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

Master's thesis

Data storage for electrophysiological experiments

Pilsen 2020 Jan Šedivý

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 2020

Jan Šedivý

Abstract

This master's thesis describes the current solution of storing the neuroinformatics laboratory experiments of the University of West Bohemia. An analysis of available data standards for storing neurophysiological experiments is done where each data standard is evaluated on the following of the FAIR principles of storing open science data. The most appropriate standard is selected based on the FAIR principles assessment and the complexity of potential conversion. A conversion tool is implemented to convert the current experiments to the new data standard. Based on the selected data standard, a suitable repository is chosen to store the converted experiment datasets and replace the current solution. A FAIR principles assessment is done on the resulted dataset uploaded in the repository.

Abstrakt

Tato práce popisuje současné řešení ukládání elektrofyziologických experimentů neuroinformatické laboratoře na Západočeské univerzitě. Jsou zkoumány dostupné datové standardy vhodné k ukládání podobných experimentů. Každý z těchto standardů je ohodnocen na základě splňování tzv. FAIR principů pro ukládání otevřených vědeckých dat. Z tohoto ohodnocení a posouzení náročnosti případné konverze stávajících dat je vybrán nejvhodnější standard. Následně je implementován nástroj na konverzi stávajících dat do vybraného standardu. V návaznosti ke zvolenému standardu je vybráno a zprovozněno i vhodné datové úložiště, do kterého budou nová data uložena. Na výsledném řešení je provedeno vyhodnocení splnění jednotlivých FAIR principů.

Contents

1 Introduction

The variety of open scientific data and their formats is continually expanding. For maintaining the ability to manipulate, analyze, and share these expanding data, a multitude of different recommendations and principles are being introduced. These principles offer guidelines to define these data and their metadata, to make them more findable, user-readable, and machine-operable.

The neuroinformatics laboratory at the Faculty of Applied Sciences, University of West Bohemia, performs different types of electrophysiological experiments. The locally maintained EEGbase portal is currently used to store the resulting datasets of these experiments. However, the portal is too complicated and becoming outdated in terms of technology and in terms of compliance with the open data principles as well.

This master's thesis focuses on the FAIR principles and describes their proposed guidelines to achieve the dataset's findability, accessibility, interoperability, and readability. The current data model to store the datasets combines different technologies that do not comply well with the principles. Multiple data standards suitable for storing neurophysiological experiments are described. The most suitable data standard is selected upon the assessment of the FAIR principles. However, the difficulty of the follow-up conversion must be taken into consideration as well. Based on the selection of the new data model, an appropriate data repository is chosen. This repository is used as the storage for the newly converted datasets and should act as a supporting role in the satisfaction of the FAIR principles.

A Python tool for converting the original EEGbase data to the selected data standard is implemented. Although the primary focus is put into converting the EEGbase data and its structure, the design has been made more general to allow for broader use of the tool and conversion of differently structured metadata. The goal of the conversion is to lead to a FAIR dataset without the loss of any crucial information from the original dataset.

The implemented tool is tested on a set of selected datasets from the EEGbase portal, and the converted datasets are shared into the new repository. The final result is evaluated for compliance with the FAIR principles. The ideal outcome of this thesis is the proposed and implemented solution replacing the current EEGbase portal and its data model completely.

2 FAIR principles

There are several guidelines and principles available to follow while storing, maintaining and disclosing open scientific data. One of these guidelines are so-called FAIR principles. This chapter describes these principles, including examples of correct ways of fulfilling them.

The main goal of FAIR principles is to make data more machine-friendly. That is, to make computational systems more capable of finding, accessing, interoperating and reusing data, without the need of any or minimal human intervention. These attributes are becoming more significant as the reliance on computational support for humans dealing with data grows with the increasing volume, creation speed and complexity of given data [1].

FAIR principles include four main categories described below, each category is separated into multiple steps that the data have to meet. They are brief, domain-independent, high-level principles that can be applied to a wide range of outputs. Some of the principles may include underlying points that further specify the main cause (i.e. the A1 principle contains the further specifying points A1.1 and A1.2). The principles refer to three types of entities: data (or any digital object), metadata (information about that digital object), and resource (the infrastructure component, e.g., a repository). Throughout the principles, a phrase '(meta)data' is used in cases where the principle should be applied to both metadata and data [1, 2].

Findable

The first step in using data is being able to find them. This includes assigning a persistent identifier which ensures that metadata and data are easy to find for both humans and computers. Machine-readable metadata are essential for automatic discovery of datasets and services [1, 3].

F1. (meta)data are assigned a globally unique and persistent identifier

F2. data are described with rich metadata (defined by R1 below)

F3. metadata clearly and explicitly include the identifier of the data it describes

F4. (meta)data are registered or indexed in a searchable resource

Accessible

Once the required data are found, the next step is to know how they can be accessed. As the data do not necessarily have to be open, this step may also include authentication and authorization [1, 3].

A1. (meta)data are retrievable by their identifier using a standardized communications protocol

  A1.1 the protocol is open, free, and universally implementable

  A1.2 the protocol allows for an authentication and authorization procedure, where necessary

A2. metadata are accessible, even when the data are no longer available

Interoperable

The data usually need to be integrated with other data. In addition, the data need to interoperate with applications or workflows for analysis, storage, and processing. For that, the data should use community agreed formats, languages and vocabularies. The metadata should also use community agreed standards and vocabularies, and contain links to related information using identifiers [1, 3].

I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.

I2. (meta)data use vocabularies that follow FAIR principles

I3. (meta)data include qualified references to other (meta)data

Reusable

Optimizing the reusability of data is the preeminent goal of the FAIR principles. To achieve reusability data and its metadata should be well-described so that they can later be replicated and combined in different settings [1, 3].

R1. meta(data) are richly described with a plurality of accurate and relevant attributes

  R1.1. (meta)data are released with a clear and accessible data usage license

  R1.2. (meta)data are associated with detailed provenance

  R1.3. (meta)data meet domain-relevant community standards

2.1 Applying the principles

The FAIR principles in themselves do not determine the exact form and shape of the data. They are more of a set of recommendations to follow when maintaining open data, independent of the domain of the data. The GO FAIR is an initiative whose goal is the promotion of these principles in open science, e.g., the Internet of FAIR Data & Services (IFDS)[4] or European Open Science Cloud (EOSC)[5]. It achieves this by offering an open ecosystem for individuals, institutions, and organizations to work together through different Implementation Networks. Each network deals with different levels of FAIR (technological, skill development, or policies and incentives) [6].

