University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Standardized data formats for electrophysiological data

Pilsen 2021

Lukáš Ščurko

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Lukáš ŠČURKO**
Osobní číslo:	**A18N0100P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Standardizované datové formáty pro elektrofyziologická data**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s tzv. FAIR principy pro podobu a správu vědeckých dat.
2. Seznamte se s aktuální standardizačními aktivitami, formáty, datovými úložišti a open source nástroji pro popis a správu elektrofyziologických dat.
3. Na základě bodů 1, 2 a doporučení vedoucího vyberte vhodný(é) standard(y); dále vyberte a zprovozněte vhodné datové úložiště pro popis elektrofyziologických dat.
4. Navrhněte a implementujte transformační nástroje do datových standardů z bodu 3; převeďte do těchto standardů vybraná data z neuroinformatické laboratoře KIV/NTIS a uložte je v úložišti z bodu 3.
5. Výsledný kód a dokumentaci sdílejte ve veřejně dostupných repositářích tak, aby byly užitečné globální komunitě.
6. Zhodnoťte výsledné řešení.

Rozsah diplomové práce:	**doporuč. 50 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Ing. Roman Mouček, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání diplomové práce:	**11. září 2020**
Termín odevzdání diplomové práce:	**20. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 24. září 2020

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, 19th May 2021

Lukáš Ščurko

Abstract

This master's thesis describes standards for storing electrophysiological data in order to find a suitable standard for data conversion from the BrainVision standard, which is used in the neuroinformatics laboratory at the University of West Bohemia. The standards are examined from several points of view, on the basis of which the standard into which the existing data has been converted is selected. It was verified whether the standards meet the Fair principles. The complexity of data conversion and supporting libraries to the standards examined was also assessed. Furthermore, a tool is created that converts electrophysiological data from BrainVision standard to the selected standard. A suitable data storage was selected for the converted data, in which the data was stored.

Abstrakt

Tato diplomová práce popisuje standardy pro ukládání elektrofyziologických dat za účelem najít vhodný standard pro převod dat z BrainVision standardu, který je používán v neuroinformatické laboratoři na Západočeské univerzitě. Standardy jsou prozkoumány z několika hledisek, na základě kterých je vybrán standard, do kterého se převedla stávající data. Bylo ověřováno, zda standardy splňují Fair principy. Byla také posuzována složitost převodu dat a podpůrných knihoven k prozkoumávaným standardům. Dále je vytvořen nástroj, který elektrofyziologická data z BrainVision standardu převede do vybraného standardu. Pro převedená data bylo zvoleno vhodné datové úložiště, do kterého se data uložila.

Contents

1 Introduction

Nowadays, standards for working with electrophysiological data are constantly expanding and emerging. Several standards are constantly striving to become a worldwide standard for all types of electrophysiological data.

Research laboratories produce large amounts of electrophysiological data, but also produce them in various formats that are incompatible with each other, both in terms of data structures and metadata collected. In addition to the incompatibility between formats, the amount of information when collected is varied and not standardized. This means that for researchers who process this data, the amount of work is increasing to get a result. Recent research shows that the time spent by researchers in searching for and identifying multiple useful data sources can take up to 80% of their time dedicated to the project or research question itself [1]. This means large time losses and it is necessary to reduce these 80%. This is how it works in general and it also applies to the Laboratory of Neuroinformatics at the Faculty of Applied Sciences of the University of West Bohemia that performs EEG experiments. They produce data that is in the BrainVision format, uses the standard, but even so, there are huge time losses. It is important to reduce inefficiency. However, the data is stored in an outdated and too complex repository.

In this thesis I will analyze all available standards and also look at Fair principles, which form abstraction above standards. The Fair principles are described in the following chapter.There is an effort to define a standard for electrophysiology data and there have been several attempts now and in the history, with the fact that they already reflect the abstract level of Fair Principles to some extent. This means that the solution is to analytically arrive at a standard that will meet the abstract level of Fair Principles and will be usable locally for us.

I described the most widespread standardization initiatives and standards describing electrophysiological data in chapter 3.

The tools that can work with these standardization activities are described in the following chapter 4.

Repositories in which electrophysiological data can be stored are presented in chapter 5.

There is a need to shorten the time spent on rewriting data to another format, as I mentioned above. I created a tool in Python that converts the data from the BrainVision format in which the laboratory currently stores the data to one of the standards, specifically I chose the BIDS format. This format is described in section 3.3.

A python tool with a graphical user interface was created, which with the help of libraries converts data from the existing standard to the BIDS format. This tool is described in Chapter 7.

For datasets stored in the new standard, a suitable repository has been selected. The data was saved in this repository and a DOI file was created for it. The repository is described in chapter 8.

The tool was tested on selected datasets, which were collected at the Faculty of Applied Sciences at the University of West Bohemia. The quality of the code was checked by two static analyzers. Testing is described in chapter 9.

2 Fair principles

One of the recommendations for storing electrophysiological data with regard to their openness and the possibility of subsequent analysis are FAIR principles (Findability, Accessibility, Interoperability, Reusability) [2].

FAIR principles are not a standard or specification, but their main goal is that they can be a guide for those who want to increase the reuse of their data. They serve as principles to ensure mutual compatibility of stored data across different systems, allowing entities to share measured data with each other. [2]

FAIR principles determine the goals that should be achieved when designing a storage method [3]:

- **To be Findable:**
  - (Meta)data are assigned a globally unique and eternally persistent identifier.
  - Data are described with rich metadata.
  - (Meta)data are registered or indexed in a searchable resource.
  - Metadata specify the data identifier. [3]
- **To be Accessible:**
  - (Meta)data are retrievable by their identifier using a standardized communications protocol.
  - The protocol is open, free, and universally implementable.
  - The protocol allows for an authentication and authorization procedure, where necessary.
  - Metadata are accessible, even when the data are no longer available. [3]
- **To be Interoperable:**
  - (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
  - (Meta)data use vocabularies that follow FAIR principles.
  - (Meta)data include qualified references to other (meta)data. [3]

- **To be Re-usable:**
  - Meta(data) have a plurality of accurate and relevant attributes.
  - (Meta)data are released with a clear and accessible data usage license.
  - (Meta)data are associated with their provenance.
  - (Meta)data meet domain-relevant community standards. [3]

If a data source is intended to be FAIR, sufficient metadata must be provided to automatically identify its structure, provenance, licensing and potential uses, without having the need to use specialized tools. Moreover, any access protocols should be declared where they do or do not exist. The use of vocabularies and standard ontologies further benefit to the degree of FAIRness of a data set. [4]

In the recent extended explanation of what these principles really mean, the A in FAIR was redefined as "Accessible under well defined conditions". This means that data do not have to be open, but the data access protocol should be open and clearly defined. In fact, data should be "as open as possible, as closed as needed". [4]

The recognition that computers must be capable of accessing a data object autonomously was the core to the FAIR principles since the beginning. The recent reinterpretation of these principles maintains their focus on the importance of data being accesible to autonomous machines and further clarifies on the possible degrees of FAIRness. While there is no such notion as unFAIR, the authors discuss the different levels of FAIRness that can be achieved. As such, the addition of rich, FAIR metadata is the most important step towards becoming maximally FAIR. When data objects themselves can be made FAIR and open for reuse, the highest degree of FAIRness can be achieved. When all of these are linked with other FAIR data, the Internet of FAIR data is reached. Ultimately, when a large number of applications and services can link and process FAIR data, the Internet of FAIR Data and Services is attained. [4]

The integration and reuse of huge amounts of biomedical data currently available in digital format has the ability to impact clinical decisions, pharmaceutical discoveries, disease monitoring and the way population healthcare is provided globally. Storing data for future reuse and reference has been a critical factor in the success of modern biomedical sciences [1]. In order for data to be reused, first it has to be discovered. Finding a dataset for a study can be burdensome due to the need to search individual repositories, read numerous publications and ultimately contact data owners or publication authors on an individual basis. Recent research shows that the time spent by researchers in searching for and identifying multiple useful data sources can take up to 80% of their time dedicated to the project or research question itself [1].

3 Current standardization activities

There are already several standardization initiatives and standards describing electrophysiological data. In this chapter we will describe the most widespread. All of the following standards meet Fair principles, which were mentioned above.

3.1 NIX

The NIX project, originally know as Pandora, aims to develop standardized methods and models for storing electrophysiology and other neuroscience data together with their metadata in one common file format based on HDF5. [5]

NIX uses highly generic models for data as well as for metadata and defines standard schemata for HDF5 files which can represent those models. NIX also aims to provide a convenient C++ library to simplify the access to the defined format [5].

3.1.1 The model

The design principle of the data model used by NIX, was to create a rather minimalist, generic, yet expressive model, that is able to represent data stored in other widely used formats or models like NEO[1] without any loss of information. Due to its generic approach, the data model is furthermore able to represent also other kinds of data used in the field e.g. image data or image stacks [6].

The NIX model for data consists of six main elements: Block, DataArray, Tag, MultiTag, and Source and Group [6]. You can see NIX data model in Figure 3.2.

- **A Block entity** acts as a node that contains all other entities of the data model, it can be considered as something like a collection of data or a dataset.
- **A DataArray** is a multidimensional array of data that provides some additional information about the data it contains, such as data type

[1]Neo is a Python package for working with electrophysiology data in Python.

number of dimensions unit etc. Furthermore a DataArray entity has a of dimension descriptor for each dimension that provides information like sampling interval, unit or label. In brief: the DataArray already provides enough information to interpret its content and e.g. generate a plot.

- **The MultiTag, Tag** entities is mainly used to define regions of interests or certain points inside one or many DataArray entities. Therefore a Tags can be represent all kinds of different elements, for example an event, a spike train, an epoch or a segment.
- **Source** entities can define the provenance of a Tag or DataArray.
- **The Group**, finally acts as a simple grouping element which, in its current form, just expresses, that the members of the group (dataArrays, tags and multiTags) somehow belong together. A Group can link to Sources and can have metadata attached to it.

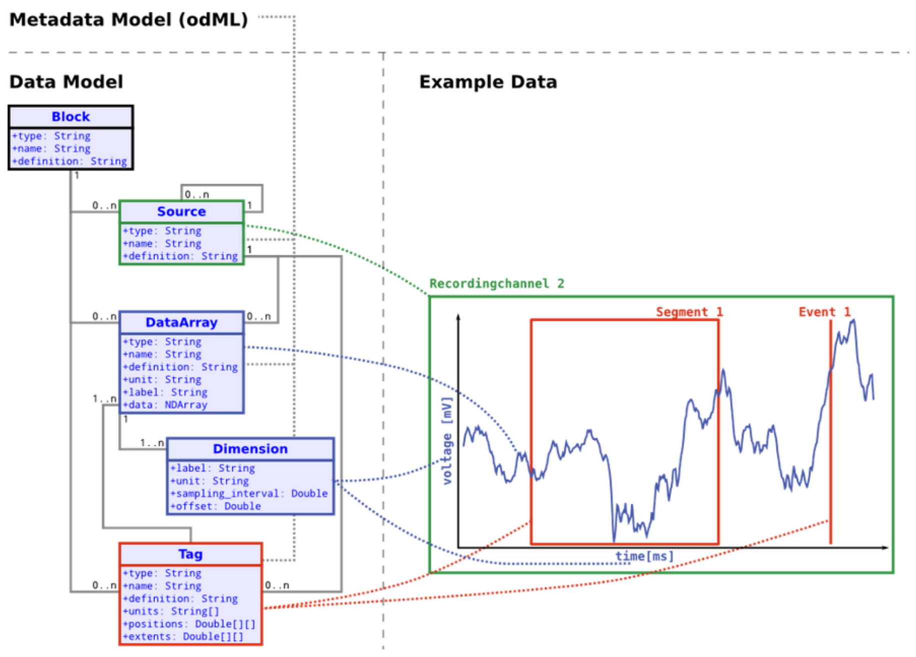


Figure 3.1: Simplified version of the data model that demonstrates how the basic entity types can describe a dataset containing an annotated analog signal. [6]

3.2 odML

odML (open metadata Markup Language) is a file format (XML, JSON, YAML) for storing metadata in an organised human- and machine-readable way. Well organized metadata management is a key component to guarantee reproducibility of experiments and to track provenance of performed analyses. [7]

This format specifies a hierarchical structure for storing arbitrary meta information as extended key-value pairs, so called properties, which can be logically grouped into sections and subsections. The odML defines the format, not the content, so that it is inherently extensible and can be adapted flexibly to the specific requirements of any laboratory. [8]

3.2.1 odML data model

The model is as simple as possible while being flexible, allowing interoperability, and being customizable. The model defines four entities (Property, Section, Value, RootSection) whose relations and elements are shown in Figure 3.2. [9]

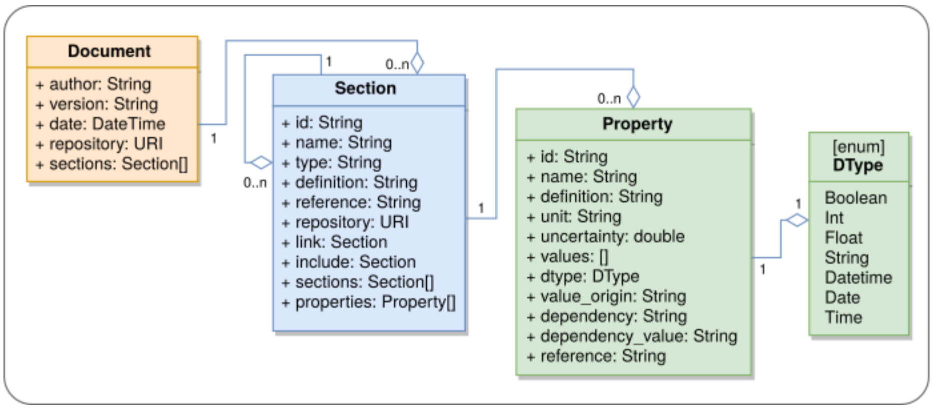


Figure 3.2: odML data model [9]

Property and Section are the core entities. A Section contains Properties and can further have subsection thus building a tree-like structure. The model does not constrain the content, which offers the flexibility essential for comprehensive annotation of neuroscience data. [9]

3.3 BIDS

The BIDS format is currently the standard for storing neuroimaging data. It tries to cover the most common experiments, but at the same time is intuitive and easy to adopt. The specification is intentionally based on simple file formats and folder structures to reflect current laboratory practices and make it accessible to a wide range of scientists coming from different backgrounds. [10]

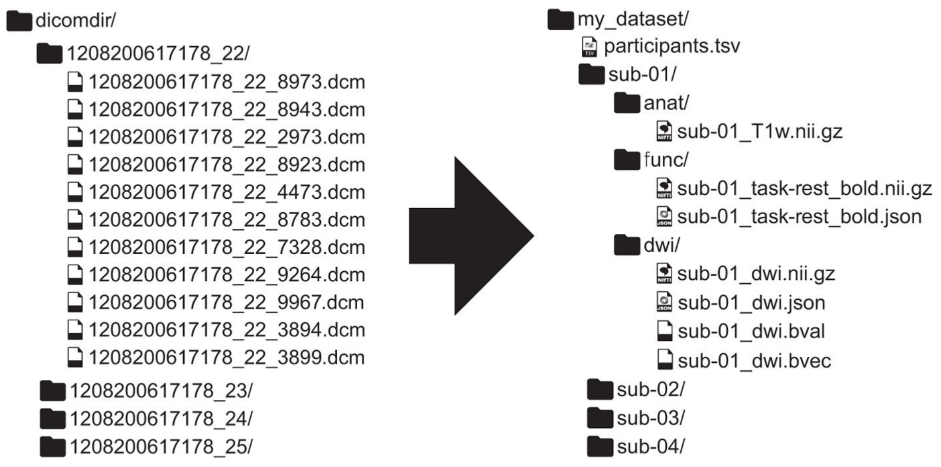

dicomdir/
1208200617178_22/
1208200617178_22_8973.dcm
1208200617178_22_8943.dcm
1208200617178_22_2973.dcm
1208200617178_22_8923.dcm
1208200617178_22_4473.dcm
1208200617178_22_8783.dcm
1208200617178_22_7328.dcm
1208200617178_22_9264.dcm
1208200617178_22_9967.dcm
1208200617178_22_3894.dcm
1208200617178_22_3899.dcm
1208200617178_23/
1208200617178_24/
1208200617178_25/
my_dataset/
participants.tsv
sub-01/
anat/
sub-01_T1w.nii.gz
func/
sub-01_task-rest_bold.nii.gz
sub-01_task-rest_bold.json
dwi/
sub-01_dwi.nii.gz
sub-01_dwi.json
sub-01_dwi.bval
sub-01_dwi.bvec
sub-02/
sub-03/
sub-04/

Figure 3.3: BIDS is a format for standardizing and describing outputs of neuroimaging experiments (left) in a way that is intuitive to understand and easy to use with existing analysis tools (right) [11].

3.3.1 BIDS-EEG

The extension of BIDS to EEG data closely follows the general BIDS specification: Each subject has a directory of raw data containing subdirectories for each session and modality. This is accompanied by a dataset_description.json file and a metadata file with the suffix _eeg.json, that specifies the task, the EEG system used (amplifier, hardware filter, cap, placement scheme, etc.). [12] This structure can be seen in Figure 3.3.

The process of converging on a list of suitable data formats for EEG-BIDS was governed by three major requirements: A suitable data format should (i) address the needs of a large portion of the global EEG community, (ii) be interoperable according to the FAIR principles, and (iii) meet the technical requirements of neuroscientific workflows, such as saving numerical data with high precision. [13]

As a solution to this challenge, the EEG-BIDS specification incorporates only two recommended “official” data formats: The European Data Format (EDF), which is an ongoing international effort to provide a common data format for electrophysiological recordings that began in 1992, and the BrainVision Core Data Format, developed by Brain Products GmbH.

While the BrainVision Core Data Format was designed by Brain Products GmbH for its proprietary EEG recording equipment and analysis software, it is based on the Microsoft Windows INI file and has a concise documentation. Both of these formats follow the three requirements for suitable data formats for EEG-BIDS: (i) A recent survey indicates that they are widely used in the community[2], (ii) they have open access documentation and an open source implementation for both reading and writing in at least two programming languages that are widely used in the field (in this case, Python and MATLAB, among others), and (iii) they have high numerical precision (EDF:16 bits, BrainVision Core Data Format:32 bits). To accommodate a larger scientific audience and facilitate adoption, the EEG-BIDS standard also allows two "unofficial" commonly used data formats: The format used by the MATLAB toolbox EEGLAB (".set" and ".fdt" files), and the Biosemi format (".bdf"). While not actively encouraged, these two formats are included due to their popularity and their interoperability among the major software packages. Future versions of BIDS may extend the list of "officially" supported data formats, based on the fulfillment of the above mentioned three requirements for suitable data formats. Independently of the raw data format used, critical metadata about the recording are always available in BIDS .tsv and .json files. [13]

Community Tools and Software Support

As part of the BIDS project, datasets formatted to follow the EEG-BIDS standard can be validated using the "bids-validator", a JavaScript application that runs locally as a command line version (using Node.js) or within an Internet browser[3]. With this validation tool, researchers can check their newly formatted datasets and make full use of the data structure's strengths for instance, checking for missing data or underspecified metadata. [13]

The BIDS starter kit [4] is a collection of community-driven guides, tutorials, helper scripts, and wiki resources to help researchers get started with BIDS. The resources cover two popular programming languages (Python and MATLAB) and will be extended over time to incorporate additional guides. [13]

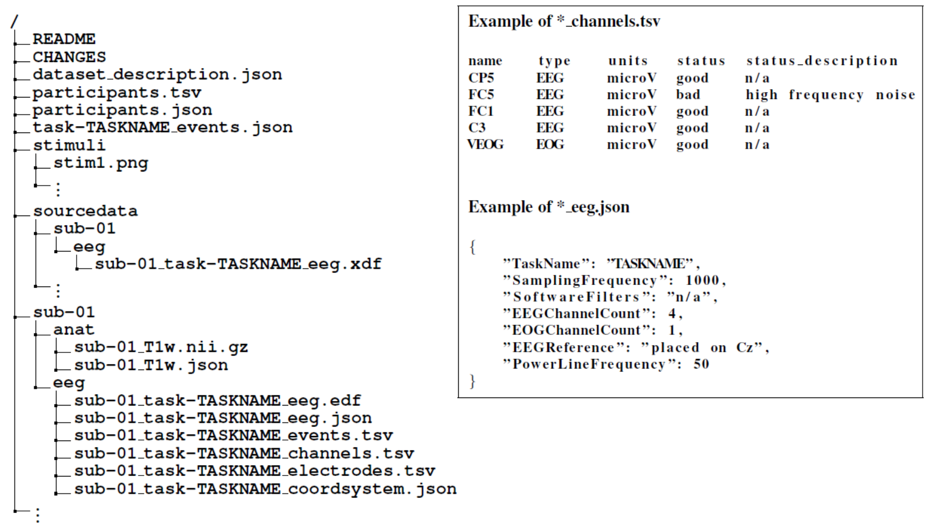
[2]`https://bids.berkeley.edu/news/bids-megeegieeg-data-format-survey`.

[3]`https://bids-standard.github.io/bids-validator/`.

[4]`https://github.com/bids-standard/bids-starter-kit`.

Directory tree of a BIDS dataset

A prototypical directory tree of a BIDS dataset containing EEG data. This can be seen in Figure 3.4. At the root level of the directory, the README, CHANGES, and dataset_description.json files provide basic information about the dataset. A participants.tsv data file is accompanied by a participants.json file, which contains the description of the columns in its associated .tsv file. The panel in the upper right of the figure provides examples on the typical format within a .tsv and .json file. Usually, each .tsv file is accompanied by a .json file that provides metadata. The EEG data and anatomical MRI scans are saved per subject within the eeg and anat subdirectories respectively. If the original data is not supported by BIDS, it can be included in an additional sourcedata directory. Finally, a stimuli directory contains the stimuli that were presented to the participants in the experiment. [12]



```
/
├─README
├─CHANGES
├─dataset_description.json
├─participants.tsv
├─participants.json
├─task-TASKNAME_events.json
├─stimuli
│ ├─stim1.png
│ └─⋮
├─sourcedata
│ ├─sub-01
│ │ └─eeg
│ │   └─sub-01_task-TASKNAME_eeg.xdf
│ └─⋮
├─sub-01
│ ├─anat
│ │ ├─sub-01_T1w.nii.gz
│ │ └─sub-01_T1w.json
│ └─eeg
│   ├─sub-01_task-TASKNAME_eeg.edf
│   ├─sub-01_task-TASKNAME_eeg.json
│   ├─sub-01_task-TASKNAME_events.tsv
│   ├─sub-01_task-TASKNAME_channels.tsv
│   ├─sub-01_task-TASKNAME_electrodes.tsv
│   └─sub-01_task-TASKNAME_coordsystem.json
└─⋮
```

Example of *_channels.tsv

name	type	units	status	status_description
CP5	EEG	microV	good	n/a
FC5	EEG	microV	bad	high frequency noise
FC1	EEG	microV	good	n/a
C3	EEG	microV	good	n/a
VEOG	EOG	microV	good	n/a

Example of *_eeg.json

```
{
    "TaskName": "TASKNAME",
    "SamplingFrequency": 1000,
    "SoftwareFilters": "n/a",
    "EEGChannelCount": 4,
    "EOGChannelCount": 1,
    "EEGReference": "placed on Cz",
    "PowerLineFrequency": 50
}
```

Figure 3.4: A prototypical directory tree of a BIDS dataset containing EEG data. [12].

