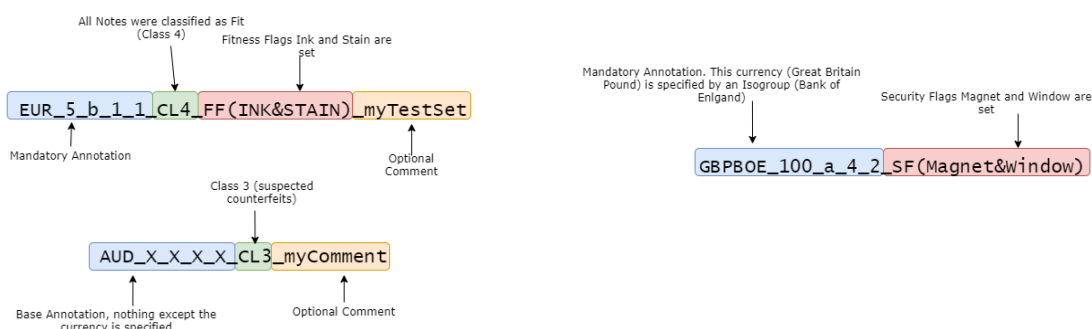


Figure 5.4.: Note Set Annotation Examples

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

6. Automated generation of Note Sets

6.1. Introduction

6.2. Technologies used

To develop an automated Note Set generation process, access to MDS and MFX services is of course required. To ensure availability of these services outside the CAPaaS-Cloud, a local Kubernetes cluster on which these services run was set up using *kind* (Kubernetes in Docker). Thus 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 Note Set generation software itself the decision was made to use Jupyter Notebooks in a first phase. These Notebooks will be enhanced and made more user-friendly using the *ipywidgets* framework and then given to the users. The reasons for this decision are as follows:

1. Adaptation specialists are expert users, technically speaking 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.
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.

6.3. Exemplary Notebooks

To acquaint users with the new data container and to start a dialogue with them about their needs when it comes to data set generation, I developed a number of exemplary Notebooks, each for a use case, that seemed realistic and typical. The use cases are the following:

6.3.1. Crawl a directory and generate NoteSets automatically

Splitting a chosen directory into NoteSets. The user has a directory of files which lie in the Data Pool and 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, this use case occurred and the Notebook designed for it could be put into practice (see below).

6.3.2. 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.

6.3.3. Perform Data analysis on a chosen Noteset

In this use case the user has an existing Noteset persisted in .ns file format or a .nl Notelist 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 of flattening 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. 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 the actual quantiles can be freely chosen.

6.3.4. 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 how already recorded notes would perform using 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 DataPool) 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.

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.

6.4. A practical application - analyzing a suspicious 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 Class 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 Class 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, Class 3 notes (see Figure 6.1). 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 Raw Data Pool (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.

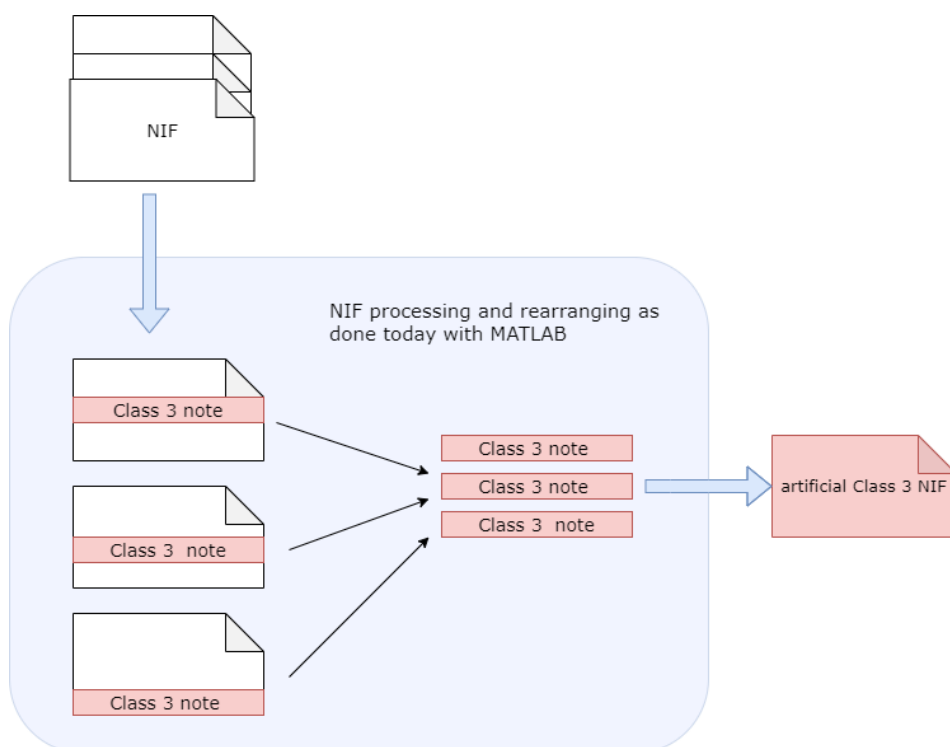


Figure 6.1.: Obtaining specified data sets for critical MYR data by splitting binary data as done today

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 adaptators 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 Class 1 (rejects), Class 2 (counterfeits) and Class 3 (suspected counterfeits). Class 4 (fit notes) were not of interest in this scenario. This reduced the data load, since the majority of notes were Class 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

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 6.2.: Automatically generated and labeled NoteSets

werde converted into Notelistfiles and given to adaptation specialists for further analysis in MCM.