The GO FAIR initiative also suggests a process of applying FAIR principles to an already existing, non-FAIR data. The steps of this process are interpreted in Figure 2.1. Upon retrieving the non-FAIR data (1), we need to analyze (2) its structure, relations between elements and so on. Then, a semantic model needs to be defined (3), describing the meaning of entities and relations in the dataset, i.e., ontologies or vocabularies, followed by transformation of the data onto the model (4). The transformation of the data may not suit some non-structured data like image pixels, audio, etc. After defining the metadata (6) and including the license information (5), the dataset is ready to be deployed to the data resource (7) where it can be indexed and accessed along with other FAIR data [7].

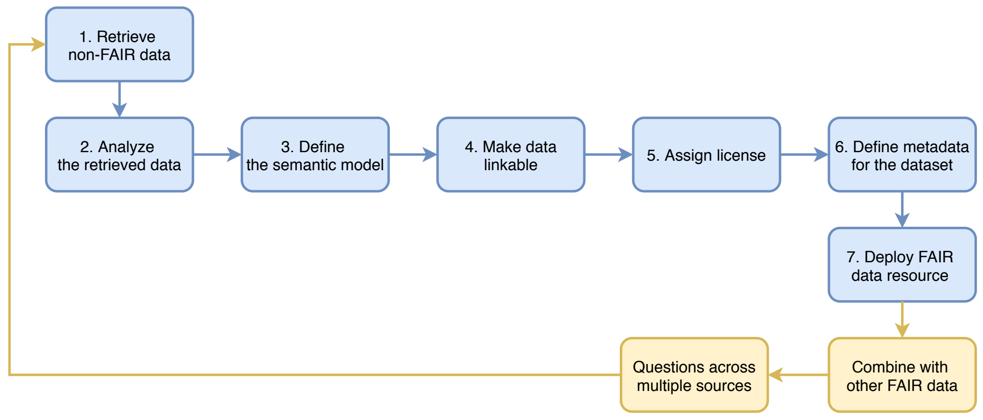

1. Retrieve non-FAIR data
2. Analyze the retrieved data
3. Define the semantic model
4. Make data linkable
5. Assign license
6. Define metadata for the dataset
7. Deploy FAIR data resource
Combine with other FAIR data
Questions across multiple sources

Figure 2.1: The FAIRification process suggested by GO FAIR [7].

3 EEGbase

The EEGbase portal is a web-based system developed and maintained by the Department of Computer Science and Engineering, University of West Bohemia, with the initial concepts dating back to the early 2010s. The system allows researchers to upload, download and manage EEG/ERP experiments consisted of data, metadata and experimental scenarios. The features of the system, among others, include sharing of knowledge, working in research groups, managing scientific discussions or running methods for signal processing [8].

3.1 Architecture

The EEGbase portal represents a central point of the complete infrastructure which provides a web-based interface for use by human users. Services provided by this portal include long-term, sustainable storing of data and metadata collected from experiments, various workflows for data processing and its sharing.

Besides this classical interface, intended for human users, a collection of web services is implemented to access the data and metadata by external software tools. These tools can be split into two separate groups. The first group includes tools, which are implemented as standalone libraries directly integrated within the EEGbase portal. One of these tools is the Semantic Framework, which aims to provide the experimental metadata in the semantic web languages and technologies, such as RDF or OWL. Data expressed in these languages and technologies are readable by semantic reasoners. The second group consists of standalone tools that can run and manage the EEGbase data locally on a user's computer. An example of these tools is JERPA, a desktop system for running signal processing methods and signal visualization, mobile version of the portal, tools for signal visualization or different analytical and statistical tools. The interfaces are implemented using RESTfull and SOAP web services [8, 9].

A considerable part of the infrastructure is created by several third-party hardware devices and software tools. These devices and tools are controlled by the experimenter who interacts with the EEG/ERP Portal [8]. The overall architecture of the EEGbase portal can be seen in Figure 3.1.

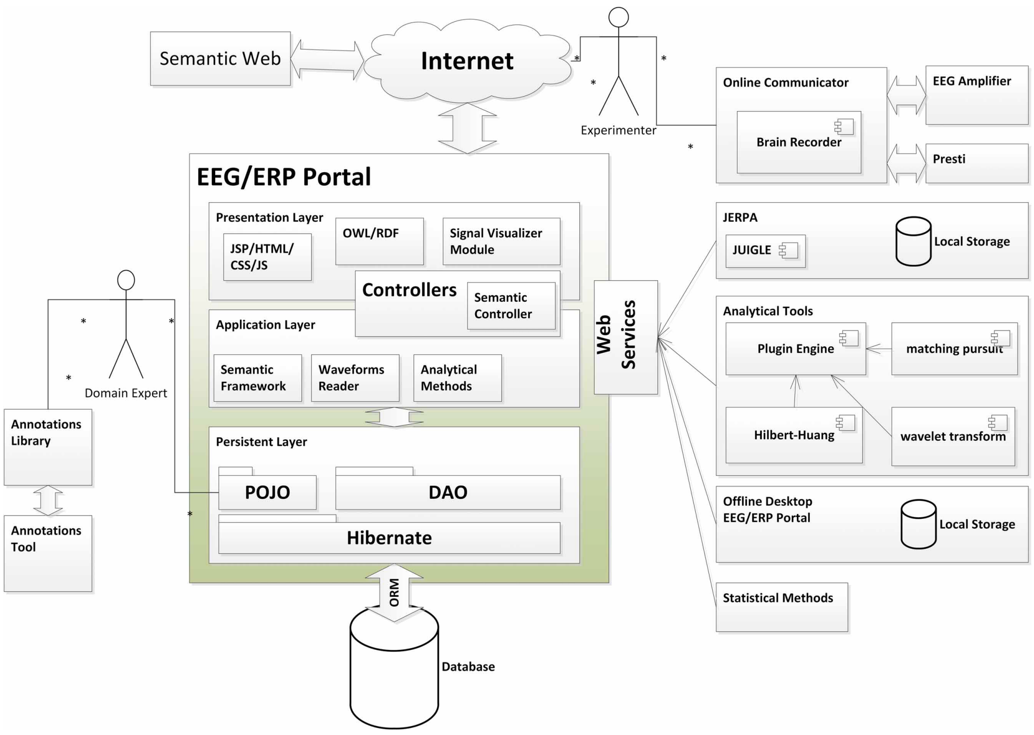


Figure 3.1: Overall architecture of the EEGbase portal [9].