3.4 NWB

Neurodata Without Borders: Neurophysiology (NWB:N) is a data standard for neurophysiology, providing neuroscientists with a common standard to share, archive, use, and build common analysis tools for neurophysiology data. NWB:N is designed to store a variety of neurophysiology data, including data from intracellular and extracellular electrophysiology experiments, data from optical physiology experiments, and tracking and stimulus data. [14]

The project includes not only the NWB format, but also a broad range of software for data standardization and application programming interfaces (APIs) for reading and writing the data as well as high-value data sets that have been translated into the NWB data standard. [14]

You can clearly see what NWB is in Figure 3.5.

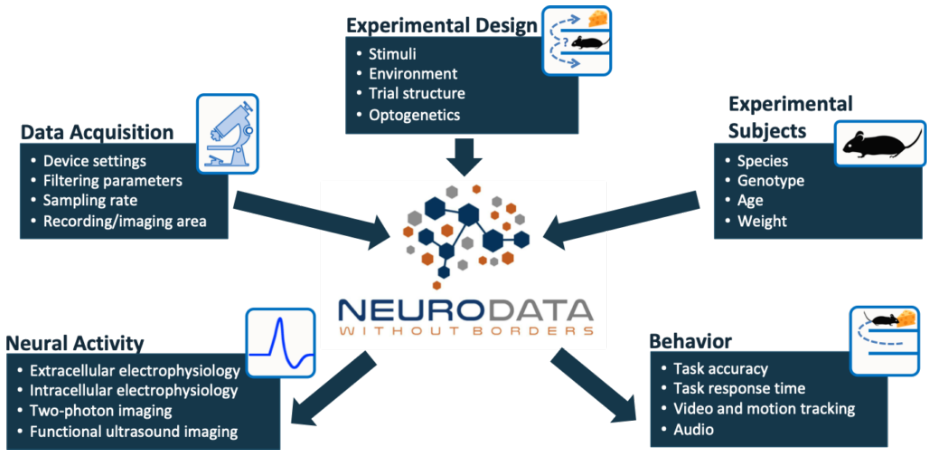


Figure 3.5: NWB model, what NWB is working with and what it can be used for. [14]

NWB is more than just a file format; it defines an ecosystem of tools, methods, and standards for storing, sharing, and analyzing complex neurophysiology data. [14]

3.5 Comparison of NIX, NWB and BIDS

In this section we will compare NIX, NWB and BIDS formats. These formats will be compared on the basis of structure, source code and documentation, as well as statistics obtained from Github. Based on this comparison, I wanted to decide which format to choose. It was not easy to decide on a specific format, because as you can see in the following description, the formats are similar in terms of documentation and source code, with a few differences. So I had to add other factors based on which to select the format. There were two factors, namely the similarity of the BrainVision format, so this factor was for BIDS and also for the possibility of consulting with the developers of the format, which was NIX, as the university has some cooperation with them. That's why I decided between these formats and I eliminated NWB. I also used the table in the decision, which you can see in

	NIX/odML	NWB	BIDS
Contributors	14	23	37
Commits	1709	3094	1743
Watch	8	22	21
Star	13	67	91
Fork	22	41	67

Table 3.1: Compare NIX and BIDS by github.

Table 3.2. It compares these two formats, BIDS and NIX in Fair Principles, which are described in the first chapter.

NIX and NWB use the 3-Clause BSD License. Both are open source and have their source code and documentation stored on github.

A comparison of the statistics that can be read from github on December 16, 2019 is shown in the following table. It can be seen that NIX / odML lags the most in these statistics. NWB and BIDS are approximately at a similar level.

3.5.1 NIX

Project documentation of NIX is split up into three parts: technical information, general introduction and tutorial, and API documentation. The Nixpy, which is an extension to NIX and provides Python bindings for NIX, has a file in addition to the documentation. The file outlines the features that have been implemented in NIX and have not yet been added to NIXPy. The features are separated into two sections: Released features and unreleased ones. The latter is for features that exist in nix/master but have yet to be included in a stable release.

In another file there is a guide, which shows build options for both, 32- and 64-bit Windows. In the manual there is a link to using the installer provided (exe file). The instructions are brief but clear.

NIX has almost no commented code. Compared to NIX, nixpy has a detailed commented code. In the case of a longer method, nixpy has detailed comments that describe the entire functionality of the method. The method has no comment if the method is short and the name clearly explains what the method does.

If we want to use Elephant, then it would be appropriate to use NEO, then NIX needs to be used to connect MNE with NEO. You can see this on the model in Figure 3.6.

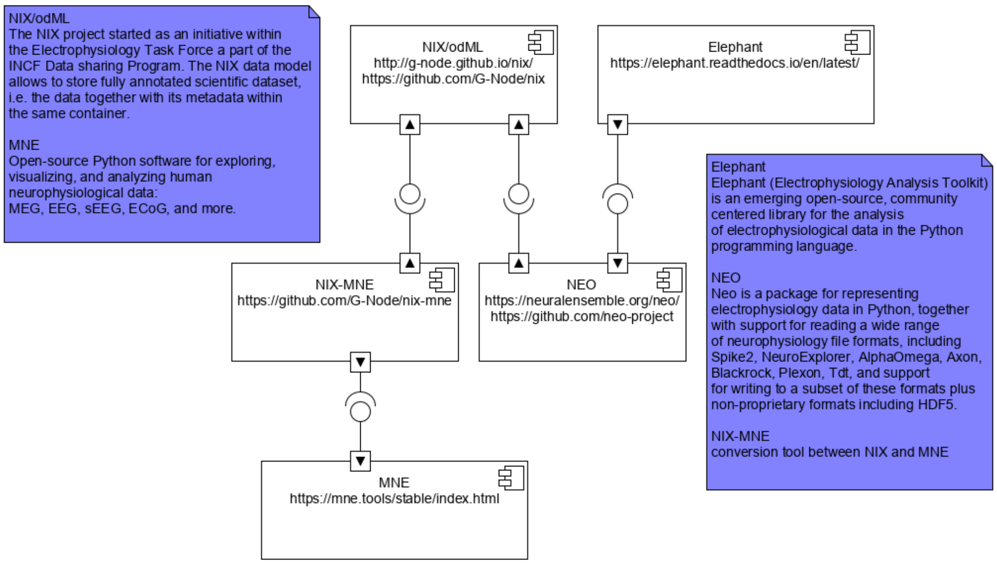

NIX/odML
The NIX project started as an initiative within the Electrophysiology Task Force a part of the INCF Data sharing Program. The NIX data model allows to store fully annotated scientific dataset, i.e. the data together with its metadata within the same container.
MNE
Open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more.
NIX/odML
http://g-node.github.io/nix/
https://github.com/G-Node/nix
Elephant
https://elephant.readthedocs.io/en/latest/
Elephant
Elephant (Electrophysiology Analysis Toolkit) is an emerging open-source, community centered library for the analysis of electrophysiological data in the Python programming language.
NEO
Neo is a package for representing electrophysiology data in Python, together with support for reading a wide range of neurophysiology file formats, including Spike2, NeuroExplorer, AlphaOmega, Axon, Blackrock, Plexon, Tdt, and support for writing to a subset of these formats plus non-proprietary formats including HDF5.
NIX-MNE
conversion tool between NIX and MNE
NIX-MNE
https://github.com/G-Node/nix-mne
NEO
https://neuralensemble.org/neo/
https://github.com/neo-project
MNE
https://mne.tools/stable/index.html

Figure 3.6: Component diagram of connection between MNE and NEO. It is necessary to use NIX.

3.5.2 BIDS

BIDS has a library to centralize interactions with datasets conforming BIDS format that is called pybids. If you want install the most recent release, use pip (pip install pybids). Pybids has commented methods in great detail. Methods have detailed comments on individual parameters. What the methods do can be understood from the title, or the first sentence of the comment explains it. Pybids has written tests for each part.

3.5.3 NWB

Pynwb doesn't have as much commented code as nixpy or pybids. Some files are not commented at all. Pynwb has a link to the installation guide in readme. Pynwb contains a link to the installation instructions in the readme file. You can easily install pynwb according to this tutorial. In addition to the installation, the manual also includes instructions for running the tests. As for code testing, unlike nixpy, which has all tests in one folder and pybids, which has tests in each part of the implementation, pynwb has tests divided into unit and integration testing.

Criterion under consideration	System	
	NIX, odML	BIDS, LORIS
Findable		
Data are assigned a globally unique and eternally persistent identifier.	YES	YES
Metadata are assigned a globally unique and eternally persistent identifier.	YES	NO
Data are described with rich metadata.	YES	YES
(Meta)data are registered or indexed in a searchable resource.	YES	YES
Metadata specify the data identifier.	NO	NO
Accessible		
(Meta)data are retrievable by their identifier using a standardized communications protocol.	NO	YES
The protocol is open, free, and universally implementable.	YES	YES
The protocol allows for an authentication and authorization procedure, where necessary.	YES	YES
Metadata are accessible, even when the data are no longer available.	YES	YES
Interoperable		
(Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.	YES	YES
(Meta)data use vocabularies that follow FAIR principles.	YES	YES
(Meta)data include qualified references to other (meta)data.	YES	NO
Re-usable		
Meta(data) have a plurality of accurate and relevant attributes.	YES	YES
(Meta)data are released with a clear and accessible data usage license.	YES	YES
(Meta)data are associated with their provenance.	YES	YES
Data meet domain-relevant community standards.	YES	YES
Metadata meet domain-relevant community standards.	NO	YES

Table 3.2: Comparison of systems in terms of FAIR principles. [3]

4 Tools

We described current standardization activities in the previous chapter. And we will describe the tools that can work with these standardized structures in this chapter.

4.1 LORIS

The first tool we will describe is LORIS. LORIS is a web-based data management system for longitudinal, multisite neuroimaging research data collection, with a key feature being joint management of heterogeneous data modalities (behavioural/clinical, genetic, biosamples, and imaging). [15]

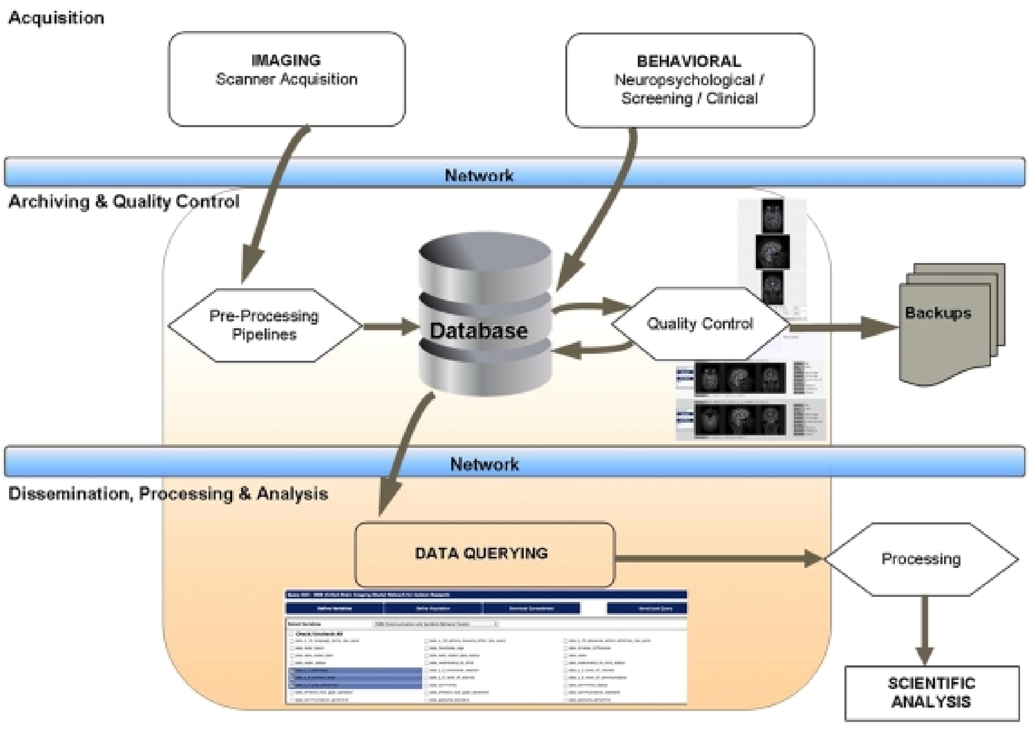


Figure 4.1: Data flow within the LORIS system. [15]

The emerging Brain Imaging Data Structure (BIDS) standard extension for EEG offers a format for organizing and describing outputs of EEG experiments, which can easily be imported into LORIS, while enabling linking to other modalities collected for the same subjects. Figure 4.1 depicts the LORIS workflow for importing electrophysiology datasets organized in a BIDS structure. [15]

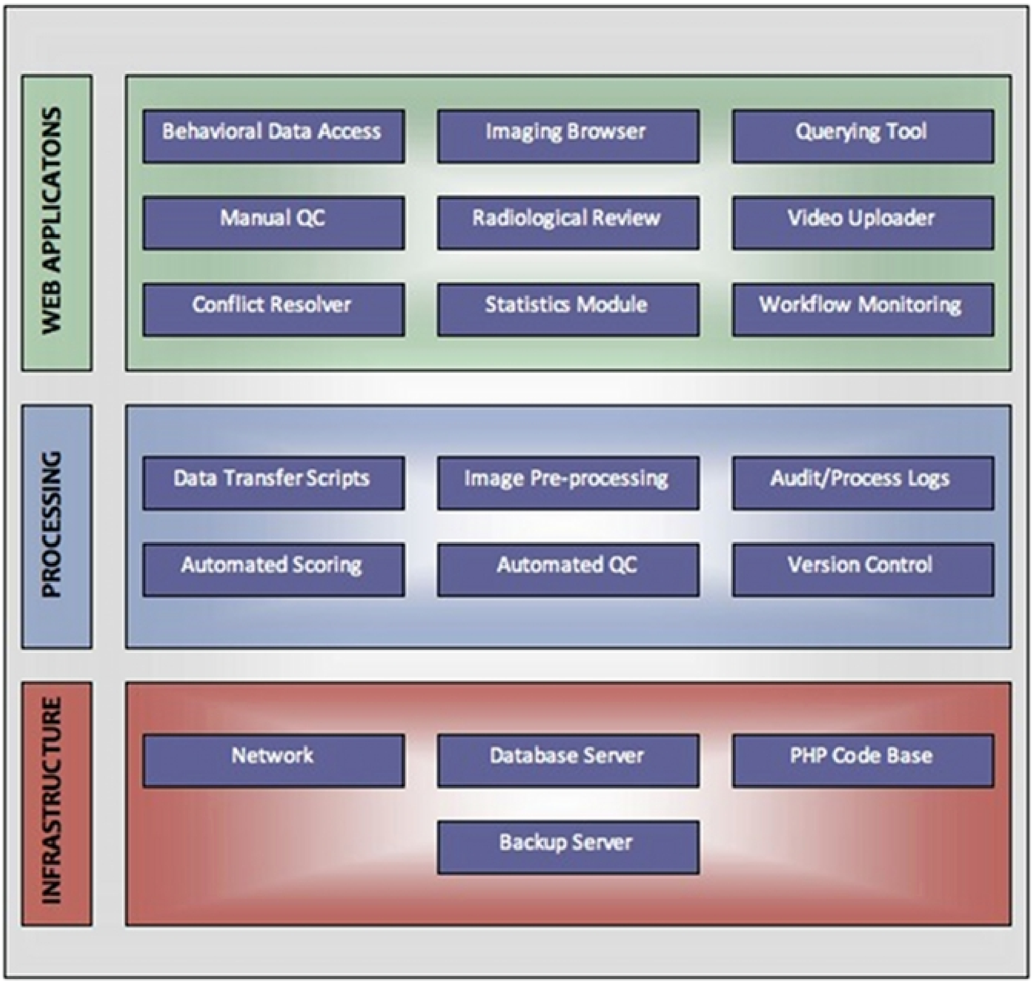


Figure 4.2: LORIS system architecture layers. [15]

From a system architecture point of view, LORIS can be viewed as having three component layers: an infrastructure layer, processing layer, and web application layer. You can see this three component layers in Figure 4.2. Data entry, transfer, image pre-processing, visualization, and quality control are all aspects that take place within these layers, with the ultimate goal of serving data via the Data Querying GUI to facilitate processing and analysis. [15]

The LORIS processing layer is indexed to provide optimized views such that queries are executed as quickly as possible. While on the front end, menus have been created to allow for filtering, both of the data itself as well as the top-level view of status information, which allows project manager to filter for status information of any dataset without the need to enter and view individual profile data. [15]

The basic construct is developed around a structured data model called the “Subject Profile,” which is used to map the study-specific battery of instruments and methodologies, including an array of external metadata collected about the study subjects. Data is organized in a subject-centric manner (i.e., around a study subject on which data is collected) and offers all necessary identifying subject information to allow project managers to filter for status information without a need to enter and view individual profiles. Demographic information is collected and a unique set of anonymized subject identifiers are assigned. [15]

The creation of the Subject Profile contains “timepoints”—longitudinal extensions representing study iterations where a subject returns for multiple visits. Timepoints are used to collect the full array of instruments and other data collected during a subject’s visit. In case of multi-site studies, the timepoints are associated with a specific study site enabling the researchers to track individual subjects over time at different geographical locations. [15]

Researchers can access LORIS modules in a seamless and intuitive fashion using the following three sections of its front-end layer: (1) the Behavioral Database, (2) Imaging Browser, and (3) Data Querying GUI (DQG). [15]

4.2 Elephant

Elephant (Electrophysiology Analysis Toolkit) is an open-source, community centered library for the analysis of electrophysiological data in the Python programming language. The focus of Elephant is on generic analysis functions for spike train data and time series recordings from electrodes, such as the local field potentials (LFP) or intracellular voltages. In addition to providing a common platform for analysis codes from different laboratories, the Elephant project aims to provide a consistent and homogeneous analysis framework that is built on a modular foundation. [16]

4.2.1 Elephant library structure

Elephant is a standard python package and is structured into a number of submodules. A sketch of the layout of the Elephant library (0.3.0 release) is shown in Figure 4.3. [16]

Conceptually, modules of the Elephant library can be divided into those related to a specific category of analysis methods, and supporting modules that provide a layer of various core utility functions. All available modules are available directly on the the top level of the Elephant package in the elephant subdirectory to avoid unnecessary hierarchical clutter. [16]

4.3 MNE

MNE is an open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more. [17]

Tutorials are available on the official website. These tutorials provide narrative explanations, sample code, and expected output for the most common MNE-Python analysis tasks. The emphasis here is on thorough explanations that get you up to speed quickly, at the expense of covering only a limited number of topics. The sections and tutorials are arranged in a fixed order, so in theory a new user should be able to progress through in order without encountering any cases where background knowledge is assumed and unexplained. More experienced users (i.e., those with significant experience analyzing EEG/MEG signals with different software) can probably skip around to just the topics they need without too much trouble. [17]

In addition to tutorials, examples are also available. The examples gallery provides working code samples demonstrating various analysis and visualization techniques. These examples often lack the narrative explanations seen in the tutorials, and do not follow any specific order. These examples are a useful way to discover new analysis or plotting ideas, or to see how a particular technique you've read about can be applied using MNE-Python. [17]

Cou can see the workflow of the MNE software in Figure D.1 and Extensive data model is in Figure D.2.

4.4 NEO

Neo is a Python package for working with electrophysiology data in Python, together with support for reading a wide range of neurophysiology file formats, including Spike2, NeuroExplorer, AlphaOmega, Axon, Blackrock, Plexon, Tdt, and support for writing to a subset of these formats plus non-proprietary formats including HDF5. [18]

The goal of Neo is to improve interoperability between Python tools for analyzing, visualizing and generating electrophysiology data by providing a common, shared object model. In order to be as lightweight a dependency as possible, Neo is deliberately limited to represention of data, with no functions for data analysis or visualization. [18]

Neo is used by a number of other software tools, including SpykeViewer (data analysis and visualization), Elephant (data analysis), the G-node suite (databasing), PyNN (simulations), tridesclous (spike sorting) and ephyviewer (data visualization). OpenElectrophy (data analysis and visualization) uses an older version of neo. [18]

Neo implements a hierarchical data model well adapted to intracellular and extracellular electrophysiology and EEG data with support for multi-electrodes (for example tetrodes). Neo's data objects build on the quantities package, which in turn builds on NumPy by adding support for physical dimensions. Thus Neo objects behave just like normal NumPy arrays, but with additional metadata, checks for dimensional consistency and automatic unit conversion. [18]

You can see the class diagram of NEO in Figure D.3.

4.5 BIDS App

A BIDS App is a container image capturing a neuroimaging pipeline that takes a BIDS formatted dataset as input. The BIDS App is mainly for the neuroimaging pipeline, but it could also be used in our case. Each BIDS App has the same core set of command line arguments, making them easy to run and integrate into automated platforms. BIDS Apps are constructed in a way that does not depend on any software outside of the image other than the container engine. [19]

BIDS Apps rely upon two technologies for container computing:

- **Docker** - for building, hosting as well as running containers on local hardware (running Windows, Mac OS X or Linux) or in the cloud.
- **Singularity** - for running containers on HPCs.

BIDS Apps are deposited in the Docker Hub repository, making them openly accessible. [19]

The source code of each App is stored in separate GitHub repository. Each repository is connected to a Continuous Integration server responsible for building testing and deploying the corresponding App. For every new release of an App, a new container image is deposited in Docker Hub. Users can directly download and run the BIDS Apps container images either directly using Docker on any Windows, Mac, and Linux machine or convert them to Singularity and run them on an HPC. [20]

4.5.1 Command-line interface

To improve user experience and ability to integrate BIDS Apps into various computational platforms, each App follows a set of core command-line

arguments:[20]
runscript input_dataset output_folder analysis_level
For example:
runscript /data/ds114 /scratch/outputs participant

- input_dataset provides a path to the dataset to be analyzed (read-only), which must conform to the BIDS standard
- output_folder is the folder where results of the analysis will be stored
- analysis_level denotes the stage of the analysis that will be performed

They also provide several utilities that make it easier to work with BIDS-compatible directory structures-most notably, the PyBIDS Python package[1], which provides tools for simple but powerful logical queries over entities defined in the BIDS specification (e.g., retrieving a list of all unique subjects; getting the fieldmap files for all subjects with a valid first scanning run; etc.). [21]

In addition to conforming to a standardized command-line argument scheme, run scripts are also responsible for validation of the input data before running any analysis. To facilitate the process they have developed a command-line validator that checks whether the input datasets are compliant with the BIDS standard. [21]

This approach to run their workflows requires sticking with three standards: 1) a common command-line interface, 2) a Docker container to ensure portability, and 3) a standard for organizing input data. Containers created this way can be easily integrated into OpenfMRI as well as other data analysis platforms. Thanks to Docker-to-Singularity conversion they can also be easily run on High Performance Computers (clusters) without the need to install all of the dependencies. [21]

4.5.2 Command-line specification

Each workflow/pipeline will be run independently for each subject (the map step). Results of this execution (arranged in whatever way the pipeline prefers) can be optionally processed in a group level analysis (reduce step–see Figure 4.4). [20]

[1]`https://github.com/INCF/pybids`.