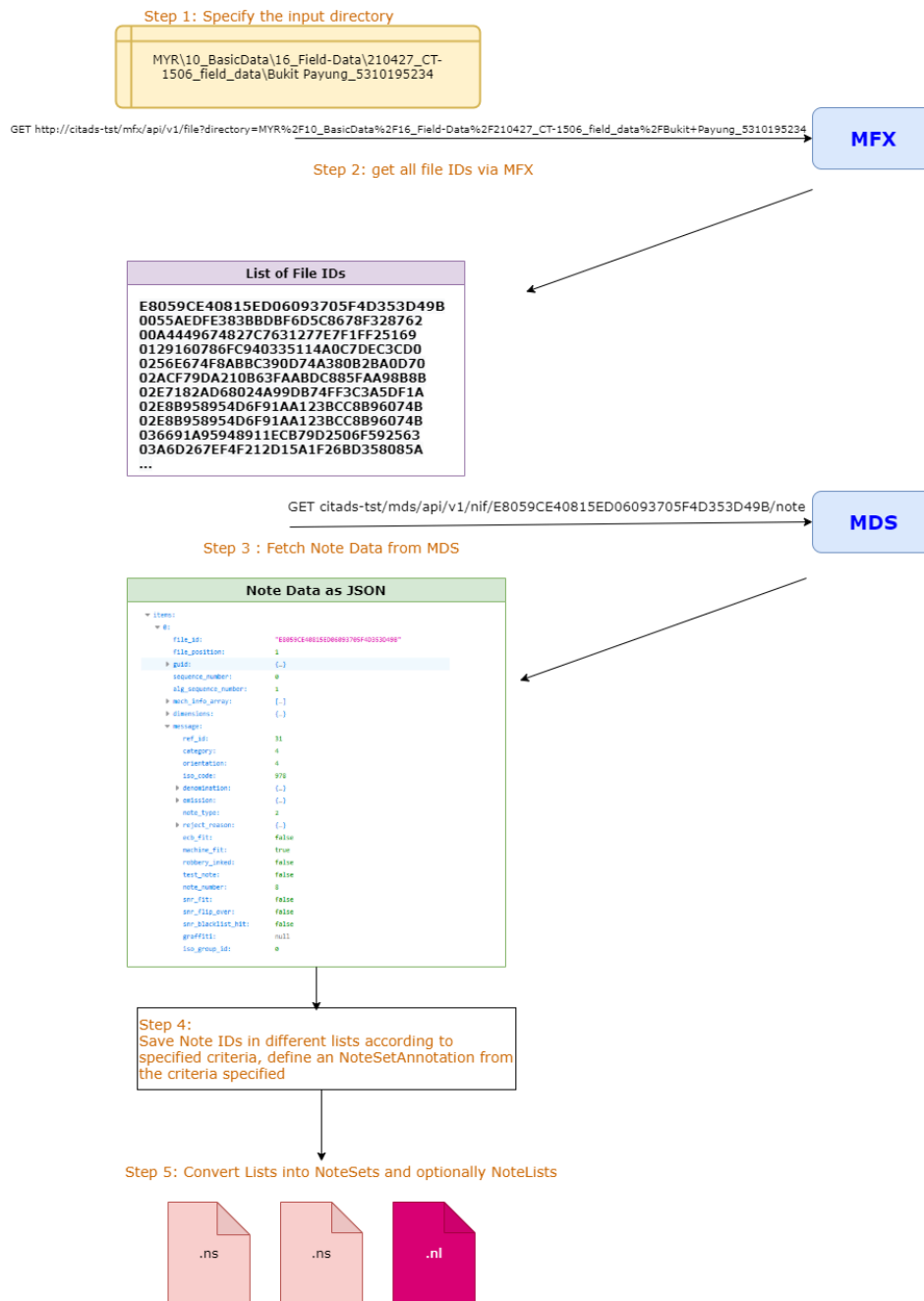


Figure 6.3.: Obtaining specified data sets for critical MYR data with the help of Move Data Services MFX and MDS as proposed in this project

7. Conclusion / Results

7.1. 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 of this undertaking.

7.2. Conclusions on introducing the NoteSet data container

The introduction of the NoteSet data container provided a step towards a reformed data management, which focuses on seeing note records as independent from the NIF file they were recorded in, a concept that has in the past been difficult to communicate to the users. The simplicity of this format provided an intuitive way to communicate this idea to the users, and the NoteSet generation workflows I could demonstrate in Jupyter Notebooks provided a tangible example of this concept's advantages, namely that one can easily generate NoteSets according to different criteria, as well as merge and compare them fast.