The core of the EEGbase portal is implemented using Java SE 6 and the Spring framework in version 3.1.4. The persistent layer uses the object-relational mapping tool Hibernate to access data and metadata from the database. The current implementation separates the storage of relational and non-relational data. The relational data, containing mainly data used by the EEGbase web interface, i.e. user information, basic experiments information or scenarios, are stored using the PostgreSQL[10] database in version v9.2. The experiment data and metadata are stored as non-relational using the Elasticsearch[11] NoSQL database [8, 9].

3.2 Data formats

The data of an experiment are recorded from an EEG cap using the BrainVision Recorder software, collecting the data of an experiment and the devices metadata into the BrainVision format. This format consists of three files:

- A binary data file (.eeg) containing the voltage values of the EEG.
- A text header file (.vhdr) containing metadata describing the raw EEG data stored in the corresponding .eeg file.

```
[Common Infos]
DataFile=P3Numbers_20150414_m_13_003.eeg
MarkerFile=P3Numbers_20150414_m_13_003.vmrk
DataFormat=BINARY
NumberOfChannels=4
; Sampling interval in microseconds
SamplingInterval=1000
```

Listing 3.1: Exemplary header file [13].

- A text marker file (.vmrk) containing stimuli markers used in the experiment.

The header and marker files store information in the key-value form, with the possibility of separating them into sections represented by section name inside square brackets [12, 13]. An example of what a header file can look like is in listing 3.1. In this file, the `DataFile` and `MarkerFile` keys specify the corresponding .eeg and .vmrk files of the experiment and other key-value pairs like `DataFormat` or `SamplingInterval` provide additional metadata about the recording [13].

Listing 3.2 is an exemplary marker file. The key `DataFile` references the binary data file with recorded data. Each key in the "Marker infos" section represents a position in data points where the marker occurred, with some additional info, i.e. marker "Mk2" is of type "Stimulus", which occurred at position 7501 in recorded data.

```
[Common Infos]
DataFile=P3Numbers_20150414_m_13_003.eeg

[Marker Infos]
Mk2=Stimulus,S 6,7501,0,0
Mk3=Stimulus,S 6,9021,0,0
Mk4=Stimulus,S 4,10538,0,0
```

Listing 3.2: Exemplary marker file [13].

The additional experiment metadata are represented in a format called Open Metadata Markup Language (odML), a flexible and unified metadata format for data annotation used in neurophysiology [14]. These metadata are linked to the related data of the experiment. An example describing the start time of an experiment is shown in listing 3.3.

In addition to these data and metadata formats, different ontologies are used to help data sharing. One of these ontologies is the NEMO ontology. This ontology provides formal semantic definitions of concepts in ERP re-

```
<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>
        2015/04/14 10:01
        <type>string</type>
      </value>
    </property>
  </section>
</odML>
```

Listing 3.3: Exemplary experiment metadata in odML format [13].

search, including ERP patterns, spatial, temporal, functional (cognitive/behavioural) attributes of these patterns, data acquisition and analysis methods. Another used ontology is the OBI ontology for biological and clinical investigation description. Its terminology contains domain-specific terms and universal terms for general biological and technical usage. This ontology represents the design of an investigation, the protocols and instrumentation used, the material used, the data generated and the type analysis performed on it [8].

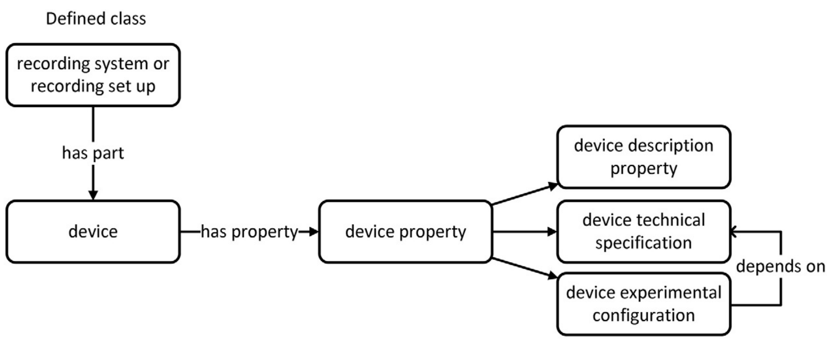
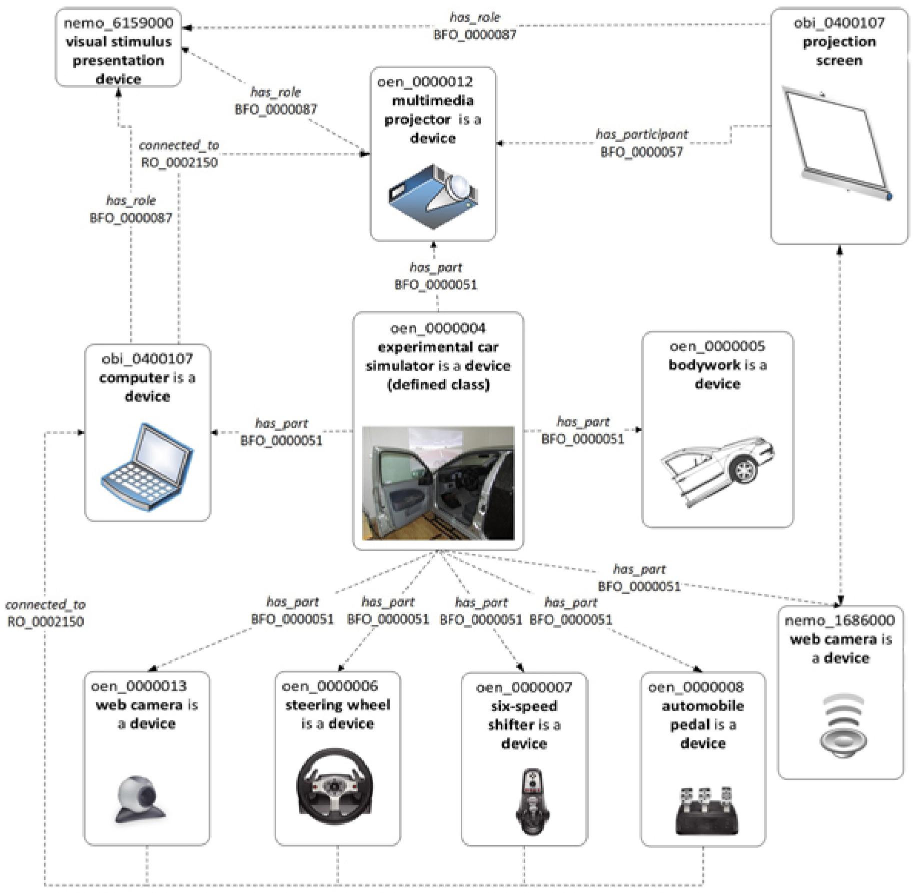

Defined class
recording system or recording set up
has part
device
has property
device property
device description property
device technical specification
device experimental configuration
depends on

Figure 3.2: Knowledge model of the OEN device branch [15].