4.6 The BIDS Validator

The BIDS Validator is designed to work in both the browser and in Node.js. They target support for the latest long term stable (LTS) release of Node.js and the latest version of Chrome [22].

4.6.1 API

The BIDS Validator has one primary method that takes a directory as either a path to the directory (node) or the object given by selecting a directory with a file input (browser), an options object, and a callback [22].

4.7 SPM

The SPM software package has been designed for the analysis of brain imaging data sequences. The sequences can be a series of images from different cohorts, or time-series from the same subject. The current release is designed for the analysis of fMRI, PET, SPECT, EEG and MEG. [23]

4.8 Pybids

Pybids is a Python library to centralize interactions with datasets conforming BIDS (Brain Imaging Data Structure) format. [24] Pybids is a set of tools for working with Brain Imaging Data Structure (BIDS) datasets. Pybids makes it easier for neuroimagers who utilize the BIDS standard to query, summarize, and manipulate their data. A number of Python packages for analyzing neuroimaging data, including Nipype and nistats, are optimized to work with BIDS datasets [25].

Pybids is currently designed to work with image libraries. In the future it is planned to expand the BIDS data standard to include MEG and EEG[25]. In this thesis we work with EEG-BIDS, which unfortunately Pybids is not yet working with, but MNE works with EEG-BIDS.

In addition to MNE, EEG-BIDS also cooperates with SPM, which is written in Matlab[23].

4.9 MNE-python

Magnetoencephalography and electroencephalography (MEG/EEG) measure the weak electromagnetic signals generated by neuronal activity in the brain. Using these signals to characterize and locate neural activation in the brain is a challenge that requires expertise in physics, signal processing, statistics, and numerical methods. As part of the MNE software suite, MNE-Python is an open-source software package that addresses this challenge by providing state-of-the-art algorithms implemented in Python that cover multiple methods of data preprocessing, source localization, statistical analysis, and estimation of functional connectivity between distributed brain regions. All algorithms and utility functions are implemented in a consistent manner with well-documented interfaces, enabling users to create M/EEG data analysis pipelines by writing Python scripts. Moreover, MNE-Python is tightly integrated with the core Python libraries for scientific comptutation (NumPy, SciPy) and visualization (matplotlib and Mayavi[2]), as well as the greater neuroimaging ecosystem in Python via the Nibabel package. The code is provided under the new BSD license allowing code reuse, even in commercial products. Although MNE-Python has only been under heavy development for a couple of years, it has rapidly evolved with expanded analysis capabilities and pedagogical tutorials because multiple labs have collaborated during code development to help share best practices. MNE-Python also gives easy access to preprocessed datasets, helping users to get started quickly and facilitating reproducibility of methods by other researchers. [26]

4.9.1 Design, Application Programming Interface (API) and Data Structures

M/EEG data analysis typically involves three types of data containers coded in MNE-Python as Raw, Epochs, and Evoked objects. The raw data comes straight out of the acquisition system; these can be segmented into pieces often called epochs or trials, which generally correspond to segments of data after each repetition of a stimulus; these segments can be averaged to form evoked data. MNE-Python is designed to reproduce this standard operating procedure by offering convenient objects that facilitate data transformation. [26]

4.9.2 Preprocessing

The major goal when preprocessing data is to attenuate noise and artifacts from exogenous (environmental) and endogenous (biological) sources. Noise reduction strategies generally fall into two broad categories: exclusion of

[2]`http://mayavi.sourceforge.net/`.

contaminated data segments and attenuation of artifacts by use of signal-processing techniques. MNE-Python offers both options at different stages of the pipeline, through functions for automatic or semi-automatic data preprocessing as well as interactive plotting capabilities. [26]

4.10 MNE-BIDS

MNE-BIDS links BIDS and MNE with the goal to make your analyses faster to code, more robust to errors, and easily sharable with colleagues[27]. MNE-BIDS is a part of MNE-Python.

4.11 Summary of tools

I described the tools that can work with these standardized structures. All of the tools described above can be used for NIX or BIDS. Since I have already eliminated NWB, I have not focused on tools that would be suitable for NWB.

Elephant and NEO are tools that can be used mainly for NIX, you can see it in Figure 3.6.

MNE is important for BIDS. It is also advisable to use the BIDS Validator that checks if the BIDS data are in the correct format.

Data in one of the mentioned standards will need to be stored in a repository; this will be the topic of the next chapter.

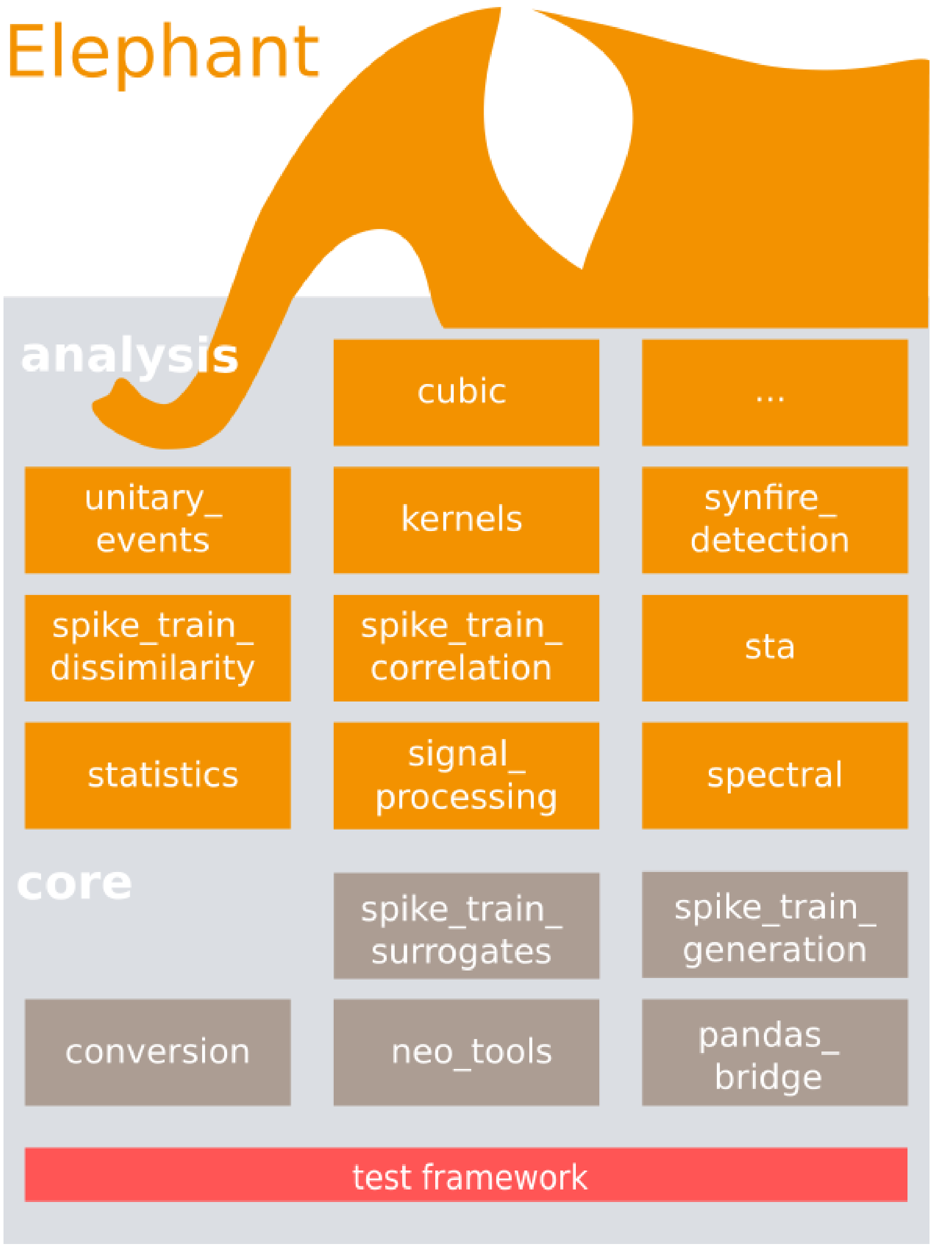


Figure 4.3: Modules of the Elephant library. Modules containing analysis functions are colored in orange shades, core functionality in grey shades [16].

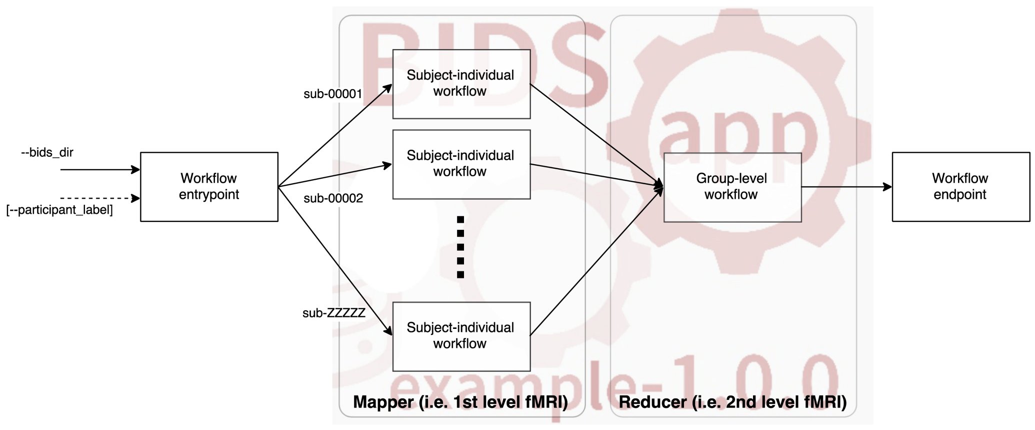

--bids_dir
[--participant_label]
Workflow entrypoint
sub-00001
sub-00002
sub-ZZZZZ
Subject-individual workflow
Subject-individual workflow
Subject-individual workflow
Group-level workflow
Workflow endpoint
Mapper (i.e. 1st level fMRI)
Reducer (i.e. 2nd level fMRI)
BIDS app
example-1.0.0

Figure 4.4: The overall structure of workflows. [20]

5 Repositories

Data that is stored in the mentioned formats needs to be stored in a repository that will be publicly available so that everyone can access the data. Ideally, we need a repository that will have a DOI (Digital Object Identifier), a unique and permanent identifier of a digital object accessible through digital networks, which will make our data more traceable. So I will describe some of such repositories.

5.1 GIN

The GIN platform is a web-based repository store that you can reach from everywhere, which is secure and which provides fine-grained access control. This enables sharing data with collaborators or publishing your data to the greater scientific community. Both you and your collaborators can work independently, using versioned repositories at work, at home, or wherever you want. All this can either be achieved through various convenient client interfaces or by using the underlying tools (git, git-annex) directly. You can also set up your own GIN server [28].

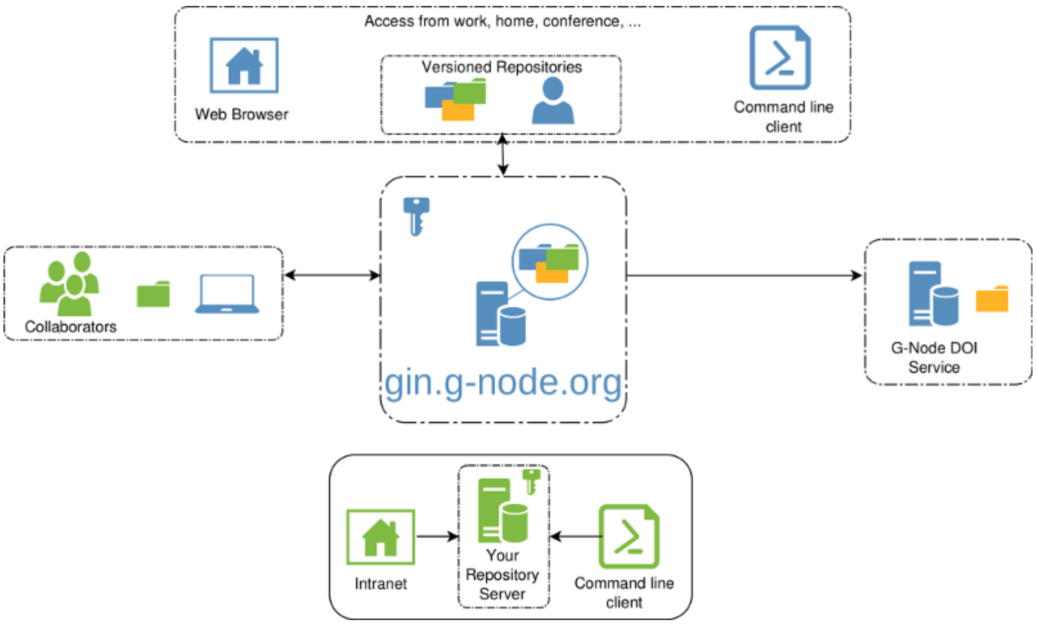


Figure 5.1: Schematic overview of GIN illustrating its many usage possibilities. [28]

Any GIN repository hosted on the official GIN service can be permanently archived (at a given point in time) and referenced by a unique Digital Object Identifier (DOI) [29]. You can see everything described above in Figure5.1.

The GIN service is based on the Gogs Git service.

The data versioning, storage, and synchronisation part of the project is built on git and git-annex[29].

5.1.1 Set up GIN

To work with GIN, you can use the web interface directly, where you register and then you can create repositories and work with it.

However, if you want to configure GIN in your own lab, you have the following two options:

- without Docker
- with Docker

Without Docker

There is a detailed guide to setting up a GIN on the website[1]. This Tutorial is written with respect to Ubuntu 16.04. It should work for all tuxish distributions, however, the packages might have different names and their installation could be different.

With Docker

The easiest way to set up a GIN server is via prebuild docker containers. You will need a working Docker installation. There is a detailed guide to setting up a GIN on the website[2].

5.1.2 GIN search

The GIN data search provides you with a set of tools to search the gin-index (gindex). Gindex analyses repositories for content types it knows (Text, XML, PDF, JSON, odML, NEV to name a few) and that are not too big. From there, it builds a representation that is good for (full-text) searching. [30]

There are several ways to search the gindex [30].

[1]`https://gin.g-node.org/G-Node/Info/wiki/In+House+Without+Docker`.

[2]`https://gin.g-node.org/G-Node/Info/wiki/In+House`.

- Match search: which tries to find the terms you provide (exactly) and returns them sorted by a score that depends on the number of matches and the size of the document the match is in. This is probably the best for most people. [30]
- Fuzzy term matching: which is a way to find matches to individual terms and things that look close to the term provided (eg. bla also matches blu). [30]
- Wildcard term matching: Matches individual terms with the possibility to provide wildcards. ? can be used to replace a single character, and * to replace zero or more characters (eg.bl* matches on blu, blo, and bl etc.). [30]
- Query String: This is the most powerful search and kind of a combination of the three above. You can provide a full query string using among other things, the wildcard introduced above, and also the fuzzy operator   which uses a Damerau-Levenshtein distance of 2 to find all terms that match (eg. "Sp?ke Sortung " does match Spike Sorting and a lot of other things). In query strings, you can also use the boolean operators AND OR and NOT. For example, "Spike NOT Sorting" would match documents that have Spike but not Spike Sorting". If you want you can even group those "(Spike NOT Sorting) AND train". [30]

It's kind of obvious that wildcard and fuzzy searching can be computationally pretty costly. Therefore give gindex some time to retrieve the results. [30]

5.1.3 GIN-proc

gin–proc is a GIN micro-service which allows the users to design efficient workflows for their work - by automating Snakemake, and build the workflows with open-source version of Continuous Integration (CI) service Drone. [31]

This tool/micro-service is required since, given the GIN user base of neuroscientists and other pro-fessionals from the related fields, shouldn't be involved in writing thousands of repeated workflows for their data, and then testing it manually. This tool will increase their efficiency by almost exponential levels by eradicating redundancy from their work. [31]

5.2 DANDI

DANDI is a platform for publishing, sharing, and processing neurophysiology data funded by the BRAIN Initiative. The platform is now available for data upload and distribution. [32]

DANDI is powered by Girder, a part of Resonant, Kitware's open-source platform for data management, analytics, and visualization. [32]

5.2.1 Resonant

Resonant is a platform consisting of tools that work in concert to provide storage, analysis, and visualization solutions for your data. All Resonant components are fully open source under the Apache v2 license. [33]

Data Management

Upload, share, and manage your data using Amazon S3, Distributed file systems, SQL, NoSQL, and more. [33]

Analytics

Perform heavy-lifting on your data with Python and Docker containers through a uniform interface. [33]

Visualization

Gain insight into data with flexible, scalable web visualizations. [33]

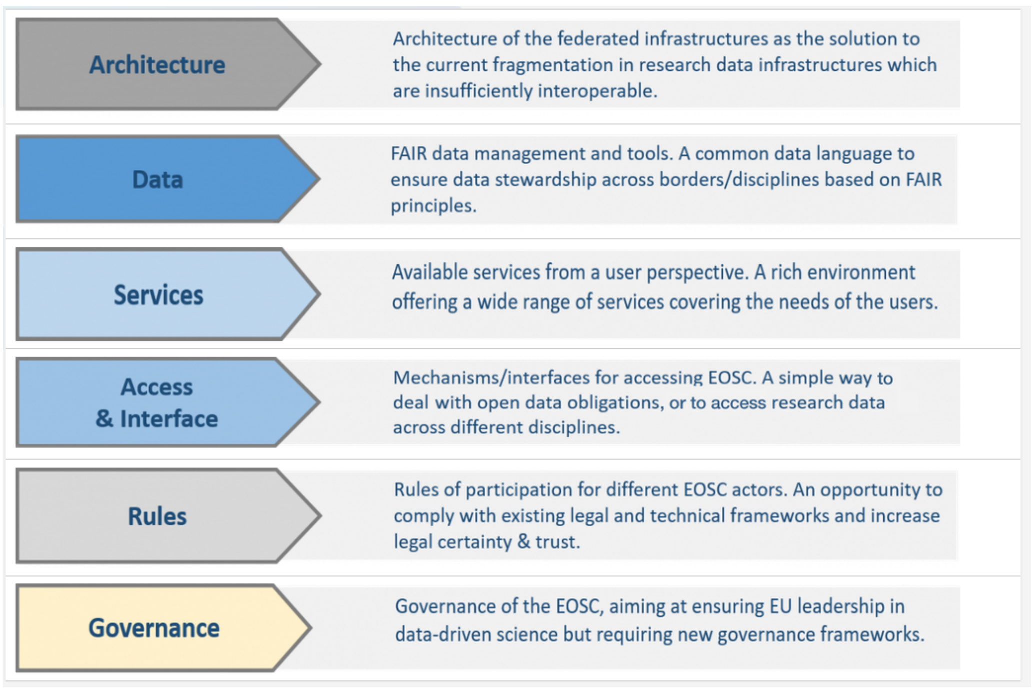
5.2.2 Girder

Girder is a Data Management Toolkit. It is a complete back-end (server side) technology that can be used with other applications via its RESTful API, or it can be used via its own front-end (client side web pages and JavaScript). [33]

Girder is designed to be robust, fast, scalable, extensible, and easy to understand. Girder is built in Python. [33]

5.3 EOSC

The European Open Science Cloud (EOSC) initiative has been proposed in 2016 by the European Commission as part of the European Cloud Initiative to build a competitive data and knowledge economy in Europe. An extensive consultation with scientific and institutional stakeholders took place in 2016 and 2017. An engagement process was initiated with a first EOSC Summit in June 2017, resulting in the EOSC Declaration endorsed by more than 70 institutions. The summary outcome of the consultation was presented in March 2018 by the European Commission in the form of a roadmap for implementing the EOSC. [34] What EOSC offers is shown in Figure 5.2.



Architecture
Architecture of the federated infrastructures as the solution to the current fragmentation in research data infrastructures which are insufficiently interoperable.

Data
FAIR data management and tools. A common data language to ensure data stewardship across borders/disciplines based on FAIR principles.

Services
Available services from a user perspective. A rich environment offering a wide range of services covering the needs of the users.

Access & Interface
Mechanisms/interfaces for accessing EOSC. A simple way to deal with open data obligations, or to access research data across different disciplines.

Rules
Rules of participation for different EOSC actors. An opportunity to comply with existing legal and technical frameworks and increase legal certainty & trust.

Governance
Governance of the EOSC, aiming at ensuring EU leadership in data-driven science but requiring new governance frameworks.

Figure 5.2: EOSC [34]

The European Commission is providing financial support to implement the EOSC by means of projects under the EU Framework Programme for Research and Innovation (Horizon 2020). The EOSC will consist of several projects, most of which are not yet complete. [34]

5.4 EBRAINS

EBRAINS integrated workflows at multiple scales allow users to perform complex computational experiments, including estimation of model parameters, model validation and large-scale simulations including analysis and visualisation. Workflows addressing the needs of scientists across disciplines and levels of expertise will be accessible through pre-configured web applications or flexible digital notebooks. [35]

5.4.1 The new data era

EBRAINS Data & Knowledge services promote a culture of data cooperation and discovery. They offer one of the most comprehensive platforms for sharing brain research data ranging in type as well as spatial and temporal scale [35].