On a technical level this data container represents a simplification. Not only is it a lot smaller than the currently used Notelistfile format, but also there is no need to annotate the individual objects within the set. Rather an annotation can be given optionally to the entire set.

However, the users experienced substantial performance problems when working with general NoteSets in MCM. The reason for this is the monolithic 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 $\frac{1}{7}100$. This is because the way users 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 special Notes based on certain criteria from the entire Data Pool. However, 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 at most 1 or maybe 2 such notes in a NIF with 100 notes. Therefore 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 meant 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 if we had stored the 100 Category 3 notes in a Notelist. Rather, this problem has not occurred before, because there had previously not been any way or process to generate tailored data sets which were also loadable in MCM. On the side of application software developers, this problem reinforced the awareness of the necessity to adapt the MCM architecture to allow lazy loading of notes with the help of MDS.

7.3. Conclusions on Tools and Technologies used

The motivation for using Jupyter Notebooks was that it would offer a way for quick results independent of the actual toolchain. However, I found that working with Jupyter in a development mindset came with its own set of problems. First, I found it to be somewhat fragile. This was in large part due to the fact that I had to develop my code in the proprietary Cloud, where the MDS and MFX server run. For some reason probably due to limited authorization, I was unable to install any extensions in Jupyterlab. Therefore I always had to switch between Jupyter Notebook and Jupyter Lab to find ways to do certain things for example plot something or use the ipywidgets framework, because something might work in one of either but not in the other. Often, the virtual environment kernels I was using

would break in the course of switching between JupyterLab and Jupyter Notebook. Generally I was often left with the feeling of having to redo my entire setup between the days I was working on the Notebooks. I do think that Jupyter Notebooks were the right choice to try out different workflows for the NoteSet generation, also because they lend themselves to demonstration, but it might have been advantageous to implement the functionality in a small library or similar.

7.4. Outlook

The following next steps regarding the integration of the NoteSet data container in the toolchain are currently under discussion

- 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 (this functionality is already implemented in the pynoteset library) and could be an incentive for users to use this data container in their daily work.
- 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. After all a note is never say an "EUR 5 b Class 3" as such, but only because there was a particular CDF release once loaded onto the reader which classified the note to be an "EUR 5 b Class 3".

List of Figures

3.1. NoteSet Generation with MDS labeling	7
3.2. NoteSet Generation with MOVEm Simulator labeling	7
3.3. NoteSet Generation with MCM labeling	7
3.4. NoteSetGeneration with Plugin Algorithm	7
4.1. Content of NOTEMSG2 Chunk as displayed in a proprietary ChunkFormat parsing software	8
4.2. MCM uses MFX for lookup of file paths in the Data Lake	9
4.3. MDS architecture and API	10
5.1. Extract of a generic Note Set file	12
5.2. Extract of a Note Set file using abbreviations and comments	12
5.3. Class Diagram of <i>pynoteset</i>	13
5.4. Note Set Annotation Examples	15
6.1. Obtaining specified data sets for critical MYR data by splitting binary data as done today	18
6.2. Automatically generated and labeled NoteSets	19
6.3. Obtaining specified data sets for critical MYR data with the help of Move Data Services MFX and MDS as proposed in this project	20

APPENDICES

A. The pynoteset library

The pynoteset library was developed within the scope of this project. Its purpose is to generate and handle the newly introduced NoteSet data container. This small library can theoretically be deployed in the entire Python toolchain of application software at CI Tech. An analogous implementation in C# for the .NET toolchain is still pending

A.1. The NoteSet class

This is the base class representing a noteset object. A NoteSet represents a set of Note IDs. It can optionally contain an annotation property, which is of type NoteSetAnnotation.

A.1.1. from_note_id_list(note_id_list)

Description

Generates a NoteSet object from a list of Note IDs

Parameters

note_id_list A list of strings representing Note IDs.

A.1.2. from_note_id_list_and_annotation(note_id_list, annotation)

Description

Generates a NoteSet object from a list of Note IDs and a NoteSetAnnotation object.

Parameters

note_id_list A list of strings representing Note IDs.

annotation NoteSetAnnotation

A.1.3. from_notelistfile(nl_file)

Description

Generate a NoteSet object from a Notelistfile object, i.e. convert a Notelistfile into a NoteSet.

Parameters

nl_file Notelistfile

A.1.4. diff(noteset_1, noteset_2)

Description

Find the difference of two NoteSets. Return two notesets:

- NoteSet of all elements contained in noteset_1 but not noteset_2
- NoteSet of all elements contained in noteset_2 but not noteset_1

Parameters

noteset_1 NoteSet

noteset_2 NoteSet

A.1.5. merge(noteset_1, noteset_2)

Description

Merge two NoteSets into a new NoteSet

Parameters

noteset_1 NoteSet

noteset_2 NoteSet

A.1.6. export_to_ns_file(filepath)

Description

Save a NoteSet as ASCII file

Parameters

filepath string or path-like object