These ontologies alone are not able to fully describe the information stored in the EEGbase portal, namely the set of the domain terms. That is why an additional Ontology for describing Experimental Neurophysiology (OEN) is being used. This ontology is separated into two branches for experimental data and metadata, each dealing with structured terminology to annotate them. It is used as an extension of OBI for devices and related information. The model of this annotation is shown in Figure 3.2 for a 'device branch' from the experiments [8].

Figure 3.3 shows the shortened description of an exemplary experimental set-up using terms (labels/names in bold) from OBI, NEMO, and OEN (all distinguished by a prefix of the id) and their relations (dashed arrow, label, and id). The full description is in Appendix C.

All of these technologies combined make up the complete structure of an experiment that the EEGbase portal can store. Figure 3.4 shows an overview of the structure. Full schema is in Appendix D. A `results` text file



nemo_6159000 visual stimulus presentation device
has_role BFO_0000087
obi_0400107 projection screen
has_role BFO_0000087
oen_0000012 multimedia projector is a device
has_participant BFO_0000057
connected_to RO_0002150
has_role BFO_0000087
has_part BFO_0000051
oen_0000004 experimental car simulator is a device (defined class)
obi_0400107 computer is a device
oen_0000005 bodywork is a device
has_part BFO_0000051
has_part BFO_0000051
has_part BFO_0000051
connected_to RO_0002150
has_part BFO_0000051
has_part BFO_0000051
has_part BFO_0000051
has_part BFO_0000051
nemo_1686000 web camera is a device
oen_0000013 web camera is a device
oen_0000006 steering wheel is a device
oen_0000007 six-speed shifter is a device
oen_0000008 automobile pedal is a device

Figure 3.3: Description of experimental set-up in EEGbase [8].

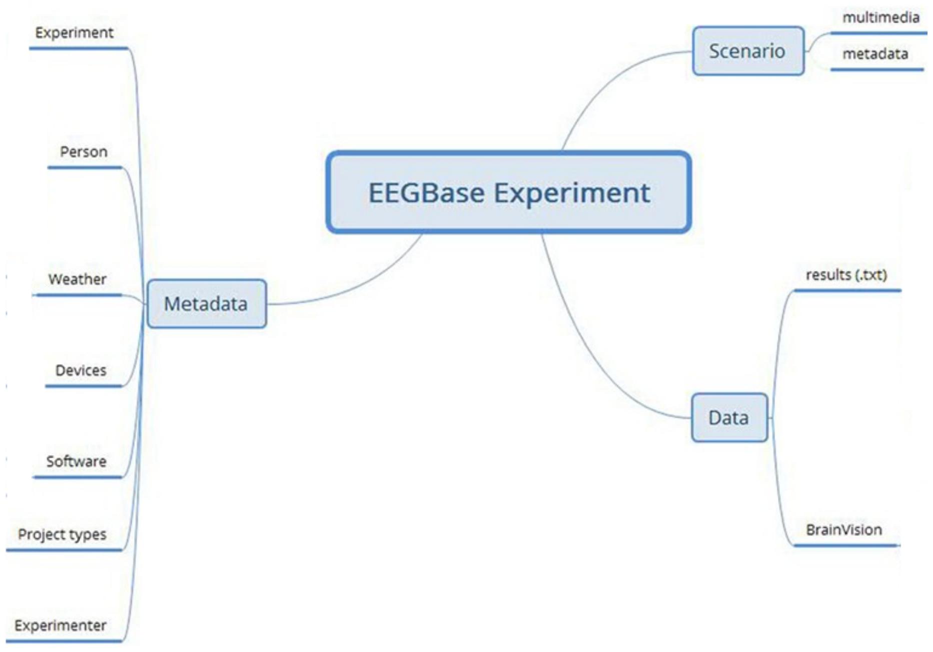


Figure 3.4: The structure of an EEGbase experiment dataset [16].

can accompany data in some cases. This file is used with some of the event-related potential experiments and contains event summaries and averages. Scenario information is stored separately from the data and metadata. These may include camera records and its metadata or other multimedia used during the experiment [16].

3.2.1 Feasible experiments

A total of seven different types of experiments from the EEGBase portal were provided for the conversion to a new data standard. The names and the dataset structure of the experiments are as follows:

- Developmental coordination disorder in children - experimental work and data annotation: Datasets from the [17] case study. A total of 32 datasets are present in the experiment. Besides the BrainVision and odML files, these also include the scenario files (`.sce`, `.exp`, and audio files) and other metadata in the form of text files.

- Driver's attention case studies containing multiple different experiments as part of the [18] project often being solved within student work tasks.

  - Car simulator - driver attention: 18 datasets consisting of the BrainVision and odML files only.
  - Driver's attention and sleep deprivation: Datasets consisting of the BrainVision and odML files with camera recordings from the experiment as `.avi` files. It is made up of a total of 11 datasets.
  - Driver's attention with visual stimulation and audio disturbance: 16 datasets containing scenario files besides the BrainVision and odML files.
  - Driver's attention with auditory stimulation: Includes 15 datasets of BrainVision, odML and scenario files. Scenario files include multimedia files and logging text files.

- Event-related potential datasets based on three-stimulus-paradigm: Experiment consisting of 21 datasets formed by the BrainVision and odML files, accompanied by a pdf license file and a readme text file.

- PROJECT DAYS P3 NUMBERS - This experiment contains 250 datasets. Apart from the BrainVision and odML files, scenario files are included as well in the form of a zip compressed file and a text file in the BrainVision data folder with some additional metadata and results.

3.2.2 FAIR principles assessment

The following is an assessment of the EEGbase portal and the data formats it uses on the individual points of the FAIR principles.

Findable

F1 Neither the data nor the metadata are identified by any unique global identifier.

F2 Data are described with enough metadata about the EEG data and the experiment itself.

F3 Metadata do not include any unique identifiers to the data it describes as there are no identifiers for the data. There is a filename in the BrainVision metadata, where the data are stored, no identifier is present in the experiment metadata. The dataset is identified by an experiment number, which is only local.