5.4.2 Share data

EBRAINS provide the guidance and tools needed to overcome the hurdles associated with sharing data. The EBRAINS data curation service ensures that your dataset will be shared with maximum impact, visibility, reusability, and longevity. [35]

5.4.3 Find data

The user interface of the EBRAINS Knowledge Graph - allows you to easily find data of interest. EBRAINS hosts a wide range of data types and models from different species. All data are well described and can be accessed immediately for further analysis. [35]

5.4.4 KG Ebrains

The EBRAINS Knowledge Graph bases on BlueBrain Nexus which provides a multi-modal solution for an eventual consistent data store. [36]

In Figure 5.3 you can see that the components of the EBRAINS KG are originating either from BlueBrain Nexus (blue boxes) or from extensions built and/or integrated by EBRAINS (yellow boxes). The diagram shows write (blue arrows - asynchronous are dotted) and read (green arrows) operations. The arrows describe the directions of the data flow. [36]

BlueBrain Nexus

As part of the BlueBrain Nexus, Apache Cassandra stores an event log of JSON-LD messages and is the primary storage component. The in-built indexing mechanism then ensures the indexing of the JSON-LD into multiple index-databases: Blazegraph is a triple store, elasticsearch is used for full-text queries. Since the indexing mechanism works asynchronously, the databases are eventually consistent. [36]

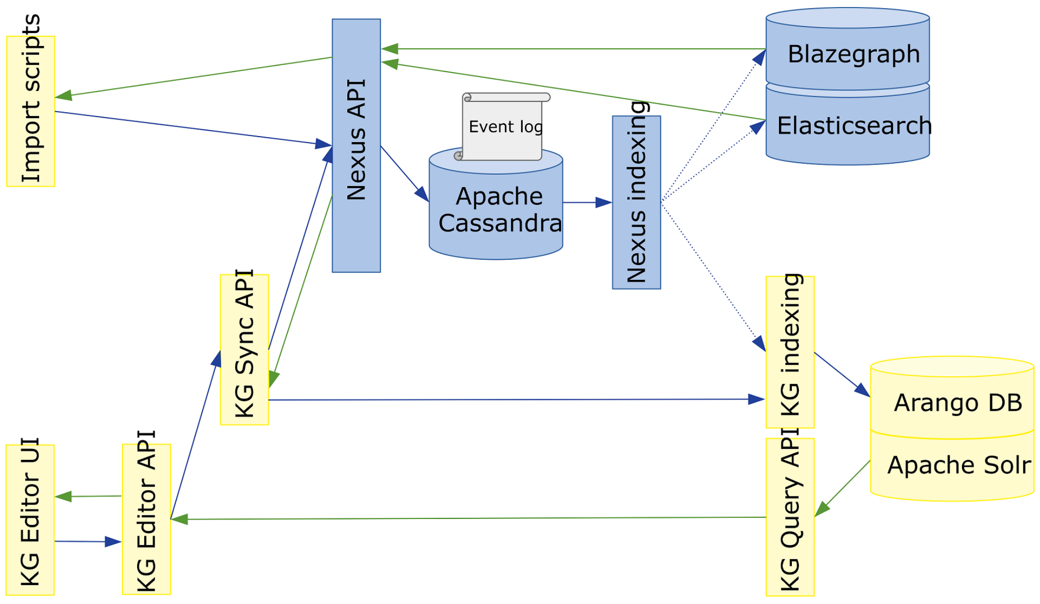


Figure 5.3: The components of the EBRAINS KG are originating either from BlueBrain Nexus (blue boxes) or from extensions built and/or integrated by EBRAINS (yellow boxes). The diagram shows write (blue arrows - asynchronous are dotted) and read (green arrows) operations. The arrows describe the directions of the data flow. [36]

KG Query API

An additional indexing client normalizes the incoming payload (full qualification), executes inference logic, indexes the data in Arango DB and interprets semantics (e.g. recognizes spatial anchoring payloads and indexes them after a rasterization it in the additional Apache Solr index). [36]

KG Editor

Although nice for scalability, the eventual consistency causes problems for applications such as the KG Editor where postponed updates can lead to confusing states on a reactive UI with data manipulation (it e.g. can happen that changes which were just applied by a user are not yet reflected in the database). The KG Sync API therefore provides a synchronous alternative API primarily created for this use-case: Creations / modifications / deletions are applied to the Arango index directly after they have been transferred in the Nexus API. Therefore, they are immediately reflected in queries of the KG Query API which allows us to provide a responsive UI. The standard indexing process will overwrite this "temporary indexing" after a while. [36]

Import scripts

Automated import scripts (typically written in Python) which load data from a specific source, transform it to the required JSON-LD structures and make use of the Nexus API to upload the data to the Knowledge Graph can be triggered externally. At EBRAINS, we're using a job scheduler who manages these kind of reoccurring jobs. [36]

5.5 Summary of repositories

From the above repositories, the GIN repository seems to be the best one; I will store the converted data there. GIN is a well-known and already proven repository that meets everything we need. Other repositories are relatively new and constantly changing, so I don't think they might be reliable now.

6 Summary

I have studied all available formats and tools for processing electrophysiological data. Before implementation, it is necessary to first summarize everything that was found in the previous chapters and decide which standards and tools choose and utilize. Before choosing a standard, I have to state the criteria according to which I compare the individual standards. The selection criteria are as follows:

- Fair principles adoption
- standard description complexity
- suitability for our lab
- size of community
- support of tools
- quality of documentation
- data from github

I will no longer compare NWB here, as I have already excluded it in the subchapter 3.5. That's why I decided between these formats and eliminated NWB. Based on the criteria of suitability for our lab I eliminated NWB.

I put BIDS and NIX in two tables, because deciding between these two standards was not easy so in the first table are numbers from the github of both standards. You can see the table below and you can see that the BIDS format shows better results in all values.

GitHub allows "watch" a project, which means you're notified whenever there are any updates. Later the company has added another level of watching, dubbed "stars," to the mix. When you star a project you can keep track of it, but you won't be notified of every change. [37] I determined the size of the community based on Table 6.1.

The next table compares other criteria that I determined at the beginning.

Both standards follow Fair Principles, differing only in a few points. You can see them in Table 3.2. I used ratings at 3 levels (bad, good and very

	NIX/odML	BIDS
Contributors	14	37
Commits	1709	1743
Watch	8	21
Star	13	91
Fork	22	67

Table 6.1: Compare NIX and BIDS by github criteria (size of community).

	NIX/odML	BIDS
Fair principles	YES	YES
standard description complexity	good	good
support of tools	good	very good
quality of documentation	good	good
suitability for our lab	good	very good

Table 6.2: Compare NIX and BIDS by selection criteria.

good) in Table 6.2. I did not use a bad evaluation even once, because all the standards described in this thesis are of good quality. I decided between good and very good based on the data I had to convert. So it was important for me what suitable methods and tools the standard has for me.

I chose BIDS from the formats described above. One of the reasons is that data from BrainVision format can be converted to BIDS format via MNE. Data can also be converted to NIX format via MNE. That was one of the factors why I decided between BIDS and NIX. But the BIDS format seems more straightforward to me, because the BrainVision format is accepted as one of BIDS default data format. NIX further parses BrainVision data, but it is not necessary in most cases. NIX needs NEO to get the data into an analytics tool. MNE is a very powerful tool that can work well with data and since it is closely related to the BIDS format, I leaned towards this format. In addition, the way through NIX has already been used at the university in one diploma thesis, so I want to go a different way.

I also described tools that work with the described standards. As I mentioned in the previous paragraph, MNE is closely related to BIDS, which can be used to convert data from BrainVision to the BIDS standard. It follows that I will use MNE for the conversion.

Next, we need to save the data that we have converted to BIDS format to a repository. A suitable repository is the G-Node's hosted GIN repository. In this repository, we can use the otherwise paid identification via DOI of the dataset, which is a key part of compliance with FAIR principles.

An overview of all standards and repositories can be seen in Figure 6.1.

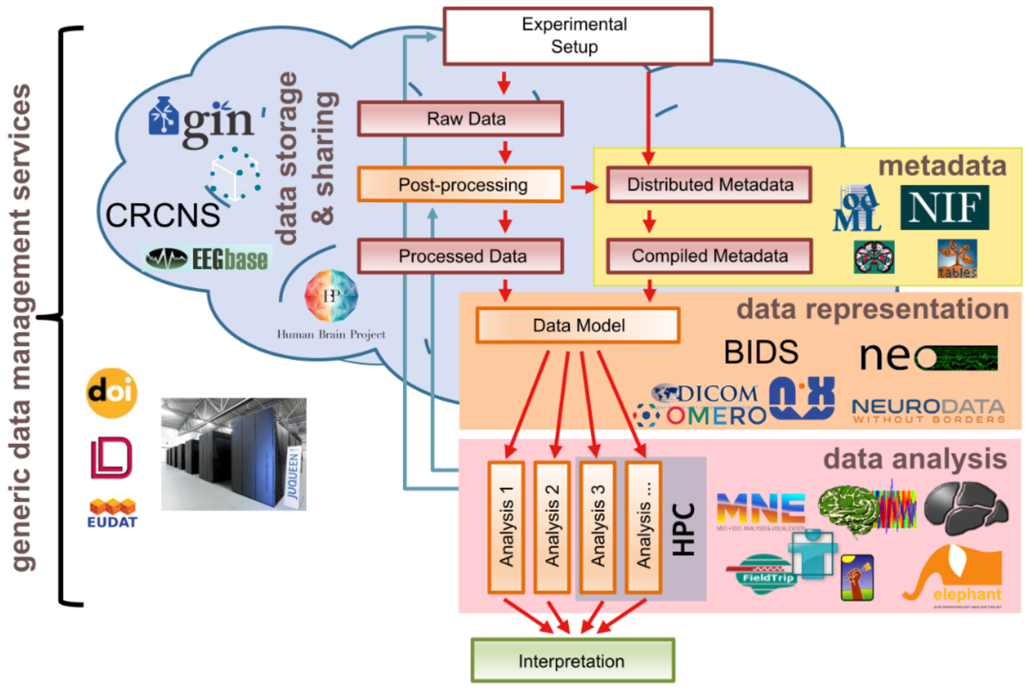


Figure 6.1: An overview of the standards, tools and repositories described above. [38].

In the next chapter I will describe the solution using the selected standard, tools and repositories. The solution will be a python tool which will convert data from the BrainVision format to BIDS and the resulting data will be stored in a GIN repository.

7 Implementation

In the previous chapters, we have described several standardization initiatives and standards describing electrophysiological data, the tools that can work with these standardization activities and we also mentioned repositories.

Now we move on to the implementation itself, when I convert BrainVision data to BIDS format by Python tools. I will create a graphical user interface for better user manipulation. So the user will easily convert BrainVision data to BIDS format.

7.1 Conversion of dataset

Now when we have the BIDS data standard selected as a suitable replacement for the BrainVision standard, we need to find a way to convert data and metadata from the previous format to the selected one. The goal is to design and implement a tool that converts data and metadata from the BrainVision standard to the BIDS standard.

7.1.1 BrainVision data

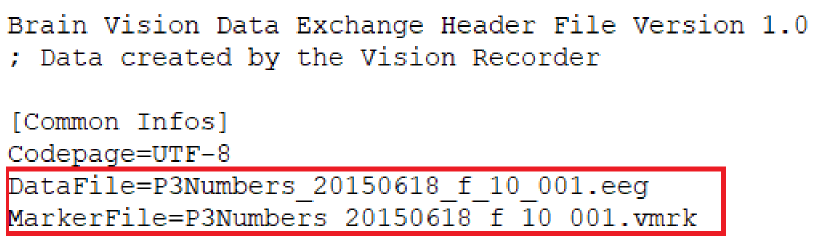
The BrainVision data format consists of three files [39]:

- A text header file (.vhdr) containing metadata
- A text marker file (.vmrk) containing information about events in the data
- A binary data file (.eeg) containing the voltage values of the EEG signal

Both text files are based on the Microsoft Windows INI format consisting of [39]:

- sections marked as [square brackets]
- comments marked as ; comment
- key-value pairs marked as key=value

In Figure 7.1 we can see that we have three separate files for each dataset. It means that the single files have internal pointers to each other's locations (see the DataFile and MarkerFile keys in Figure 7.1 in red box) [39].



```
Brain Vision Data Exchange Header File Version 1.0
; Data created by the Vision Recorder

[Common Infos]
Codepage=UTF-8
DataFile=P3Numbers_20150618_f_10_001.eeg
MarkerFile=P3Numbers_20150618_f_10_001.vmrk
```

Figure 7.1: Example of a text header file (.vhdr)

Structure of BrainVision dataset

The Neuroinformatics laboratory at the Faculty of Applied Sciences of the University of West Bohemia uses the dataset structure you can see in Figure 7.2. Next I will describe the structure that can be seen in this Figure. Each dataset contains one or more experiments. The experiment number

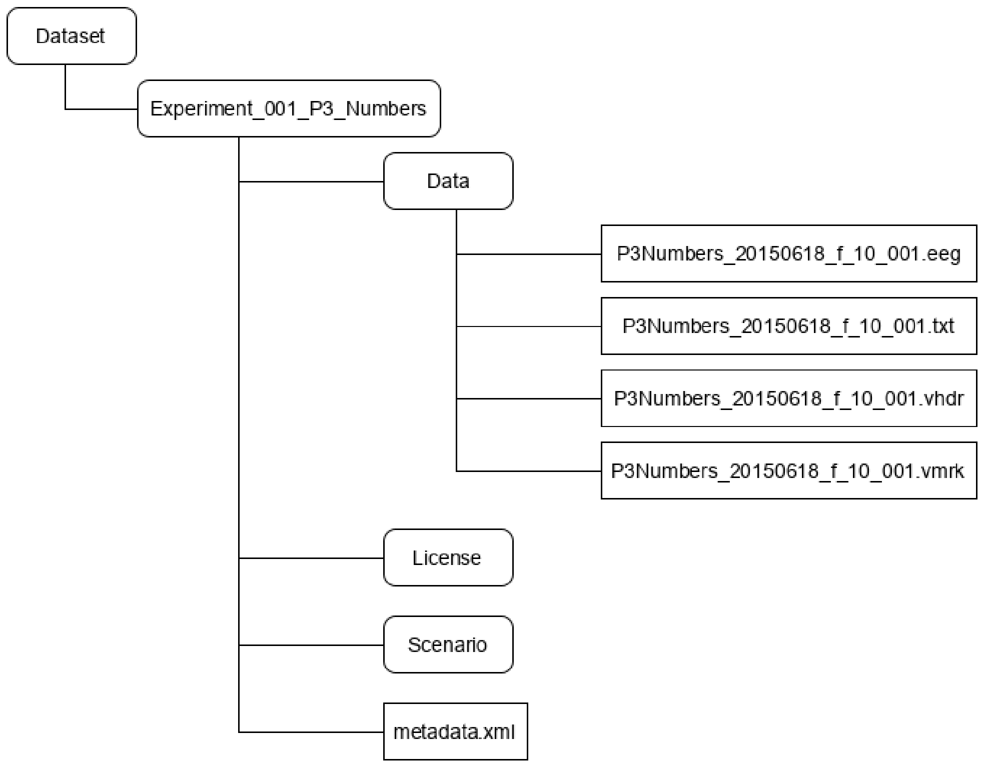

Dataset
Experiment_001_P3_Numbers
Data
P3Numbers_20150618_f_10_001.eeg
P3Numbers_20150618_f_10_001.txt
P3Numbers_20150618_f_10_001.vhdr
P3Numbers_20150618_f_10_001.vmrk
License
Scenario
metadata.xml

Figure 7.2: Directory structure of the BrainVision dataset used by the Laboratory of Neuroinformatics at the Faculty of Applied Sciences of the University of West Bohemia.

can be seen in the directory name. Each experiment contains directories Data, License, Scenario and file metadata.xml. The Laboratory of Neuroinformatics uses metadata.xml file to store metadata. This file follows the odML structure.

The Data directory contains BrainVision data. The License directory contains a license agreement in the form of a pdf file and the Scenario directory contains scenarios describing the measurement process.

Conversion of BrainVision data

Each dataset contains one or more experiments. Each experiment contains the three files listed above. To convert between the BrainVision standard and the BIDS standard, we only need a text header file (.vhdr), from which we can get a link to other two necessary files.

7.1.2 BIDS-EEG

The structure of the BIDS standard is described in subsection 3.3.1. Here we describe the individual files that are in the BIDS directory tree.

As you can see in Figure 7.3, the root contains files describing the dataset in general ("README", "dataset_description.json") and a file describing the participants ("participants.tsv") [13].

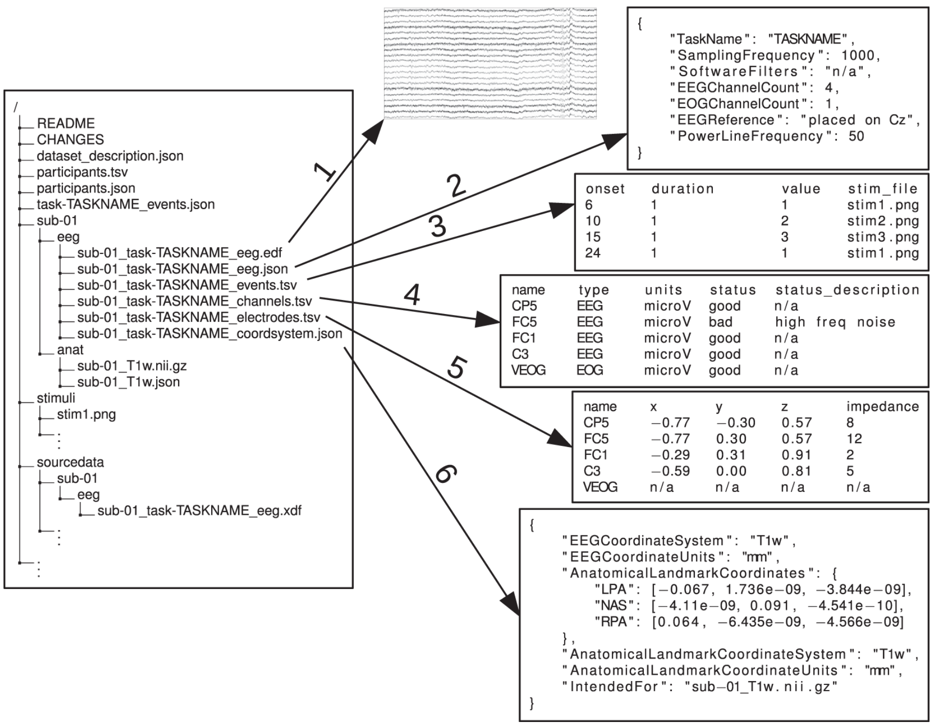
The EEG community uses a variety of formats for storing raw data, and there is no single standard that all researchers agree on. For BIDS, EEG data MUST be stored in one of the following formats [40]:

- European data format (.edf)
- BrainVision Core Data Format (.vhdr, .vmrk, .eeg) by Brain Products GmbH
- The format used by the MATLAB toolbox EEGLAB (.set and .fdt files)
- Biosemi data format (.bdf)

Figure 7.3 shows the European data format (.edf), but in our case we will use the second option, it means BrainVision Core Data Format (.vhdr, .vmrk, .eeg) by Brain ProductsGmbH.

In addition to these files, the eeg directory contains other files with a .tsv and .json extension. These files (.tsv and .json) are BIDS metadata, which makes them not only machine readable, but also human readable.

An important metadata file is the file with the suffix_eeg.json that specifies the task. Then there are the files that are recommended, such as suffix_channels.tsv. This file provides easily searchable information across BIDS datasets for e.g., general curation, response to queries or batch analysis. The required columns are channel name, type and units in this specific order. To avoid confusion, the channels should be listed in the order they appear in the EEG data file. Any number of additional columns may be added to provide additional information about the channels. Electrode positions should be added to suffix_electrodes.tsv. Coordinates are expected in cartesian coordinates according to the EEGCoordinateSystem and EEGCoordinateSystemUnits fields in suffix_coordsystem.json. If an suffix_electrodes.tsv file is specified, a suffix_coordsystem.json file must be specified as well [40].



```
/
├─ README
├─ CHANGES
├─ dataset_description.json
├─ participants.tsv
├─ participants.json
├─ task-TASKNAME_events.json
├─ sub-01
│  ├─ eeg
│  │  ├─ sub-01_task-TASKNAME_eeg.edf
│  │  ├─ sub-01_task-TASKNAME_eeg.json
│  │  ├─ sub-01_task-TASKNAME_events.tsv
│  │  ├─ sub-01_task-TASKNAME_channels.tsv
│  │  ├─ sub-01_task-TASKNAME_electrodes.tsv
│  │  └─ sub-01_task-TASKNAME_coordsystem.json
│  └─ anat
│     ├─ sub-01_T1w.nii.gz
│     └─ sub-01_T1w.json
├─ stimuli
│  ├─ stim1.png
│  └─ ⋮
├─ sourcedata
│  ├─ sub-01
│  │  └─ eeg
│  │     └─ sub-01_task-TASKNAME_eeg.xdf
│  └─ ⋮
└─ ⋮
```

```
{
    "TaskName": "TASKNAME",
    "SamplingFrequency": 1000,
    "SoftwareFilters": "n/a",
    "EEGChannelCount": 4,
    "EOGChannelCount": 1,
    "EEGReference": "placed on Cz",
    "PowerLineFrequency": 50
}
```

onset	duration	value	stim_file
6	1	1	stim1.png
10	1	2	stim2.png
15	1	3	stim3.png
24	1	1	stim1.png

name	type	units	status	status_description
CP5	EEG	microV	good	n/a
FC5	EEG	microV	bad	high freq noise
FC1	EEG	microV	good	n/a
C3	EEG	microV	good	n/a
VEOG	EOG	microV	good	n/a

name	x	y	z	impedance
CP5	−0.77	−0.30	0.57	8
FC5	−0.77	0.30	0.57	12
FC1	−0.29	0.31	0.91	2
C3	−0.59	0.00	0.81	5
VEOG	n/a	n/a	n/a	n/a

```
{
    "EEGCoordinateSystem": "T1w",
    "EEGCoordinateUnits": "mm",
    "AnatomicalLandmarkCoordinates": {
        "LPA": [−0.067, 1.736e−09, −3.844e−09],
        "NAS": [−4.11e−09, 0.091, −4.541e−10],
        "RPA": [0.064, −6.435e−09, −4.566e−09]
    },
    "AnatomicalLandmarkCoordinateSystem": "T1w",
    "AnatomicalLandmarkCoordinateUnits": "mm",
    "IntendedFor": "sub−01_T1w.nii.gz"
}
```

Figure 7.3: Exemplary EEG-BIDS dataset with previews of EEG files. The left side of the figure shows a standard BIDS directory tree, The right side of the figure shows the contents of the eeg modality directory. Figure shows the European data format (.edf), but in our case we will use the second option, it means BrainVision Core Data Format (.vhdr, .vmrk, .eeg) by Brain ProductsGmbH. [13]

7.1.3 MNE

As you can see in Figure 7.4, BIDS can communicate with MNE. We will therefore use MNE to load data from the BrainVision format and then save the data to the BIDS structure.

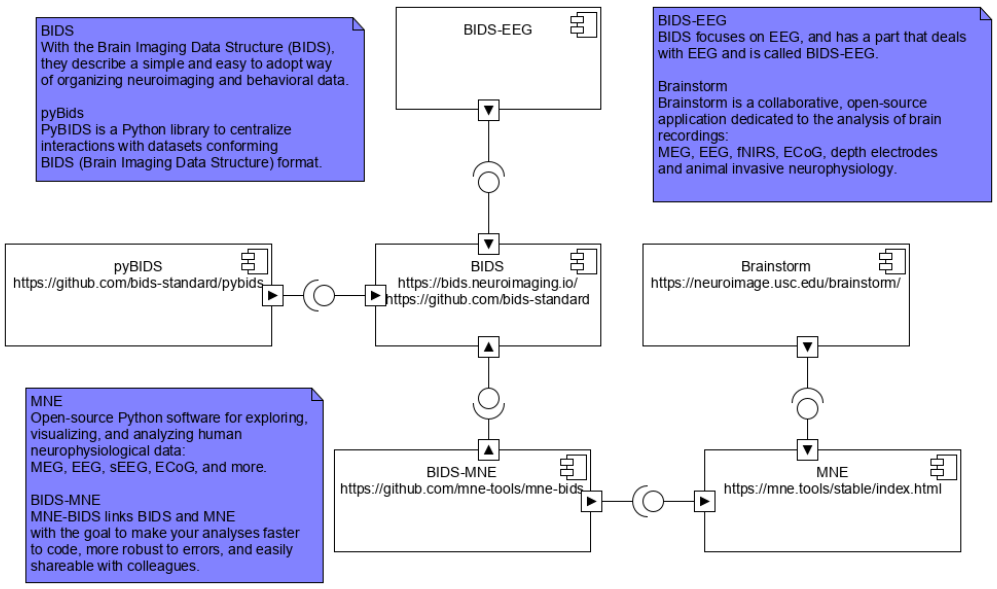

BIDS
With the Brain Imaging Data Structure (BIDS), they describe a simple and easy to adopt way of organizing neuroimaging and behavioral data.
pyBids
PyBIDS is a Python library to centralize interactions with datasets conforming BIDS (Brain Imaging Data Structure) format.
BIDS-EEG
BIDS-EEG
BIDS focuses on EEG, and has a part that deals with EEG and is called BIDS-EEG.
Brainstorm
Brainstorm is a collaborative, open-source application dedicated to the analysis of brain recordings: MEG, EEG, fNIRS, ECoG, depth electrodes and animal invasive neurophysiology.
pyBIDS https://github.com/bids-standard/pybids
BIDS https://bids.neuroimaging.io/ https://github.com/bids-standard
Brainstorm https://neuroimage.usc.edu/brainstorm/
MNE
Open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more.
BIDS-MNE
MNE-BIDS links BIDS and MNE with the goal to make your analyses faster to code, more robust to errors, and easily shareable with colleagues.
BIDS-MNE https://github.com/mne-tools/mne-bids
MNE https://mne.tools/stable/index.html

Figure 7.4: Component diagram of BIDS

MNE allows us to load BrainVision eeg data, which is a key component for us when converting data to the BIDS format. To load the data we will use the method:

- **mne.io.read_raw_brainvision (vhdr_fname, eog = 'HEOGL', 'HEOGR', 'VEOGb', misc = 'auto', scale = 1.0, preload = False, verbose = None)**

I explained what the individual parameters mean in appendix A.

The method returns a Raw object containing BrainVision data. When we have the data in the raw object, we can use the method from MNE-BIDS that saves raw data to a BIDS-compliant folder structure. The method has this form:

- **mne_bids.write_raw_bids(raw, bids_basename, bids_root, events_data=None, event_id=None, anonymize=None, overwrite=False, verbose=True)**

I again described the individual parameters of the method in appendix A.

After these actions, we have the data converted to BIDS format. Now we still have to convert metadata from BrainVision format and odMl structure to BIDS.

7.2 Conversion of metadata

7.2.1 BrainVision metadata

The Laboratory of Neuroinformatics use metadata.xml file to store metadata. This file follows the odML structure.

7.2.2 BIDS metadata

Metadata in the BIDS format is stored in several files and has a given structure. These are the following files:

- *_eeg.json
- *_channels.tsv
- *_electrodes.tsv
- *_coordsystem.json

BIDS has clearly defined fields that must be listed in the above files, then recommended and optional fields. We will describe the required fields.

*_eeg.json

This file should be present and has the following required fields:

- **TaskName** - Name of the task (for resting state use the rest prefix). No two tasks should have the same name. The task label included in the file name is derived from this TaskName field by removing all non-alphanumeric ([a-zA-Z0-9]) characters. For example TaskName faces n-back will correspond to task label facesnback [40].
- **EEGReference** - General description of the reference scheme used and (when applicable) of location of the reference electrode in the raw recordings (e.g., "left mastoid", "Cz", "CMS"). If different channels have a different reference, this field should have a general description and the channel specific reference should be defined in the _channels.tsv file [40].

- **SamplingFrequency** - Sampling frequency (in Hz) of all the data in the recording, regardless of their type (e.g., 2400) [40].
- **PowerLineFrequency** - Frequency (in Hz) of the power grid at the geographical location of the EEG instrument (i.e., 50 or 60)[40].
- **SoftwareFilters** - A JSON object of temporal software filters applied, or "n/a" if the data is not available. Each key:value pair in the JSON object is a name of the filter and an object in which its parameters are defined as key:value pairs. E.g., "Anti-aliasing filter": "half-amplitude cutoff (Hz)": 500, "Roll-off": "6dB/Octave" [40].

***_channels.tsv**

This file is recommended as it provides easily searchable information across BIDS datasets for e.g., general curation, response to queries or batch analysis. The required columns are channel name, type and units in this specific order. To avoid confusion, the channels should be listed in the order they appear in the EEG data file. Any number of additional columns may be added to provide additional information about the channels. Note that electrode positions should not be added to this file, but to *_electrodes.tsv [40].

***_electrodes.tsv**

File that gives the location of EEG electrodes. Note that coordinates are expected in cartesian coordinates according to the EEGCoordinateSystem and EEGCoordinateSystemUnits fields in *_coordsystem.json. If an *_electrodes.tsv file is specified, a *_coordsystem.json file must be specified as well. The order of the required columns in the *_electrodes.tsv file must be as listed below. This file has the following required fields [40]:

- **name** - Name of the electrode
- **x** - Recorded position along the x-axis
- **y** - Recorded position along the y-axis
- **z** - Recorded position along the z-axis

***_coordsystem.json**

A *_coordsystem.json file is used to specify the fiducials, the location of anatomical landmarks, and the coordinate system and units in which the position of electrodes and landmarks is expressed. The *_coordsystem.json is required if the optional *_electrodes.tsv is specified. If a corresponding anatomical MRI is available, the locations of landmarks and fiducials according to that scan should also be stored in the *_T1w.json file which goes alongside the MRI data [40]. This file has the following required fields:

- **EEGCoordinateSystem** - Refers to the coordinate system in which the EEG electrode positions are to be interpreted [40].
- **EEGCoordinateUnits** - Units in which the coordinates that are listed in the field EEGCoordinateSystem are represented (e.g., "mm", "cm")[40].
- **EEGCoordinateSystemDescription** - Free-form text description of the coordinate system. May also include a link to a documentation page or paper describing the system in greater detail [40].

In addition to these files, which are unique to each experiment, BIDS also has files that are global to the entire dataset. The first such file is dataset_description.json, which lists the authors, BIDS version, and dataset name. Then there are the files participants.json and participants.tsv. The participants.json file describes what the individual abbreviations in the participants.tsv file mean. There is a table in the participants.tsv file, where each row represents a participant in one experiment. The columns are participants_id, age, sex, hand.

7.2.3 Conversion metadata from BrainVision to BIDS

As described in Subsection 7.1.3, the method from the MNE library converts data from BrainVision to BIDS. However, this method also converts some metadata that is discoverable from the *.vhdr file. Thanks to this, we already get a BIDS structure with the required files. However, some fields are empty because this data are not available from the .vhdr file. Unfilled data must be obtained from the file metadata.xml, which contains all the important information about the measurement. Then there is a .txt file that has the same name as the .vhdr file. This file contains information about the person being measured, such as gender, age and handedness. Because it is not possible to map all the information from metadata.xml, this file is additionally copied to the sourcedata directory, which BIDS recommends for these cases so that no essential information is lost. In any case, some information needed to be mapped to the BIDS structure. This mapping has therefore been implemented and the created python tool can obtain important information from metadata.xml and add it to the BIDS structure. Specifically, these information are the names of the authors, the name of the dataset and information about the participants. Another mapped information are the EEGReference and EEGGround values found in the sub-*_task-*_ eeg.json file.

7.3 Conversion tool

Libraries that can convert data between BrainVision format and BIDS format are written in python. For this reason, the tool is also written in python. The conversion tool consists of several classes. It contains a GUI class, which mainly takes care of the graphical user interface, ie the visual part of the tool. Furthermore, the BrainVisionConverter class, which has the task of converting data from BrainVision to BIDS. The MetadataConvert class is implemented for metadata conversion. The last class is the ProgressBar, which starts the progress bar, ie shows the user how much of the conversion is done. A more detailed description of the classes will follow. You can see the class diagram in Figure 7.5

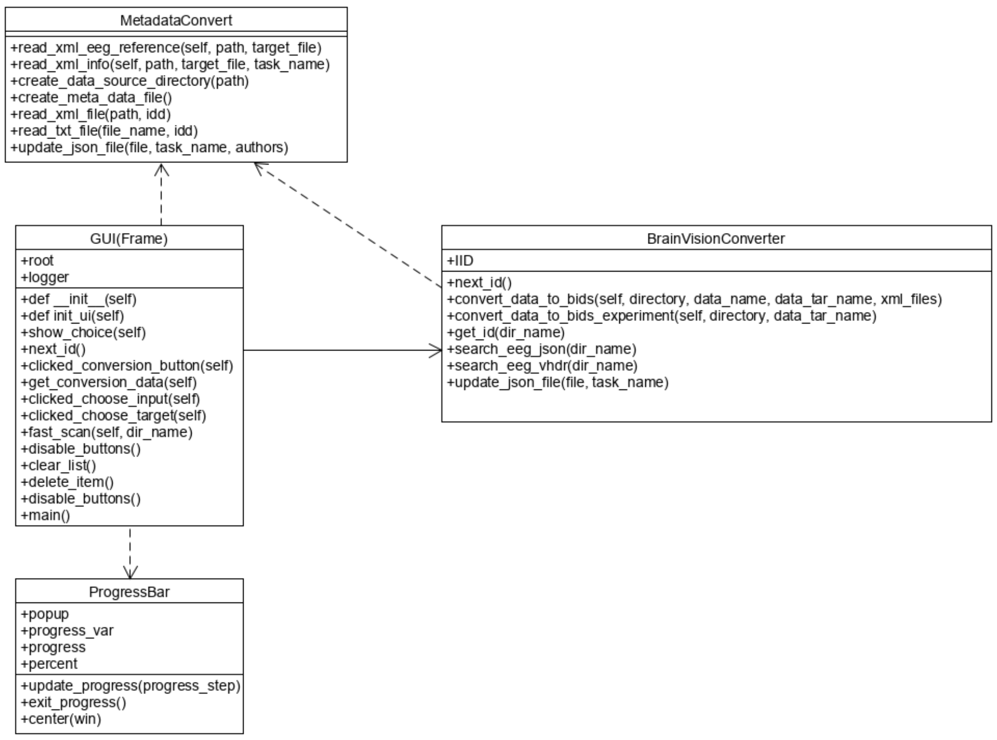

MetadataConvert
+read_xml_eeg_reference(self, path, target_file)
+read_xml_info(self, path, target_file, task_name)
+create_data_source_directory(path)
+create_meta_data_file()
+read_xml_file(path, idd)
+read_txt_file(file_name, idd)
+update_json_file(file, task_name, authors)
GUI(Frame)
+root
+logger
+def __init__(self)
+def init_ui(self)
+show_choice(self)
+next_id()
+clicked_conversion_button(self)
+get_conversion_data(self)
+clicked_choose_input(self)
+clicked_choose_target(self)
+fast_scan(self, dir_name)
+disable_buttons()
+clear_list()
+delete_item()
+disable_buttons()
+main()
BrainVisionConverter
+IID
+next_id()
+convert_data_to_bids(self, directory, data_name, data_tar_name, xml_files)
+convert_data_to_bids_experiment(self, directory, data_tar_name)
+get_id(dir_name)
+search_eeg_json(dir_name)
+search_eeg_vhdr(dir_name)
+update_json_file(file, task_name)
ProgressBar
+popup
+progress_var
+progress
+percent
+update_progress(progress_step)
+exit_progress()
+center(win)

Figure 7.5: Class diagram of the conversion tool.

7.3.1 GUI

This class provides a graphical user interface to make the application easier for users. It contains methods that handle the buttons. These methods call methods from other classes that convert the data. You can see the main window of conversion tool in Figure 7.6

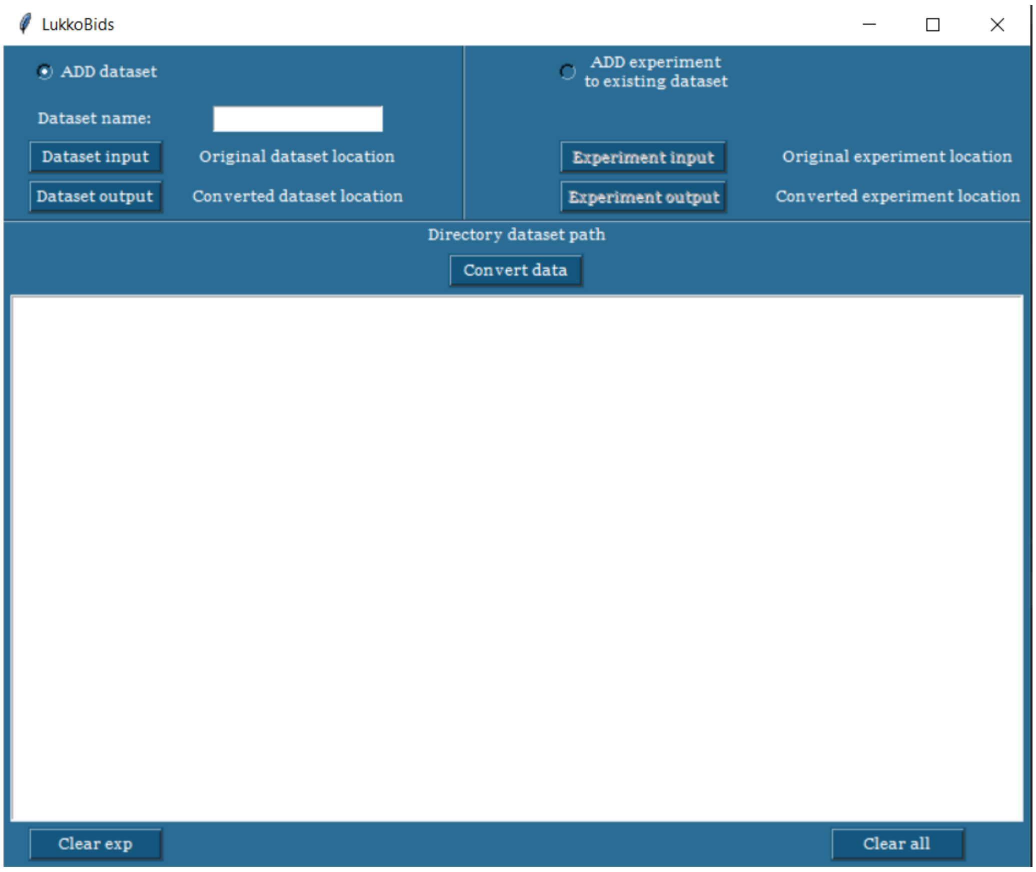


Figure 7.6: A tool window after its launch.

7.3.2 BrainVisionConverter

It uses methods from the MNE library to convert data from BrainVision to BIDS. In addition, it adds data to the *_eeg.json file that was created by the MNE method.

7.3.3 MetadataConvert

As mentioned in the previous chapter, metadata is stored in a metadata.xml file and a .txt file. So there are methods in this class that get the necessary data from these two files and store them in BIDS files.

Specifically, these are methods that write data from a .txt file to participants.tsv. These files are already described above, they are files in which information about the participant of the experiment is stored. Furthermore, the names of authors who are written to the dataset_description.json file are obtained from the metadata.xml file.

There is also a method in this class that copies the metadata.xml file to the sourcedata directory, where metadata are stored.

7.3.4 ProgressBar

The ProgressBar class shows the user how much of the conversion is done. The class consists of a start method that starts a new window with a progress bar, with a method that places this window in the center of the screen. Furthermore, a method that updates the status of the progress bar and a method that terminates the progress bar when finished.

7.4 Implementation summary

The MNE library was used to convert data from the BrainVision format to the BIDS format. The methods from this library also created an entire structure that did not have all the important fields filled in.

Therefore, it was necessary to retrieve metadata from other files and map it to BIDS files. Not all metadata could be mapped, which doesn't matter because the BIDS format thought of it, and for these cases it allows you to save metadata from other formats to the sourcedata directory, so we didn't lose any data during the conversion. In any case, some information needed to be mapped to the BIDS structure. It was therefore necessary to study the structure of BIDS metadata and the content of the metadata.xml file. I had to derive some names because there was no match in the names in the BIDS format and in the metadata.xml file, but it was the same data. Specifically, the information from metadata.xml are the names of the authors, the name of the dataset, the EEGReference and EEGGround values. Information about participants (age, gender and handedness) from .txt file was used. After finding out which metadata could be mapped, I had to implement this mapping.

You can see the directory structure of the BIDS dataset after conversion from the BrainVision dataset in Figure 7.7. The sourcedata directory contains, as already mentioned, the metadata.xml files from the individual experiments.

You can see the directory structure of the BIDS dataset in illustrative Figure 7.3. It contains file in .edf format (European data format), but our directory structure after conversion from the BrainVision dataset contains instead the .edf file, .vhdr, .vmrk and .eeg files (BrainVision Core Data Format). The sourcedata directory contains, as already mentioned, the metadata.xml files from the individual experiments.

Then there are directories with individual experiments. The experiment contains the file sub-*_scans.tsv, which is already described above, and the subfolder eeg, which contains the files that are described in Section 7.1.2.

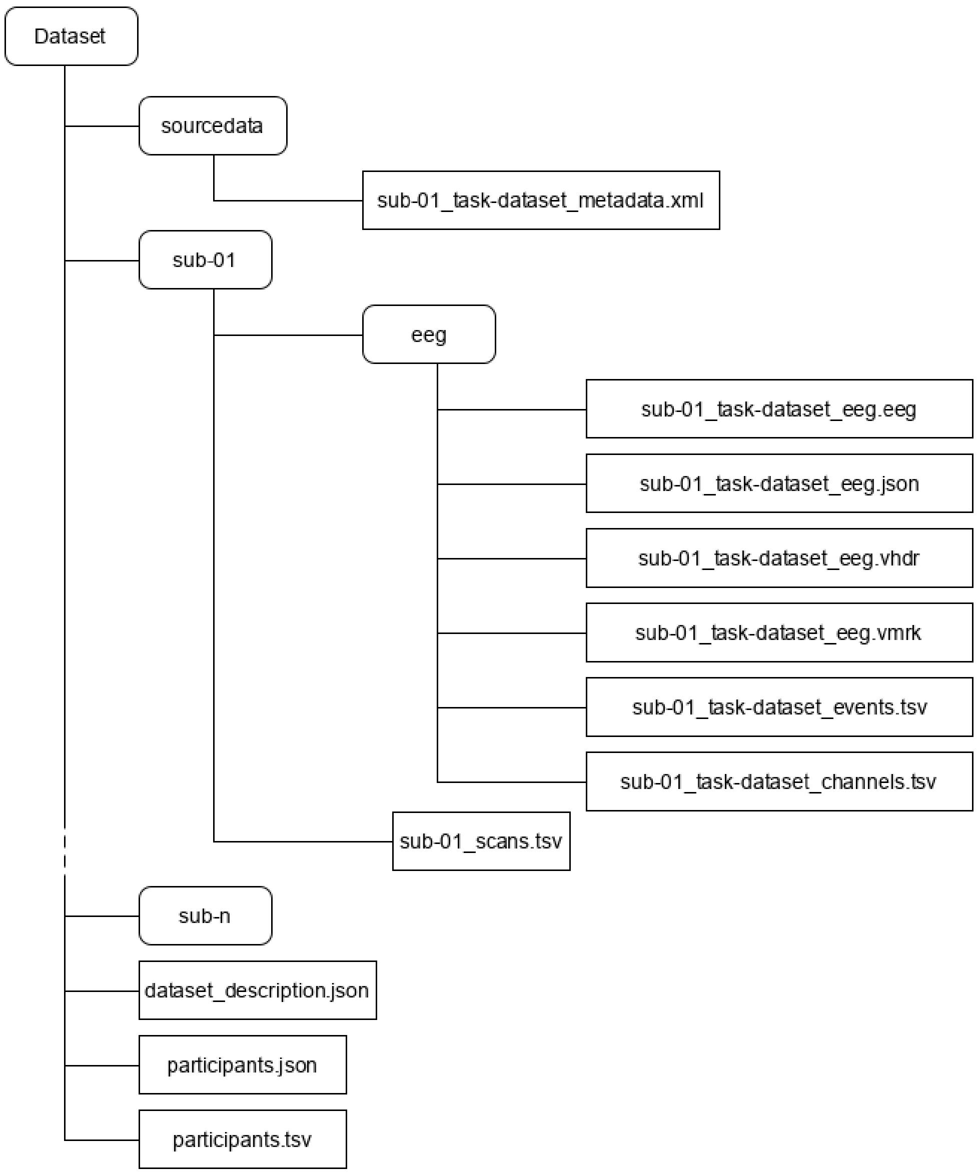

Dataset
sourcedata
sub-01_task-dataset_metadata.xml
sub-01
eeg
sub-01_task-dataset_eeg.eeg
sub-01_task-dataset_eeg.json
sub-01_task-dataset_eeg.vhdr
sub-01_task-dataset_eeg.vmrk
sub-01_task-dataset_events.tsv
sub-01_task-dataset_channels.tsv
sub-01_scans.tsv
sub-n
dataset_description.json
participants.json
participants.tsv

Figure 7.7: Directory structure of the BIDS dataset after conversion from a BrainVision dataset.

8 GIN repository

We need to save the data that we have converted to the BIDS format to a repository. A suitable repository is the G-Node's hosted GIN repository. In this repository, we can use the otherwise paid identification of DOIdataset, which is a key part of compliance with FAIR principles.

First, a new GIN account was created using the registration form; it requires only username, password, and a valid email address, but adding name and affiliation is recommended by GIN.

Furthermore, GIN-CLI was used, which is a command line client for interfacing with repositories hosted on GIN. It offers a simplified interface for downloading and uploading files from repositories hosted on GIN [41]. First, you have to log in to the GIN-CLI using the *gin login* command. Local directory for the repository is initialized using the *gin get* command. Then we move to the local directory. After that, the converted datasets can be copied into this directory. Local gin commit adds all files with a descriptive comment. In the last step, the local commit is uploaded into the remote repository. All described steps are clearly visible in Listing 8.1

```
gin login
gin get lukko45/dataset
cd dataset
gin commit --message "new dataset upload" .
gin upload
```

Listing 8.1: Step by step how to upload a dataset to a remote repository

In order to obtain a DOI, we need to have public repositories, which we achieve by unchecking private in setting a repository in a gin environment. In addition, we need to add the *datacite.yml* file into the root of the GIN repository.