F4 The EEGbase web portal serves as a searchable resource for the data and its metadata.

Accessible

A1 Data are retrievable from the EEGbase portal using the HTTP protocol, which is an open, standardized protocol (A1.1) that supports basic authentication and authorization procedures, if necessary (A1.2).

A2 As the data and metadata are stored separately, it is possible for the metadata to exist even without the data.

Interoperable

I1 EEGbase combines odML and BrainVision formats, which can be considered as community agreed formats. For odML though, the version used for the experiment metadata is an older version 1.0, which might require conversions upon further processing.

I2 Vocabularies are used only for the experiment metadata stored in odML in the form of ontologies. No terminologies are used for the BrainVision metadata.

I3 It is not possible to create a full-fledged link from one (meta)data to others, as these should be referenced by a unique identifier.

Reusable

R1 The data contain enough metadata about the recorded EEG data and the experiment itself. The data license is included in the experiment's metadata (R1.1). The provenance of (meta)data is detailed, e.g., describing the experiment's authors or the recording devices (R1.2).

Using this assessment, Table 3.1 overviews how the EEGbase portal and the data formats satisfy the FAIR principles. It fails to fully satisfy the interoperable principles, due to the older versions of some of the technologies and the lack of vocabularies for the data. Some of the findable principles are not met as there are no unique identifiers for the datasets.

Findable				**Accessible**		**Interoperable**			**Reusable**
F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
No	Yes	No	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Partially	Partially	No	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 3.1: FAIR principles assessment for EEGbase data and metadata.

4 Electrophysiological data standards

This chapter lists some of the existing data standards and repositories suitable for storing electrophysiological data. These repositories could be accompanied by additional tools, i.e. visualization or analytical tools, to expand the data workflow. Each of the standards selected in this chapter is evaluated on how well it meets the FAIR principles. Following the evaluation is a selection of a standard that suits the principles best, together with an appropriate repository to store the data in the new data standard.

4.1 NIX

NIX is a project developed by the German Neuroinformatics Node (G-Node). Its idea is to achieve standardized methods and a generalized data model that defines as few data entities as possible while still being able to represent a multitude of different data structures. The generality of designed data means that the naming of entities follows a more general way than the domain-specific terms, which allows for wider use of this standard than just for electrophysiology or neuroscience data [19].

4.1.1 Data model

The NIX data model stores annotated scientific datasets, i.e. data and their metadata as a single container [20]. The entity-relationship schema in figure 4.1 shows all the entities of the model used to store a single dataset. The relation between entities denoted with a circle means that an entity can contain zero to many entities (i.e. a Section entity can contain zero to many Property entities). Relation denoted with a line means there has to be at least one entity in relation (i.e. Property entity has to have at least one Value entity).

The model for data is composed of six main entities:

Block is an entity that acts as a top-level element connecting relevant data objects. A Block can be considered as something like a representation of a dataset or a combined result of an experiment. Every data object has to be associated with one Block object [21].

DataArray is the core entity that stores the raw data in a multidimensional array as well as some additional information about the data, i.e. type, unit or dimension. Dimensions are described separately by the Dimension entity. The DataArray in itself should contain enough sufficient information to interpret its content (e.g., generate a plot) [21, 22].

Tag and MultiTag entities are used to annotate points and regions of interest in one or more DataArrays, for example, an event, a spike train or the presentation of a certain stimulus. They are defined by starting point and duration. More detailed definitions of these Tags than just pointers into the raw data can be described by Feature objects [21].

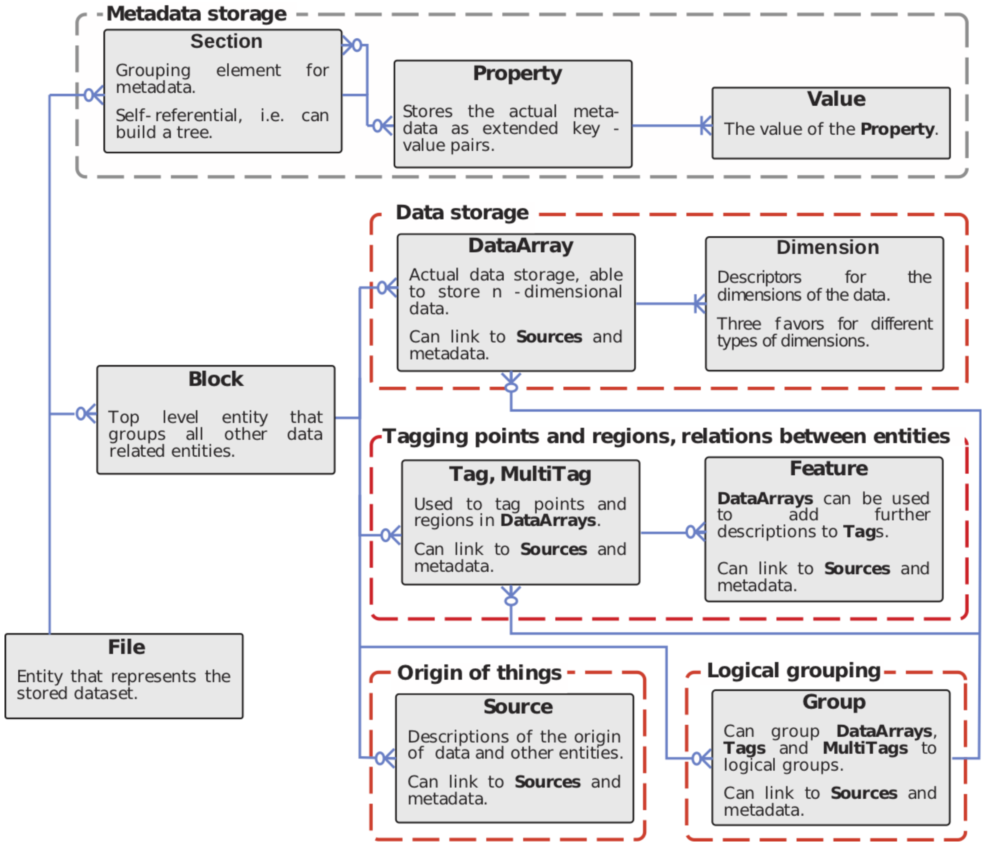
Source entity describes the origin of a DataArray or Tag. Although considered more of a metadata information, it is included for compatibility with other data formats [21].

Group provides a simple grouping of entities below the Block entity level. A Group can contain DataArrays, Tags, MultiTags, and can be linked to Sources and have metadata attached [22].