The *datacite.yml* file must contain at least the following entries:

- **Authors** - Authors are the main researchers involved in working on the data, or the authors of the publication in order of priority. A first and last name are required. Additionally, affiliation and ID (digital identifier, e.g. ORCID) are highly recommended [42].

- **Title** - The title is the descriptive name of the dataset to be published. Line breaks may not be used in the title field [42].

- **Description** - The description contains extended information about dataset [42].
- **Keywords** - The keywords entry should be used to list terms the dataset is associated with [42].
- **License** - The license entry specifies the license under which the data-set will be published [42].

References and *funding* may also be used to provide additional information.
You can see an example of *datacite.yml* in the yml code 8.1

```yaml
authors:
  -
firstname: "Lukas"
lastname: "Scurko"
affiliation: "Faculty of Applied Sciences,
              University of West Bohemia"
id: "ORCID:0000-0001-2345-6789"

title: "PROJECT_DAYS_P3_NUMBERS"

description: |
  PROJECT_DAYS_P3_NUMBERS
  that was converted from BrainVision to BIDS.

keywords:
  - Neuroscience
  - Electrophysiology
  - ERP

license:
  name: "Creative Commons CC0 1.0 Public Domain Dedication"
  url: "https://creativecommons.org/publicdomain/zero/1.0/"

resourcetype: Dataset

templateversion: 1.2
```

yml code 8.1: An example of a datacite.yml file

Once a valid DataCite file named datacite.yml has been uploaded to the root of a public repository, a preview of the contents is rendered below the README section in the repository overview. A DOI is requested by clicking the "Request DOI" button [42].

9 Evaluation

In this chapter, we describe the code quality of the electrophysiological data conversion tool.

The conversion tool should be tested at the level of static code analysis, how well the code is written. Subsequently, the functionality of the code should be tested. I will verify the functionality using unit tests. Finally, it is necessary to verify that the converted data is really in the BIDS format and this will be ensured by the BIDS validator.

9.1 Static analysis of the code

First of all, the code was checked by pep8. Pep8[1] is a tool to check the Python code against some of the style conventions in PEP 8. Pep8 directly underlines in the code the part that does not meet the given rules. This tool defines and checks the conventions like Names to Avoid, ASCII Compatibility, Package and Module Names, Class Names, Type Variable Names, Exception Names,Global Variable Names,Function and Variable Names,Function and Method Arguments, Method Names and Instance Variables, Constants, Designing for Inheritance. You can read the description of these conventions at the link in the footnote[2]. This tool did not find any error in the code.

Another static parser was pylint, which threw out a few warnings about the use of global variables that are used in the code. If these warnings were omitted, the code was rated 10 out of 10.

9.2 BIDS standard

The structure into which the implemented tool transferred data was tested by the online BIDS Validator[3]. The entire dataset, which contained 251 experiments, was uploaded to the BIDS validator. The uploaded dataset was the PROJECT_DAYS_P3_NUMBERS. This dataset passed the validator without a single error. Furthermore, other datasets were created, to which individual experiments were added and the result of the validator was the same, ie 0 errors.

[1]`https://www.python.org/dev/peps/pep-0008/`.

[2]`https://www.python.org/dev/peps/pep-0008/#naming-conventions`.

[3]`https://bids-standard.github.io/bids-validator/`.

9.3 UNIT tests

Unit tests have been written to check the individual methods. For unit tests, two files are created that check if the methods are doing what they are supposed to do. Methods for finding metadata.xml files, .vhdr files, and .txt files that are needed for conversion have been tested. Furthermore, the six methods that convert metadata from metadata.xml and .txt files have been tested by unit tests.

10 Conclusion

Standards for storing electrophysiological data were studied. The BIDS standard was chosen from the analysis, which compared standards at both the code and community levels and also compared the complexity of converting current data to a given standard.

This standard has a specified structure into which the data needed to be converted. The MNE library was used for this, which was used to convert data and some metadata. Other metadata needed to be mapped according to the BIDS specification. Metadata that could not be mapped was also stored in the BIDS structure, as BIDS remembers these cases and allows other standards to store metadata in the specified directory. As a result, no data was lost. The BIDS structure was checked by a BIDS validator, which did not find any errors.

A tool was implemented for the described data conversion and metadata. The tool is written in python, supporting libraries such as MNE are also written in python. A graphical user interface has been implemented to work with the converter. The tool was tested with two static analyzers and unit tests were also written. All code, including the necessary libraries, is stored on github.

G-Node's GIN was chosen to store the data. In this repository, we can use the otherwise paid identification of DOIdataset, which is a key part of compliance with FAIR principles.

Bibliography

[1] A. Roque, A. Tomczyk, E. D. Maria, F. Putze, R. Moucek, A. Fred, and H. Gamboa, *Biomedical engineering systems and technologies: 12th International Joint Conference, BIOSTEC 2019, Prague, Czech Republic, February 22-24, 2019, revised selected papers.* Springer, 2020.

[2] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, “The fair guiding principles for scientific data management and stewardship,” *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[3] FORCE11. (2020) The fair data principles. Accessed on: 2019/10/17. [Online]. Available: https://www.force11.org/group/fairgroup/fairprinciples

[4] A. Trifan and J. Oliveira, “Fairness in biomedical data discovery,” 01 2019, pp. 159–166.

[5] G-NODE. (2015) About nix. Accessed on: 2019/10/18. [Online]. Available: https://github.com/G-Node/nix/wiki/About

[6] G-Node. (2015) The model. Accessed on: 2019/10/21. [Online]. Available: https://github.com/G-Node/nix/wiki/The-Model

[7] J. Grewe, T. Wachtler, and J. Benda, “A bottom-up approach to data annotation in neurophysiology,” *Frontiers in neuroinformatics*, vol. 5, p. 16, 2011.

[8] J. Grewe and T. Wachtler, “odml format and terminologies for automated handling of (meta) data,” in *Front. Neurosci. Conference Abstract: Neuroinformatics*, vol. 2010, 2010.

[9] G-NODE. The odml data model. Accessed on: 2019/10/17. [Online]. Available: http://g-node.github.io/python-odml/data_model.html

[10] BIDS. About bids. Accessed on: 2019/10/18. [Online]. Available: https://bids.neuroimaging.io/

[11] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko *et al.*, “The brain imaging data structure, a format for organizing and

describing outputs of neuroimaging experiments," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[12] C. Holdgraf, S. Appelhoff, S. Bickel, K. Bouchard, S. D'Ambrosio, O. David, O. Devinsky, B. Dichter, B. Foster, C. Gorgolewski *et al.*, "Bids-ieeg: an extension to the brain imaging data structure (bids) specification for human intracranial electrophysiology," 2018.

[13] C. R. Pernet, S. Appelhoff, K. J. Gorgolewski, G. Flandin, C. Phillips, A. Delorme, and R. Oostenveld, "Eeg-bids, an extension to the brain imaging data structure for electroencephalography," *Scientific Data*, vol. 6, no. 1, pp. 1–5, 2019.

[14] NWB. Nwb: Neurophysiology. Accessed on: 2019/10/28. [Online]. Available: https://www.nwb.org/nwb-neurophysiology/

[15] S. Das, A. P. Zijdenbos, D. Vins, J. Harlap, and A. C. Evans, "Loris: a web-based data management system for multi-center studies," *Frontiers in neuroinformatics*, vol. 5, p. 37, 2012.

[16] Elephant. Elephant. Accessed on: 2019/10/21. [Online]. Available: https://elephant.readthedocs.io/en/latest/overview.html

[17] MNE. Mne. Accessed on: 2019/11/18. [Online]. Available: https://mne.tools/stable/index.html

[18] NEO. (2020) Neo. Accessed on: 2020/04/26. [Online]. Available: https://neuralensemble.org/neo/

[19] C. Gorgolewski. What is a bids app? Accessed on: 2020/02/04. [Online]. Available: http://bids-apps.neuroimaging.io/about/

[20] K. J. Gorgolewski, F. Alfaro-Almagro, T. Auer, P. Bellec, M. Capotă, M. M. Chakravarty, N. W. Churchill, A. L. Cohen, R. C. Craddock, G. A. Devenyi *et al.*, "Bids apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods," *PLoS computational biology*, vol. 13, no. 3, p. e1005209, 2017.

[21] K. Gorgolewski and F. Alfaro-Almagro. (2017) Bids apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods. Accessed on: 2020/05/10. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005209

[22] BIDS. (2019) Bids-validator. Accessed on: 2020/02/05. [Online]. Available: https://github.com/bids-standard/bids-validator

[23] UCL. Statistical parametric mapping. Accessed on: 2020/02/26. [Online]. Available: https://www.fil.ion.ucl.ac.uk/spm/

[24] BIDS. (2020) Pybids. Accessed on: 2020/02/26. [Online]. Available: https://github.com/bids-standard/pybids

[25] PyBIDS. What is pybids? Accessed on: 2020/02/26. [Online]. Available: https://bids-standard.github.io/pybids/introduction.html

[26] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen *et al.*, "Meg and eeg data analysis with mne-python," *Frontiers in neuroscience*, vol. 7, p. 267, 2013.

[27] MNE. Mne-bids. Accessed on: 2020/02/27. [Online]. Available: https://1697-89170358-gh.circle-artifacts.com/0/html/index.html

[28] G-NODE. (2019) What is gin? Accessed on: 2020/04/26. [Online]. Available: https://gin.g-node.org/G-Node/Info/wiki

[29] C. Garbers and J. Grewe. G-node projects. Accessed on: 2020/04/26. [Online]. Available: https://g-node.github.io/

[30] G-NODE. Searching data on gin. Accessed on: 2021/02/26. [Online]. Available: https://gin.g-node.org/G-Node/Info/wiki/Search

[31] G-node. gin-proc. Accessed on: 2021/02/26. [Online]. Available: https://github.com/G-Node/gin-proc

[32] DANDI. (2020) Dandi: Distributed archives for neurophysiology data integration. Accessed on: 2020/04/26. [Online]. Available: https://www.dandiarchive.org/

[33] Resonant. The resonant data and analytics platform. Accessed on: 2021/02/26. [Online]. Available: https://resonant.kitware.com/

[34] EOSC. (2020) Eosc. Accessed on: 2020/04/26. [Online]. Available: https://www.eosc-portal.eu/about/eosc

[35] Ebrains. Ebrains. Accessed on: 2020/04/26. [Online]. Available: https://ebrains.eu/services/

[36] EBRAINS. (2019) Deep dive. Accessed on: 2020/05/24. [Online]. Available: https://kg.ebrains.eu/deepdive.html

[37] Wired. Github changes make it easier to track your favorite projects. Accessed on: 2021/04/13. [Online]. Available: https://www.wired.com/2012/08/github-changes-make-it-easier-to-track-your-favorite-projects/

[38] A. Ad and T. Bronger. National research data infrastructure for the neurosciences. Accessed on: 2020/07/26. [Online]. Available: https://www.nfdi-neuro.de/files/2019-12-03_Wachtler.pdf

[39] Fieldtrip. Getting started with brainvision analyzer and easycap. Accessed on: 2020/07/13. [Online]. Available: http://www.fieldtriptoolbox.org/getting_started/brainvision/#getting-started-with-brainvision-analyzer-and-easycap

[40] BIDS. Electroencephalography. Accessed on: 2020/05/21. [Online]. Available: https://bids-specification.readthedocs.io/en/stable/04-modality-specific-files/03-electroencephalography.html

[41] G-Node. (2019) Gin-cli. Accessed on: 2020/04/10. [Online]. Available: https://github.com/G-Node/gin-cli

[42] G-Node. (2020) Obtaining a doi (digital object identifier). Accessed on: 2020/06/20. [Online]. Available: https://gin.g-node.org/G-Node/Info/wiki/DOIfile#structuring-the-datacite-file

[43] MNE. mne.io.read_raw_brainvision. Accessed on: 2020/07/15. [Online]. Available: https://mne.tools/dev/generated/mne.io.read_raw_brainvision.html

[44] MNE. mne_bids.write_raw_bids. Accessed on: 2020/07/14. [Online]. Available: https://mne.tools/mne-bids/stable/generated/mne_bids.write_raw_bids.html

[45] MNE. The typical m/eeg workflow. Accessed on: 2020/07/18. [Online]. Available: https://mne.tools/dev/overview/cookbook.html

[46] NEO. (2020) Neo core. Accessed on: 2020/04/27. [Online]. Available: https://neo.readthedocs.io/en/stable/core.html

Appendices

A MNE methods

The MNE methods that were used in the python tool to convert data from the BrainVision standard to the BIDS standard are described in this chapter.

- **mne.io.read_raw_brainvision (vhdr_fname, eog = 'HEOGL', 'HEOGR', 'VEOGb', misc = 'auto', scale = 1.0, preload = False, verbose = None)**

We will now explain what the individual parameters mean:

- **vhdr_fname** - Path to the EEG header file (.vhdr) [43].
- **eog** - Names of channels or list of indices that should be designated EOG channels. Values should correspond to the vhdr file. Default is ('HEOGL', 'HEOGR', 'VEOGb') [43].
- **misc** - Names of channels or list of indices that should be designated MISC channels (other analog channels for auxiliary signals). Values should correspond to the electrodes in the vhdr file. If 'auto', units in vhdr file are used for inferring misc channels. Default is 'auto' [43].
- **scale** - The scaling factor for EEG data. Unless specified otherwise by header file, units are in microvolts. Default scale factor is 1[43].
- **preload** - Preload data into memory for data manipulation and faster indexing. If True, the data will be preloaded into memory (fast, requires large amount of memory). If preload is a string, preload is the file name of a memory-mapped file which is used to store the data on the hard drive (slower, requires less memory) [43].
- **verbose** - If not None, override default verbose level [43].

The method returns a Raw object containing BrainVision data. When we have the data in the raw object, we can use the method from MNE-BIDS that saves raw data to a BIDS-compliant folder structure. The method has this form:

- **mne_bids.write_raw_bids(raw, bids_basename, bids_root, events_data=None, event_id=None, anonymize=None, overwrite=False, verbose=True)**

We will again describe the individual parameters of the method:

- **raw** - The raw data. It must be an instance of mne.Raw. The data should not be loaded from disk, i.e., raw.preload must be False [44].
- **bids_basename** - The base filename of the BIDS compatible files. Typically, this can be generated using make_bids_basename [44].
- **bids_root** - The path of the root of the BIDS compatible folder. The session and subject specific folders will be populated automatically by parsing bids_basename [44].
- **events_data** - The events file. If a string or a Path object, specifies the path of the events file. If an array, the MNE events array (shape n_events, 3). If None, events will be inferred from the stim channel using mne.find_events [44].
- **event_id** - The event id dict used to create a 'trial_type' column in events.tsv [44].
- **anonymize** - If None is provided (default) no anonymization is performed. If a dictionary is passed, data will be anonymized; identifying data structures such as study date and time will be changed [44].
- **overwrite** - Whether to overwrite existing files or data in files. Defaults to False. If overwrite is True, any existing files with the same BIDS parameters will be overwritten with the exception of the participants.tsv and scans.tsv files. For these files, parts of pre-existing data that match the current data will be replaced. If overwrite is False, no existing data will be overwritten or replaced [44].
- **verbose** - If verbose is True, this will print a snippet of the sidecar files. If False, no content will be printed [44].

B Converter user guide

A tool that converts data from the BrainVision format to BIDS is written in Python 3.8. The tool is stored on the github (https://github.com/lukas-45/Converter-BrainVision-to-BIDS). It is necessary to download repositories to the local storage of your computer from the given url address.

B.1 Download python

You must have python installed for the tool to work. Everything was tested on version 3.8. When installing, it is necessary to check the installation of pip. So I will describe how to install Python 3.8.

B.1.1 Download and install Python 3.8

You can download the python 3.8 from the official website[1]. I downloaded Windows x86 executable installer, but you can download this version for other platforms as well. After download you can double-click to the exe file. When the installation wizard starts, it is important to check the Add Python 3.8 to PATH option. You can see the installation window in Figure B.1. Then you will follow the guide.

B.2 Install tool

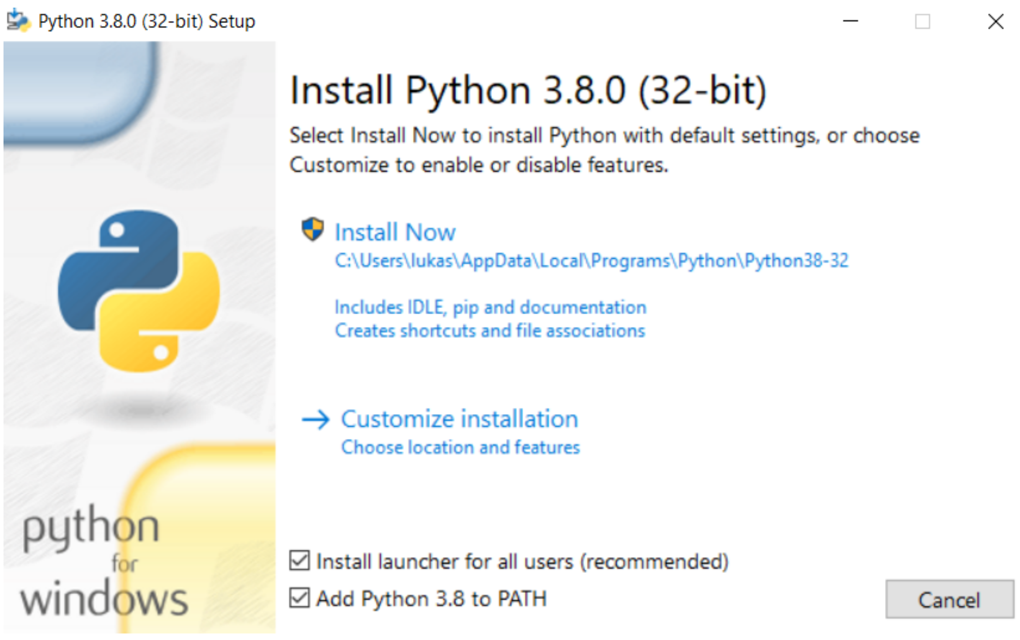
After install Python, you can download the repository to the local repository. It contains all the necessary files to run the tool. There is no need to install the tool. You only need to open command line in the directory path, where you have the conversion tool. You have to type the command which you can see in Listing B.1

```
pip install mne
```

Listing B.1: Install the necessary library to run conversion tool

After install mne, you can run the conversion tool.

[1] `https://www.python.org/downloads/release/python-380/`.


Python 3.8.0 (32-bit) Setup

Install Python 3.8.0 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\lukas\AppData\Local\Programs\Python\Python38-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

Figure B.1: A python installer window.

B.3 Run conversion tool

The conversion tool is started using the following command in the command line on the directory path, where you have the conversion tool. You have to type the command which you can see in Listing B.2.

```
python gui.py
```

Listing B.2: Run the conversion tool

After starting the tool, you will see the window which you can see in Figure B.2. We will now describe the individual parts of the window:

- Radio button(1) and radio button(5) are switches that determine whether to add an entire dataset or just add experiment to an existing dataset. If radio button(1) is active, then text field(2), button(3) and button(4) are active. If radio button(5) is active, then button(6) and button(7) are active.
- Number 2 - Here the user writes the name of the created dataset.
- Button and label(3) - Here the user selects the path to the directory of the original BrainVision dataset using the button.
- Button and label(4) - Here the user uses the button to select the path to the directory in which he wishes to save the newly created dataset.

- Button and label(6) - Here the user selects the path to the directory of the original BrainVision experiment using the button.
- Button and label(7) - Here the user uses the button to select the path to the directory of the BIDS dataset.
- Convert data button(8) - The Convert data button converts data from the BrainVision format to the BIDS format.
- Text area(9) - Here, the user will see all the experiments that are loaded from the directory specified in label(3) or label(6)
- Clear exp button(10) - Here the user uses the button to clear an experiment from the list.
- Clear all button(11) - Here the user uses the button to clear the list.
- Directory dataset path label(12) - Here the user can see the directory dataset path.

Figure B.3 shows the window after filling in the required data. You can see the window in Figure B.4 after pressing the Transfer data button. Notice that a pending window appears that shows how much data is already converted. Also note that all buttons are locked so that the user cannot change anything during the conversion. When the conversion is complete, the buttons will be available again.