All data entities have the id, name, type and definition fields in common. The only exception is the Dimension entity that has only the id and definition fields, as others are not necessary. The id field is a unique string identifier of the entity, which can be used to refer to this entity. A domain name and a randomly generated 64 digit hexadecimal number ensure the uniqueness of the id across potentially large collections of data. The type field is a string providing the context to understand the stored data. The definition field is an optional textual definition of the entity. The name field can store a human-readable form of the entity name [21].

The metadata model is mostly identical to the open metadata Markup Language (odML) model. The model consists of properties that contain the descriptive fields id, name, type, definition, and the property's values. These properties can be grouped into sections, differentiating between relevant properties. A section can be nested into another section to form tree-like structures, while the root section can only contain document information like the author's name and version and can not contain any own properties [22]. Metadata entities follow the same principles of creating their identifiers as the data entities [21].

The main data entities can be linked to the metadata by a reference on the corresponding section id. The data entity may refer to only one metadata



Metadata storage
Section
Grouping element for metadata.
Self-referential, i.e. can build a tree.
Property
Stores the actual meta-data as extended key - value pairs.
Value
The value of the Property.
Data storage
DataArray
Actual data storage, able to store n - dimensional data.
Can link to Sources and metadata.
Dimension
Descriptors for the dimensions of the data.
Three favors for different types of dimensions.
Block
Top level entity that groups all other data related entities.
Tagging points and regions, relations between entities
Tag, MultiTag
Used to tag points and regions in DataArrays.
Can link to Sources and metadata.
Feature
DataArrays can be used to add further descriptions to Tags.
Can link to Sources and metadata.
File
Entity that represents the stored dataset.
Origin of things
Source
Descriptions of the origin of data and other entities.
Can link to Sources and metadata.
Logical grouping
Group
Can group DataArrays, Tags and MultiTags to logical groups.
Can link to Sources and metadata.

Figure 4.1: NIX entity-relation schema [20].

section. There are two kinds of relations between data and metadata. If both of the related data and metadata types are identical, it is considered an extension of one another. That is, the information from the data and metadata are complementary. In case the types are not identical, the entities are related, but the data object is not a further specification of the metadata object [22].

Neither the model for data nor the model for metadata (odML) in itself is domain-specific to electrophysiology, but both models can be linked to custom or predefined, deeply described terminologies [23]. Terminologies are collections of names and definitions of sections and properties, which enable the user to give elements of the models a domain-specific, semantic context [22, 24].

4.1.2 Tools

The data and metadata containers, described by the models, are implemented by NIX using a file format called HDF5 [19]. The Hierarchical data format (HDF5) is designed for flexible and efficient storage of complex, high-volume data. The file is a container that holds datasets and groups. Datasets are objects storing the various types of data. Groups are used to organize these data objects into a tree-like structure. Every file has a top-level group called the root group [25]. It is a mature data format standard that is widely supported across programming languages (e.g., C, C++, Python, MATLAB) and tools (e.g. HDFView, ParaView, Jupyter) [26]. There are several tools providing basic operations with NIX files. These include C++ and Python low-level API libraries for reading and writing or a viewer for the NIX data files called NixView [27].

4.1.3 GIN repository

Alongside NIX, G-Node offers a repository service called GIN [28], that allows management of the NIX data, although the system is capable of handling any kind of directory structures and file types. It is based on a combination of the Git version control system and Git-annex that handles the storing of large binary files. The git-like behaviour allows tracking the changes in the data. The repository files are synchronized to a dedicated GIN server, providing secure remote access to the data. The data can be managed through the GIN client (web or desktop) or using a command-line interface that allows management over HTTPS or SSH protocols using git and git-annex directly [29, 30]. The GIN repository offers basic indexing and searching of the contents of individual dataset repositories. A planned extension of this service is to index the contents of the stored NIX files, which would further improve searching of the desired data and metadata in the resource [31].

As GIN is open-source, it is possible to set up the repository to run locally. It furthermore offers several extending services. The GIN-PROC microservice implements the pipeline processing server using Drone. With the use of pipelines, users can design efficient workflows for the data [32]. At this point, this microservice is not available in the GIN's hosted repository yet and can only be integrated into a locally run repository [33]. Another of those services is the service to generate a DOI for a dataset stored in the repository. The DOI (Digital Object Identifier) permanently identifies a digital resource, allowing datasets or files to be citable and accessible through this identifier [34]. Licensing the data uploaded to the repository is possible by putting a *LICENSE* file into the root of the repository, which contains the licensing agreements [35].

4.1.4 FAIR principles assessment

An assessment of the NIX data standard on the individual points of the FAIR principles follows, taking all the findings from this section into consideration.

Findable

F1 Both the data and metadata entities use unique alpha-numerical strings as identifiers. The whole dataset can be identified by DOI if using the GIN repository.

F2 The NIX metadata model allows for a broad description of the data, especially when supported by the use of the predefined odML terminologies.

F3 The reference between data and metadata is one-way only. The data includes the metadata identifier. The only way to trace what data the metadata entity describes is by matching type, which does not necessarily refer to a specific data entity in a dataset.

F4 The usage of a proper repository like the GIN repository may serve as a searchable resource.

Accessible

A1 This principle depends on the selected resource. The GIN repository, offered by G-Node, can serve as a searchable resource. GIN is a free, open-source service that uses HTTPS or SSH protocols (A1.1) which allow for authentication and authorization using GIN credentials (A1.2).

A2 As the data and metadata objects are separated, NIX allows for metadata to exist outside the scope of the data entities, even if they were previously referred to from a data entity that no longer exists.

Interoperable

I1 The standard has a well-defined, formal data and metadata model and uses widely-supported, accessible, open formats (odML, HDF5) that allow for easy data representation.

I2 NIX offers and encourages the use of predefined odML terminologies that are well-described to support the FAIR principles.

I3 References between data and metadata are available using unique identifiers as reference.

Reusable

R1 Using NIX, we are able to attach large amounts of descriptive (meta)data that help the user to decide if given data are useful. Usage of terminologies ensures the relevancy of the description and meeting the domain-relevant community standards (R1.3). Metadata include fields, among others, that serve as detailed data provenance, e.g., institution name or author of the experiment (R1.2). (Meta)data can be licensed by the license included alongside the dataset in the repository where they are stored or have it included directly in the metadata (R1.1).