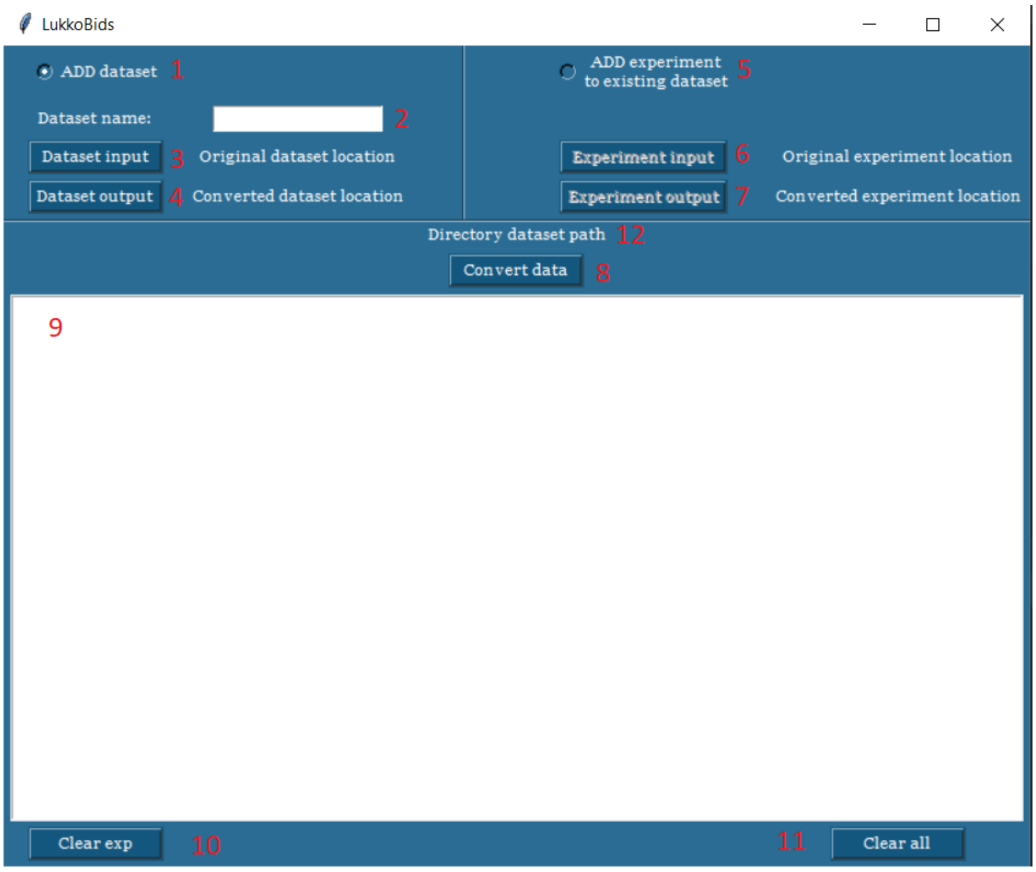
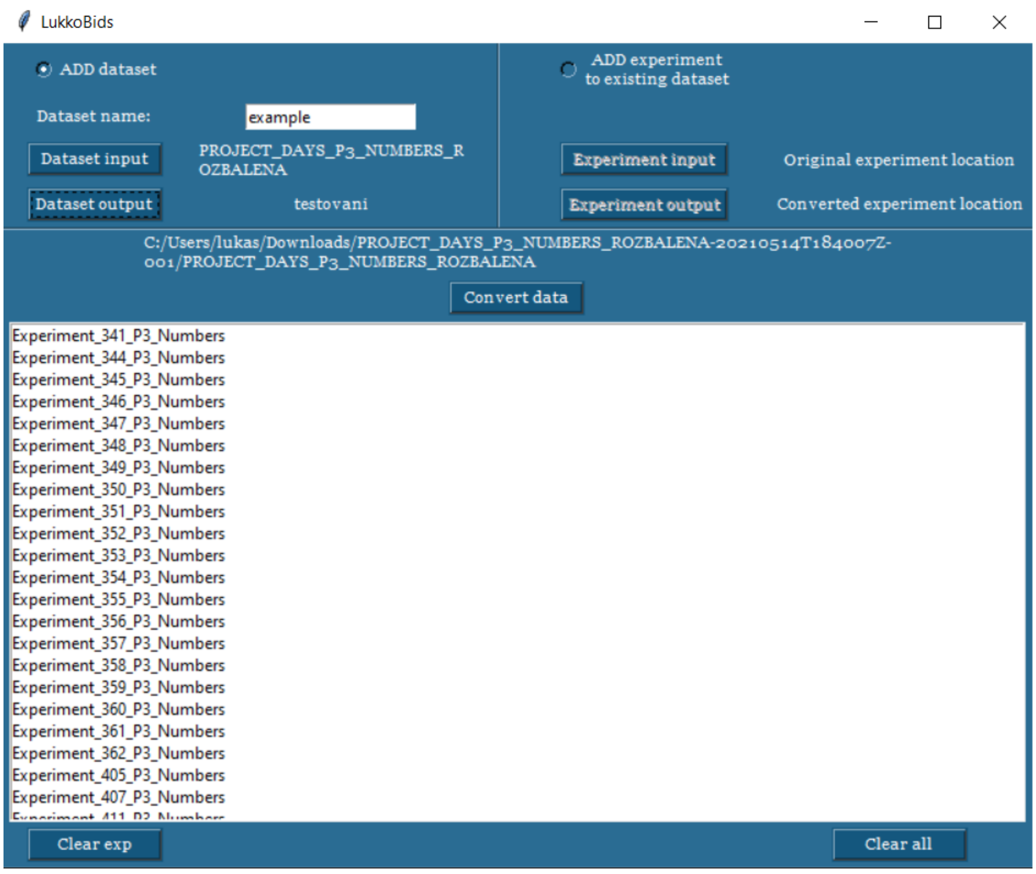

LukkoBids
ADD dataset 1
Dataset name: 2
Dataset input 3 Original dataset location
Dataset output 4 Converted dataset location
ADD experiment to existing dataset 5
Experiment input 6 Original experiment location
Experiment output 7 Converted experiment location
Directory dataset path 12
Convert data 8
9
Clear exp 10
11 Clear all

Figure B.2: A tool window after its launch.


LukkoBids

(●) ADD dataset

Dataset name: example

Dataset input PROJECT_DAYS_P3_NUMBERS_ROZBALENA

Dataset output testovani

( ) ADD experiment to existing dataset

Experiment input Original experiment location

Experiment output Converted experiment location

C:/Users/lukas/Downloads/PROJECT_DAYS_P3_NUMBERS_ROZBALENA-20210514T184007Z-001/PROJECT_DAYS_P3_NUMBERS_ROZBALENA

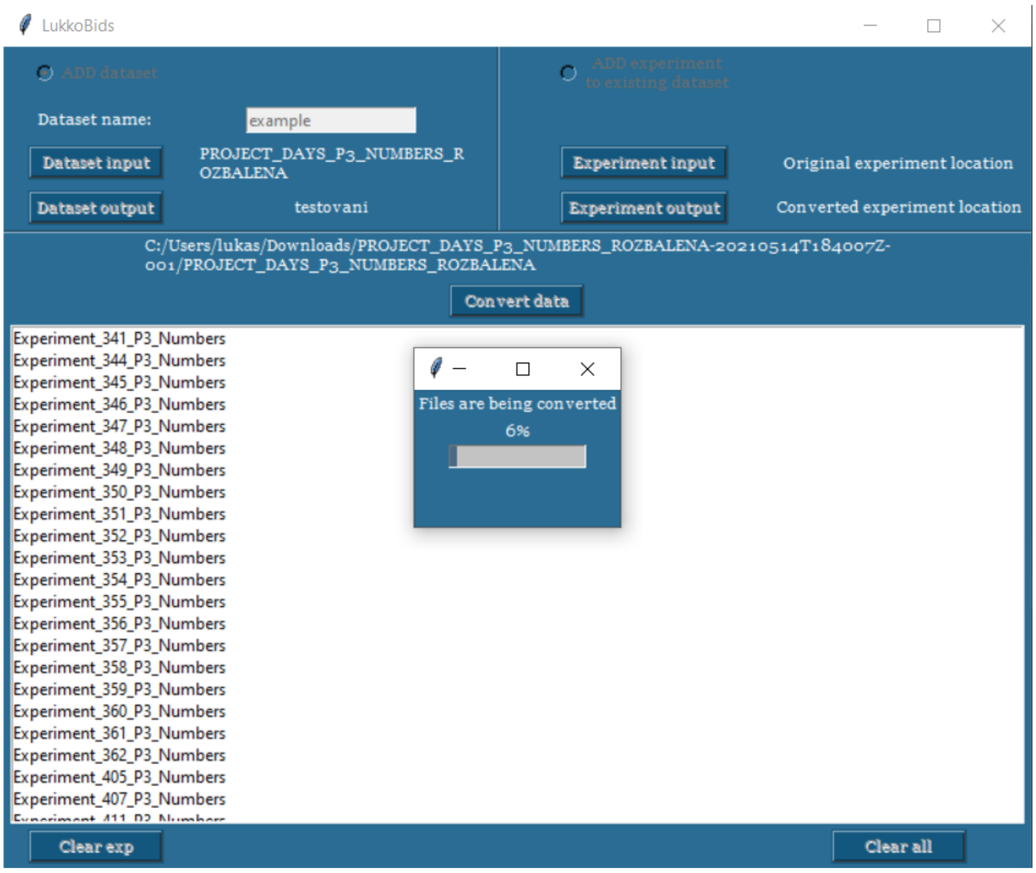
Convert data

```
Experiment_341_P3_Numbers
Experiment_344_P3_Numbers
Experiment_345_P3_Numbers
Experiment_346_P3_Numbers
Experiment_347_P3_Numbers
Experiment_348_P3_Numbers
Experiment_349_P3_Numbers
Experiment_350_P3_Numbers
Experiment_351_P3_Numbers
Experiment_352_P3_Numbers
Experiment_353_P3_Numbers
Experiment_354_P3_Numbers
Experiment_355_P3_Numbers
Experiment_356_P3_Numbers
Experiment_357_P3_Numbers
Experiment_358_P3_Numbers
Experiment_359_P3_Numbers
Experiment_360_P3_Numbers
Experiment_361_P3_Numbers
Experiment_362_P3_Numbers
Experiment_405_P3_Numbers
Experiment_407_P3_Numbers
Experiment_411_P3_Numbers
```

Clear exp

Clear all

Figure B.3: A tool window after filling in the required data.



Dataset name: example

PROJECT_DAYS_P3_NUMBERS_ROZBALENA

testovani

Original experiment location

Converted experiment location

C:/Users/lukas/Downloads/PROJECT_DAYS_P3_NUMBERS_ROZBALENA-20210514T184007Z-001/PROJECT_DAYS_P3_NUMBERS_ROZBALENA

```
Experiment_341_P3_Numbers
Experiment_344_P3_Numbers
Experiment_345_P3_Numbers
Experiment_346_P3_Numbers
Experiment_347_P3_Numbers
Experiment_348_P3_Numbers
Experiment_349_P3_Numbers
Experiment_350_P3_Numbers
Experiment_351_P3_Numbers
Experiment_352_P3_Numbers
Experiment_353_P3_Numbers
Experiment_354_P3_Numbers
Experiment_355_P3_Numbers
Experiment_356_P3_Numbers
Experiment_357_P3_Numbers
Experiment_358_P3_Numbers
Experiment_359_P3_Numbers
Experiment_360_P3_Numbers
Experiment_361_P3_Numbers
Experiment_362_P3_Numbers
Experiment_405_P3_Numbers
Experiment_407_P3_Numbers
Experiment_411_P3_Numbers
```

Files are being converted

6%

Figure B.4: A tool window after pressing the button Convert data.

C Use case

In this section I will describe the directory, data and metadata structure before and after the conversion.

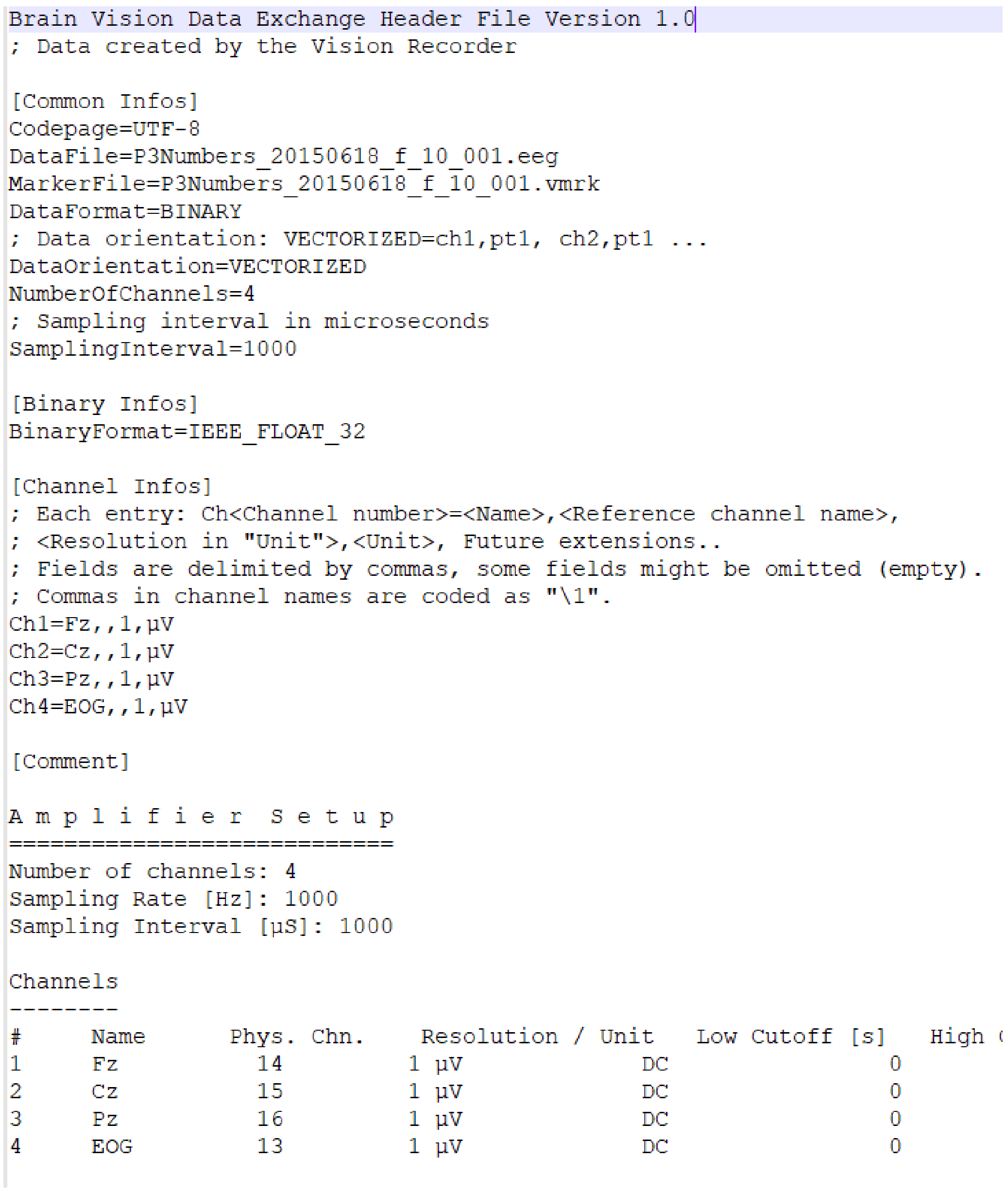
C.1 BrainVision

Before the conversion, the data and metadata were in BrainVision and odml standard. You can see the BrainVision directory structure in Figure 7.2. A very important file for the conversion is the file with the .vhdr extension. This file refers to two other files: .vmrk and .eeg. You can see the contents of the file with .vhdr extension in Figure C.1. The convert tool uses .txt file to fill information about participants to participants.tsv. You can see the contents of the file with .txt extension in Figure C.3. The metadata.xml file follows the odML structure. You can see an example of the structure of metadata.xml in Figure C.2. This file contains all the important information about the experiment.

C.2 BIDS

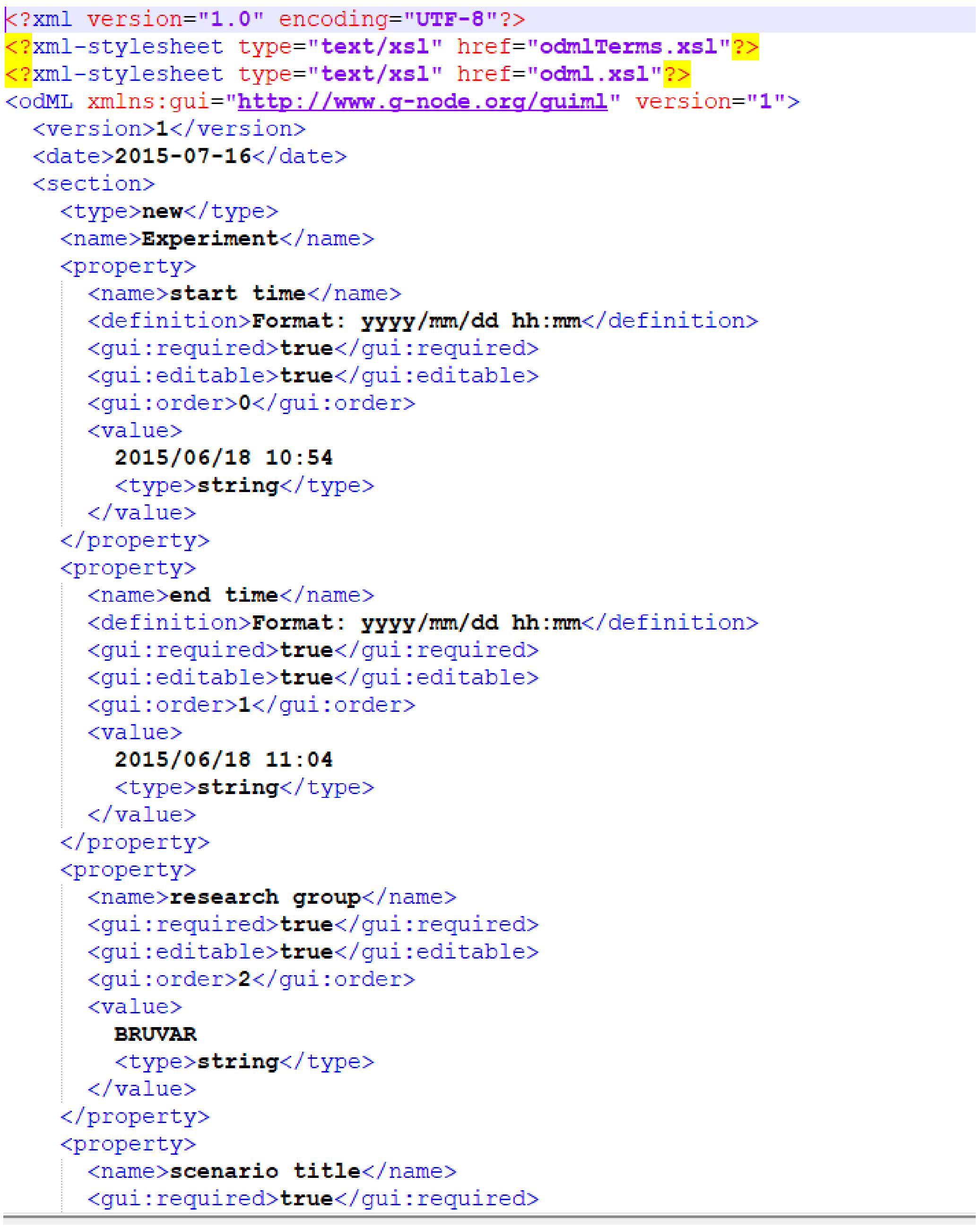
You can see the BIDS directory structure after conversion in Figure 7.7. The individual files after conversion look as follows:

- dataset_description.json in Figure C.4
- participants.json in Figure C.5
- participants.tsv in Figure C.6
- sub-*_scans.tsv in Figure C.7
- sub-*_task-*_events.tsv in Figure C.8
- sub-*_task-*_channels.tsv in Figure C.9
- sub-*_task-*_eeg.json in Figure C.10



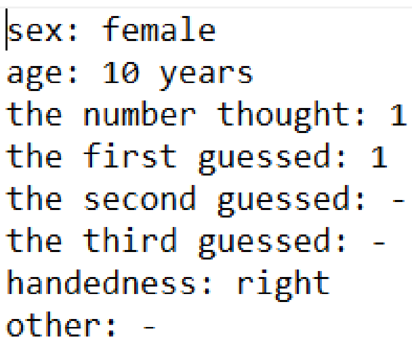
```
Brain Vision Data Exchange Header File Version 1.0
; Data created by the Vision Recorder

[Common Infos]
Codepage=UTF-8
DataFile=P3Numbers_20150618_f_10_001.eeg
MarkerFile=P3Numbers_20150618_f_10_001.vmrk
DataFormat=BINARY
; Data orientation: VECTORIZED=ch1,pt1, ch2,pt1 ...
DataOrientation=VECTORIZED
NumberOfChannels=4
; Sampling interval in microseconds
SamplingInterval=1000

[Binary Infos]
BinaryFormat=IEEE_FLOAT_32

[Channel Infos]
; Each entry: Ch<Channel number>=<Name>,<Reference channel name>,
; <Resolution in "Unit">,<Unit>, Future extensions..
; Fields are delimited by commas, some fields might be omitted (empty).
; Commas in channel names are coded as "\1".
Ch1=Fz,,1,µV
Ch2=Cz,,1,µV
Ch3=Pz,,1,µV
Ch4=EOG,,1,µV

[Comment]

A m p l i f i e r  S e t u p
============================
Number of channels: 4
Sampling Rate [Hz]: 1000
Sampling Interval [µS]: 1000

Channels
--------
#     Name      Phys. Chn.    Resolution / Unit   Low Cutoff [s]   High
1     Fz          14         1 µV             DC                0
2     Cz          15         1 µV             DC                0
3     Pz          16         1 µV             DC                0
4     EOG         13         1 µV             DC                0
```

Figure C.1: Content of .vhdr file. The file contains metadata.



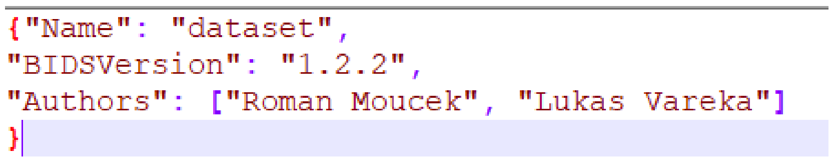
```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="odmlTerms.xsl"?>
<?xml-stylesheet type="text/xsl" href="odml.xsl"?>
<odML xmlns:gui="http://www.g-node.org/guiml" version="1">
  <version>1</version>
  <date>2015-07-16</date>
  <section>
    <type>new</type>
    <name>Experiment</name>
    <property>
      <name>start time</name>
      <definition>Format: yyyy/mm/dd hh:mm</definition>
      <gui:required>true</gui:required>
      <gui:editable>true</gui:editable>
      <gui:order>0</gui:order>
      <value>
        2015/06/18 10:54
        <type>string</type>
      </value>
    </property>
    <property>
      <name>end time</name>
      <definition>Format: yyyy/mm/dd hh:mm</definition>
      <gui:required>true</gui:required>
      <gui:editable>true</gui:editable>
      <gui:order>1</gui:order>
      <value>
        2015/06/18 11:04
        <type>string</type>
      </value>
    </property>
    <property>
      <name>research group</name>
      <gui:required>true</gui:required>
      <gui:editable>true</gui:editable>
      <gui:order>2</gui:order>
      <value>
        BRUVAR
        <type>string</type>
      </value>
    </property>
    <property>
      <name>scenario title</name>
      <gui:required>true</gui:required>
```

Figure C.2: Content of metadata.xml. The file contains metadata in odML format.



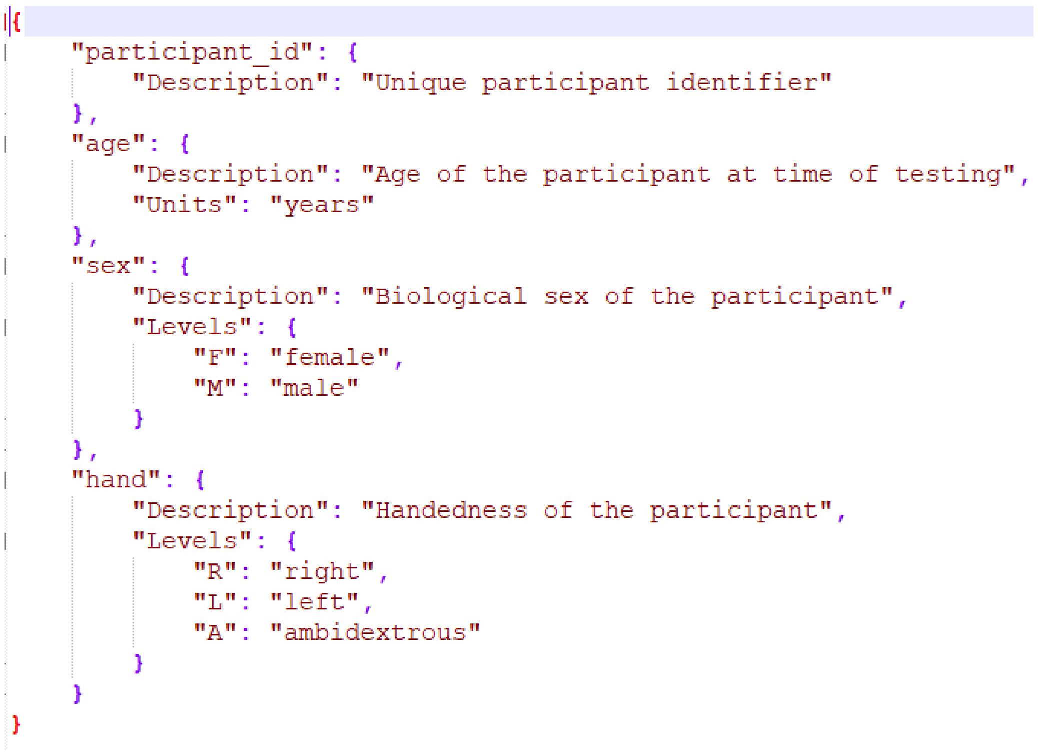
```
sex: female
age: 10 years
the number thought: 1
the first guessed: 1
the second guessed: -
the third guessed: -
handedness: right
other: -
```

Figure C.3: Content of .txt file. The file contains information about participants.