Table 4.1 summarizes the FAIR principles assessment for the NIX standard. The NIX standard meets most of the principles. The only principle that is not met according to the definition is the findable principle F3. However, an explicit association between the metadata file and the dataset is still present, only the other way as the principle states. That way, we can say this principle is at least partially met.

Findable				**Accessible**		**Interoperable**			**Reusable**
F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
Yes	Yes	Partially	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Yes	Yes	Yes	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 4.1: FAIR principles assessment for the NIX standard.

4.2 BIDS

The Brain Imaging Data Structure (BIDS) is a standard initially developed for organizing and sharing neuroimage experiments. The standard specifies the description of the experiment data and the related metadata in a filesystem hierarchy [36]. For the purposes of this thesis though, we need a way to store EEG data instead of neuroimaging data. That is possible with the usage of the EEG-BIDS extension [37]. This section describes BIDS with the usage of this extension.

4.2.1 Data structure

The extension of BIDS to EEG data closely follows the general BIDS specification (the hierarchical structure of a dataset is shown in Figure 4.2). The filenames for both the data and metadata are formed with a series of entities or key-value pairs, a suffix, and end with a file type, where keys, entities and file types are predefined in BIDS specification [38] and values are chosen by the user [37]. For a data file that was collected in a given session from a given subject, the file name will begin with the string `sub-<label>_ses-<label>` [36].

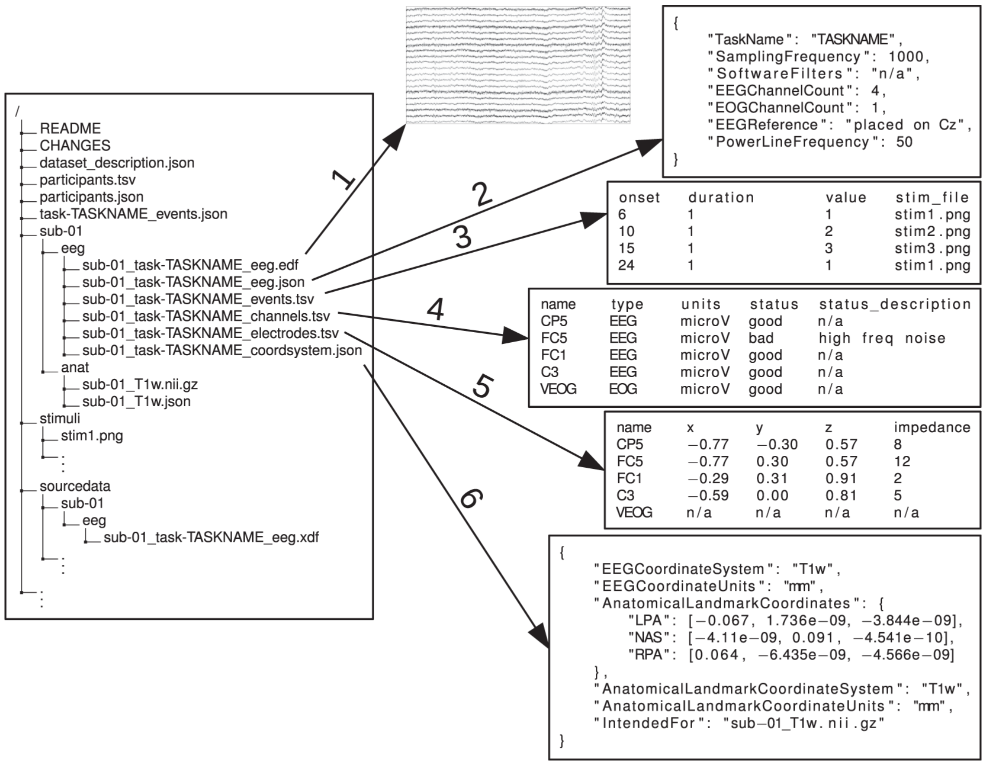
BIDS is designed to standardize and describe raw data, meaning data unprocessed or minimally processed due to file format conversion. During analysis, such data will be processed and both the intermediate as well as final results will be saved. The EEG data have to be stored in one of the following formats for BIDS to be able to process the data [37]:

- 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*)

It is recommended to use either the EDF or BrainVision format. Any other format, that is not supported for standardization, can be placed into the sourcedata folder [37]. The data file is represented as (1) in figure 4.2.

The metadata consist of files with extensions *.json*, *.bvec*, and *.tsv*. JSON file metadata store key-value pairs. Tabular data are stored as tab-delimited values (TSV) and are meant to store additional experiment description data (than can further be annotated with JSON metadata), e.g., the occurrence of events in an experiment or electrode properties. The values from the top level are inherited by all lower levels unless they are overridden by a file at the lower level. Metadata that are not associated with any subject may only be stored in the root directory, as metadata stored in the root directory serve as a general description of the dataset [36, 37]. To allow for more flexibility of the standard, only a small subset of metadata fields and files, needed to perform standard basic analyses on each type of data is required [36].