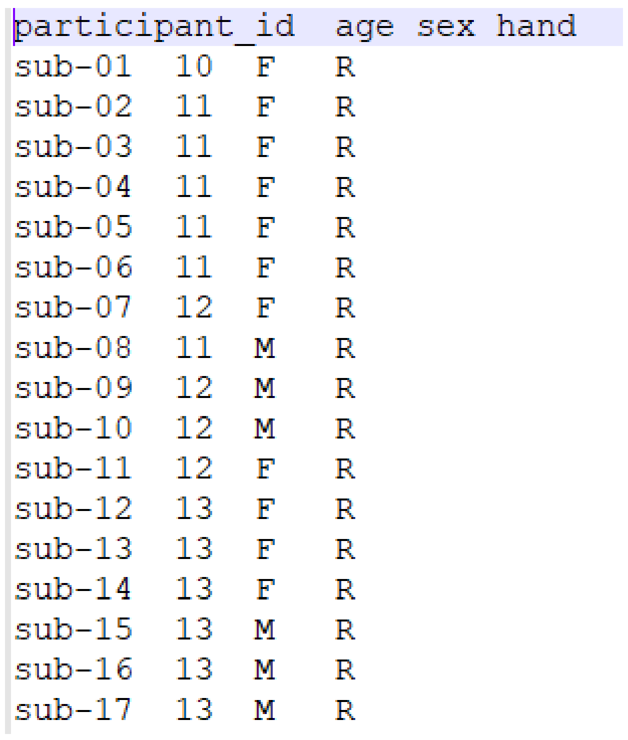
```
{"Name": "dataset",
"BIDSVersion": "1.2.2",
"Authors": ["Roman Moucek", "Lukas Vareka"]
}
```

Figure C.4: Content of dataset_description file. The file contains information about dataset (Name, version and authors).



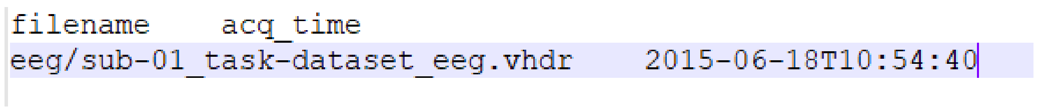
```
{
    "participant_id": {
        "Description": "Unique participant identifier"
    },
    "age": {
        "Description": "Age of the participant at time of testing",
        "Units": "years"
    },
    "sex": {
        "Description": "Biological sex of the participant",
        "Levels": {
            "F": "female",
            "M": "male"
        }
    },
    "hand": {
        "Description": "Handedness of the participant",
        "Levels": {
            "R": "right",
            "L": "left",
            "A": "ambidextrous"
        }
    }
}
```

Figure C.5: Content of participants.json file. The file contains the structure of the Participants.tsv file.



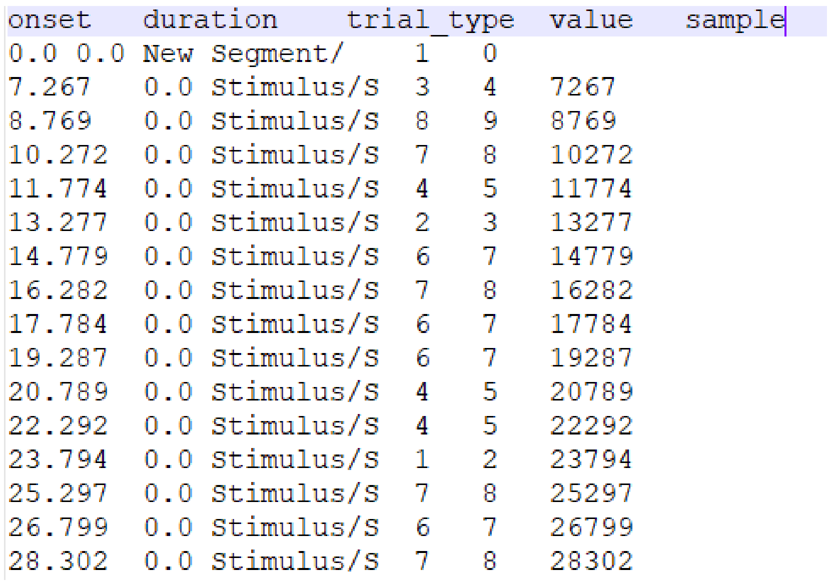
```
participant_id	age	sex	hand
sub-01	10	F	R
sub-02	11	F	R
sub-03	11	F	R
sub-04	11	F	R
sub-05	11	F	R
sub-06	11	F	R
sub-07	12	F	R
sub-08	11	M	R
sub-09	12	M	R
sub-10	12	M	R
sub-11	12	F	R
sub-12	13	F	R
sub-13	13	F	R
sub-14	13	F	R
sub-15	13	M	R
sub-16	13	M	R
sub-17	13	M	R
```

Figure C.6: Content of participants.tsv file. The file contains information about participants. It means age, gender and handedness



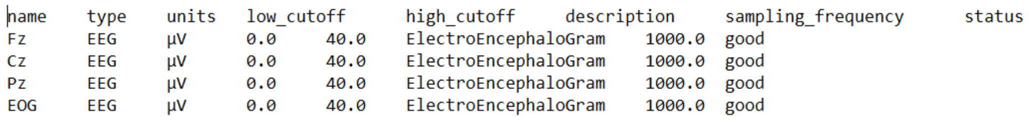
```
filename	acq_time
eeg/sub-01_task-dataset_eeg.vhdr	2015-06-18T10:54:40
```

Figure C.7: Content of scans.tsv file.



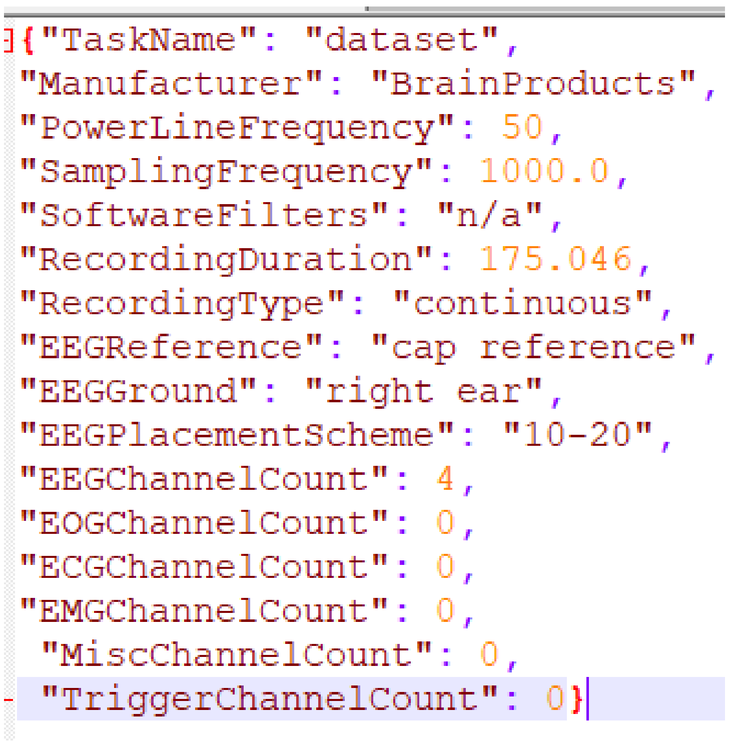
```
onset	duration	trial_type	value	sample
0.0	0.0	New Segment/	1	0
7.267	0.0	Stimulus/S	3	4	7267
8.769	0.0	Stimulus/S	8	9	8769
10.272	0.0	Stimulus/S	7	8	10272
11.774	0.0	Stimulus/S	4	5	11774
13.277	0.0	Stimulus/S	2	3	13277
14.779	0.0	Stimulus/S	6	7	14779
16.282	0.0	Stimulus/S	7	8	16282
17.784	0.0	Stimulus/S	6	7	17784
19.287	0.0	Stimulus/S	6	7	19287
20.789	0.0	Stimulus/S	4	5	20789
22.292	0.0	Stimulus/S	4	5	22292
23.794	0.0	Stimulus/S	1	2	23794
25.297	0.0	Stimulus/S	7	8	25297
26.799	0.0	Stimulus/S	6	7	26799
28.302	0.0	Stimulus/S	7	8	28302
```

Figure C.8: Content of events.tsv file.



```
name	type	units	low_cutoff	high_cutoff	description	sampling_frequency	status
Fz	EEG	µV	0.0	40.0	ElectroEncephaloGram	1000.0	good
Cz	EEG	µV	0.0	40.0	ElectroEncephaloGram	1000.0	good
Pz	EEG	µV	0.0	40.0	ElectroEncephaloGram	1000.0	good
EOG	EEG	µV	0.0	40.0	ElectroEncephaloGram	1000.0	good
```

Figure C.9: Content of channels.tsv file.



```
{"TaskName": "dataset",
"Manufacturer": "BrainProducts",
"PowerLineFrequency": 50,
"SamplingFrequency": 1000.0,
"SoftwareFilters": "n/a",
"RecordingDuration": 175.046,
"RecordingType": "continuous",
"EEGReference": "cap reference",
"EEGGround": "right ear",
"EEGPlacementScheme": "10-20",
"EEGChannelCount": 4,
"EOGChannelCount": 0,
"ECGChannelCount": 0,
"EMGChannelCount": 0,
 "MiscChannelCount": 0,
 "TriggerChannelCount": 0}
```

Figure C.10: Content of *eeg.json file.

D Figures

Some of Figures that are referenced in the text are too large, so I decided to attach them to appendices.

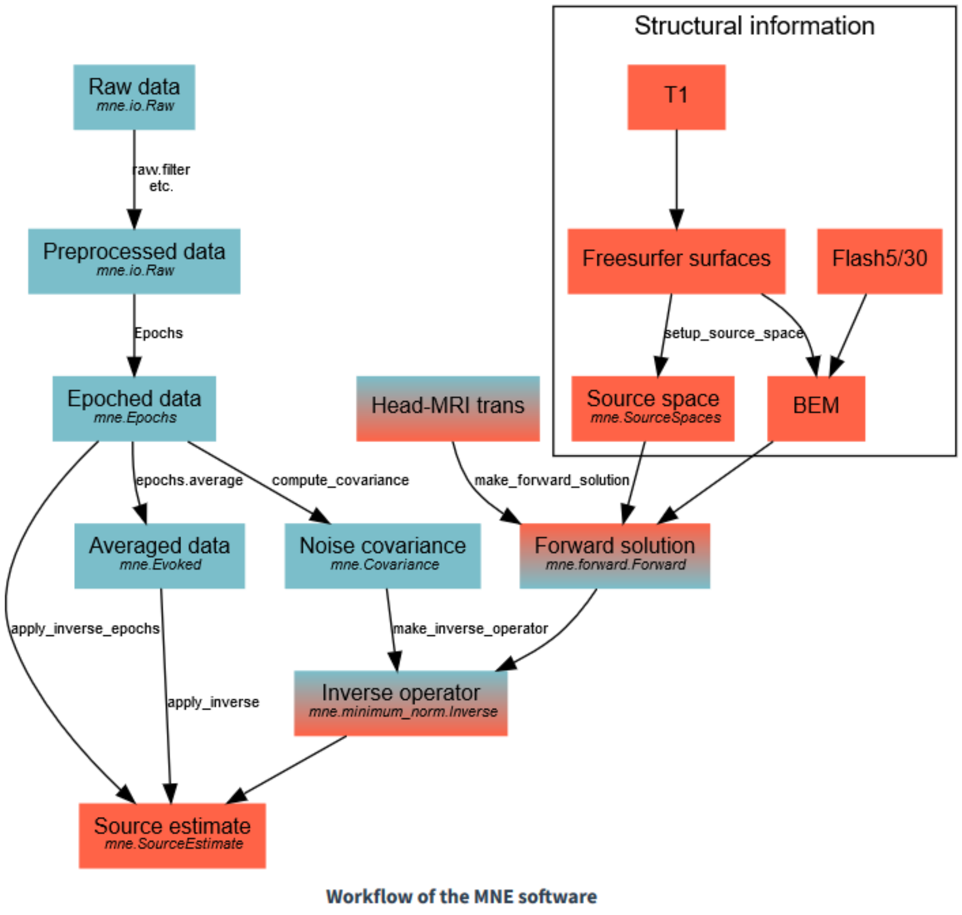
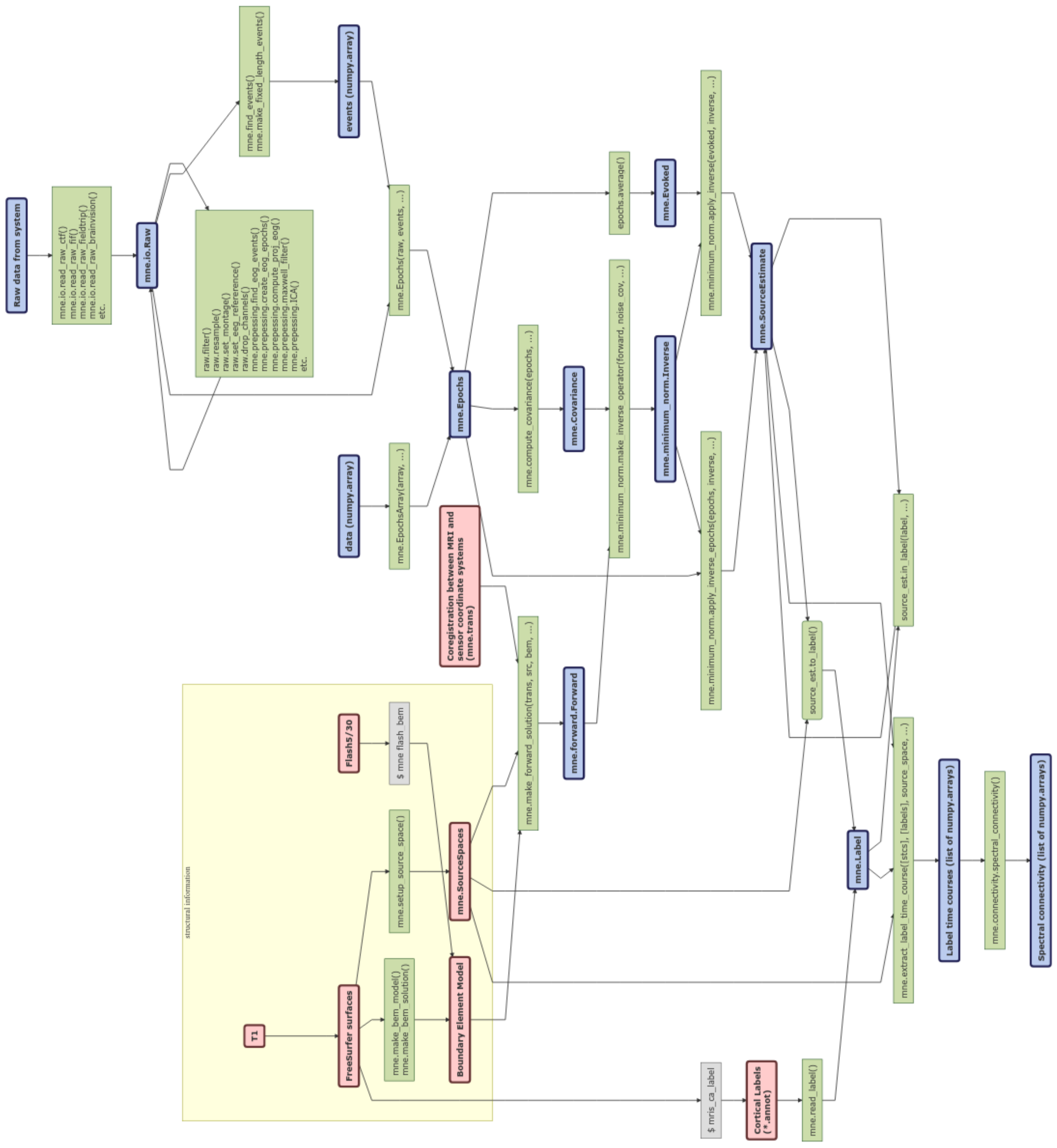


Figure D.1: Workflow of the MNE software. [45]



Raw data from system
mne.io.read_raw_ctf()
mne.io.read_raw_fif()
mne.io.read_raw_fieldtrip()
mne.io.read_raw_brainvision()
etc.
mne.io.Raw
raw.filter()
raw.resample()
raw.set_montage()
raw.set_eeg_referenerce()
raw.drop_channels()
mne.prepessing.find_eog_events()
mne.prepessing.create_eog_epochs()
mne.prepessing.compute_proj_eog()
mne.prepessing.maxwell_filter()
mne.prepessing.ICA()
etc.
mne.find_events()
mne.make_fixed_length_events()
events (numpy.array)
mne.Epochs(raw, events, ...)
data (numpy.array)
mne.EpochsArray(array, ...)
mne.Epochs
epochs.average()
mne.Evoked
mne.compute_covariance(epochs, ...)
mne.Covariance
mne.minimum_norm.make_inverse_operator(forward, noise_cov, ...)
mne.minimum_norm.Inverse
mne.minimum_norm.apply_inverse(evoked, inverse, ...)
mne.minimum_norm.apply_inverse_epochs(epochs, inverse, ...)
mne.SourceEstimate
Coregistration between MRI and sensor coordinate systems (mne.trans)
mne.make_forward_solution(trans, src, bem, ...)
mne.forward.Forward
structural information
T1
FreeSurfer surfaces
Flash5/30
$ mne flash_bem
mne.setup_source_space()
mne.SourceSpaces
mne.make_bem_model()
mne.make_bem_solution()
Boundary Element Model
$ mris_ca_label
Cortical Labels (*.annot)
mne.read_label()
source_est.to_label()
mne.Label
source_est.in_label(label, ...)
mne.extract_label_time_course([stcs], [labels], source_space, ...)
Label time courses (list of numpy.arrays)
mne.connectivity.spectral_connectivity()
Spectral connectivity (list of numpy.arrays)

Figure D.2: Data model of MNE

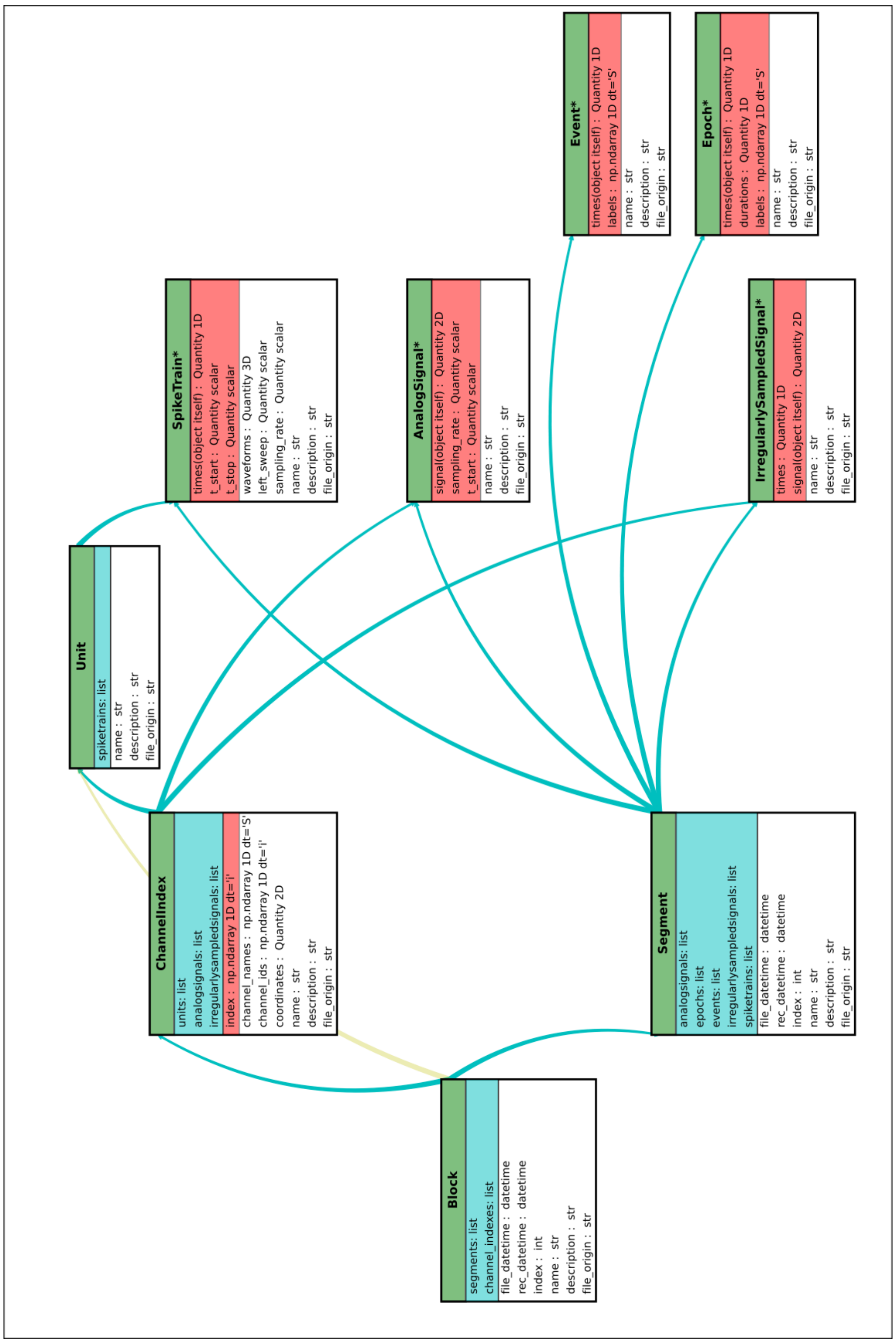


Figure D.3: Neo diagram. [46]