The TaskName field is inherited from the base BIDS standard. Other required fields, specific for the EEG extension, are EEGReference (general



```
/
├── 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 4.2: Exemplary EEG-BIDS dataset [36].

description of the reference schema), SamplingFrequency, PowerLineFrequency, and SoftwareFilters. To perform some additional types of analyses, there are other crucial metadata that are recommended to be included (information like EEGChannelCount or RecordingDuration for the EEG data). When a recommended piece of metadata is missing, the BIDS Validator will report a warning. [36, 37]. The full overview of required and recommended metadata fields for each metadata type is in the BIDS specification [38]. The specification also defines a large amount of optional metadata. These metadata fields contain information that is not necessarily suitable for any of the data analysis methods, but can be useful when trying to understand the nature of the data [36]. Exemplary metadata shown in figure 4.2 contain JSON metadata for the EEG data (2) and the EEG coordinate system (6), and tabular metadata for events (3), channels (4), and electrodes (5).

4.2.2 Tools, the OpenNeuro repository

For working with the datasets, BIDS offers tools for Python and MATLAB. Both tools allow for queries over the BIDS entities and basic manipulation and reading of the data within the files. The querying provides searching files by keywords and/or metadata, or retrieving file-associated metadata across the BIDS hierarchy [39, 40]. A python library MNE-BIDS implements the reading and writing of the BIDS structure [37].

The BIDS specification mentions the possibility to use OpenNeuro as a repository for BIDS standardized data. The OpenNeuro is a web-based repository platform for managing and sharing neurophysiology data. Only BIDS compatible datasets are accepted, each dataset is validated prior to the upload. The datasets can be uploaded as private for up to 36 months until they become public under the Creative Commons CCO license. The platform provides access to computationally expensive, state of the art pipelines for data analysis. Pipelines are run on immutable snapshots of data (versions) so the results are fully reproducible. Apart from the available set of pipelines, users can deploy their own pipelines to the platform [41].

4.2.3 FAIR principles assessment

The following is an assessment of the BIDS data standard on the individual points of the FAIR principles, considering the findings from this section.

Findable

F1 Neither the data nor the metadata are assigned a unique identifier. Only the dataset can be identified by the DOI, but that is dependant on the repository used.

F2 Data are described with sufficient metadata.

F3 As the data are not identified by any unique identifier, metadata do not include any explicit identifier of the data it describes.

F4 The OpenNeuro repository may serve as a searchable resource for the BIDS datasets.

Accessible

A1 The OpenNeuro repository, designed to store the BIDS datasets, allows retrieving the datasets using HTTP(s) as a standardized communication protocol (A1.1), with support for authentication and authorization by using user credentials if necessary (A1.2).

A2 As the data and metadata are stored in separate files, metadata will still be accessible when the data are not anymore.

Interoperable

I1 The standard utilizes formal, broadly applicable languages for representation in the form of the BrainVision or European data format for the data and JSON for the metadata.

I2 The (meta)data are described using the BIDS specification.

I3 The BIDS standard does not support referencing as there are no identifiers to (meta)data.

Reusable

R1 The data and metadata are described with rich domain-specific attributes, thanks to the BIDS specification. The standard does not contain licensing attributes. However, any datasets published onto the OpenNeuro repository are licensed under the Creative Commons license (R1.1). The BIDS specification includes required and recommended (meta)data to help with its provenance (R1.2) and with meeting the domain-relevant standards for similar datasets (R1.3).

An overview of the FAIR principles assessment on the BIDS standard is in table 4.2. Most of the principles are satisfied, but the lack of a unique identifier for both the data and metadata means it fails to satisfy principles F1, F3, and I3. The F1 principle could be at least partially satisfied by identifying the whole dataset by a unique identifier like DOI, although, that depends on the used repository to store the data.

Findable				**Accessible**		**Interoperable**			**Reusable**
F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
No	Yes	No	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Yes	Yes	No	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 4.2: FAIR principles assessment for the BIDS standard.

4.3 NWB

Neurodata Without Borders: Neurophysiology (NWB:N) is a project that defines tools, methods, and data standard providing a common solution to

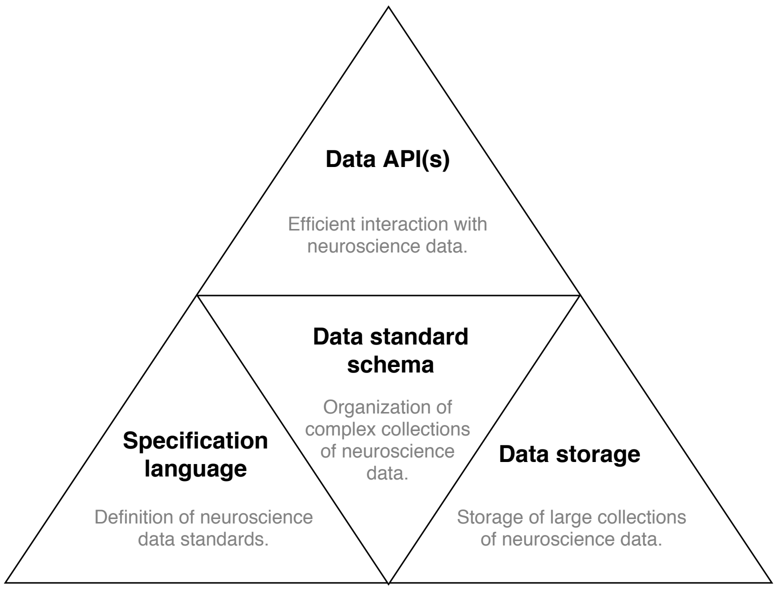

Data API(s)
Efficient interaction with neuroscience data.
Data standard schema
Organization of complex collections of neuroscience data.
Specification language
Definition of neuroscience data standards.
Data storage
Storage of large collections of neuroscience data.

Figure 4.3: The NWB:N project structure [43].

share, archive, use, and build common analysis tools for neurophysiology data. It aims to store general optical and electrical physiology data in a way that is interpretable by both human users and software tools and analysis scripts, and to impose a priori assumptions about data representation and analysis [26, 42]. Overview of all the parts of the NWB:N project and their purpose are in Figure 4.3.

4.3.1 Data standard schema

The format defines and uses the NWB:N specification language [44]. The specification language defines formal structures for describing the organization of the data, as well as enabling the definition of extensions to support the integration of currently unsupported data [26]. The basic concepts of the NWB format that hierarchically organize the data are the following [42]:

Dataset is an n-dimensional array that is the primary means for storing the data.

Attribute is a small dataset that is attached to a specific group or dataset. Attributes are typically used to store metadata specific to the object they are associated with.

Group is similar to a folder in a file system. It may contain an arbitrary number of other groups and datasets.

Link is used to refer to another group or dataset.

All datasets and groups in the format can be uniquely identified by either their name and/or type (the neurodata_type of the data component) [26].

Every main semantic component of the format has a unique neurodata_type, similar to a class in object-oriented design. That means a possibility for reuse and extension of types through inclusion and inheritance. The two important base types are NWBContainer and TimeSeries. NWBContainer defines a generic container for storing collections of data. It is used to define common features and functionality across data containers. The TimeSeries type is the central component for storing complex temporal series, as neural data typically involve measurements taken over time. This type is extended via sub-classing to account for different storage requirements and data modalities (e.g., ElectricalSeries for electrophysiology or ImageSeries for optical imaging) [26, 42].

The data organization at a high level in an NWB file uses the following main groups [26]:

Acquisition for storage of data streams recorded from the system, e.g., recordings from electrophysiology tracking systems.

Processing for the standardized processing modules, often as part of intermediate analyses required before scientific analysis, e.g., results from spike sorting, signal filtering, or image processing.

Intervals for the experimental intervals, e.g., experimental epochs or trials.

Stimulus for storage of stimulus data.

General for storage of experimental metadata, e.g., description of the hardware devices, author of the experiment or other notes.

Analysis for storage of lab-specific and custom scientific analysis of the data.

All Datasets and Groups in the NWB with an assigned neurodata_type have three required attributes. The name of the primitive that it maps onto as a neurodata_type string, the namespace where the neurodata type is defined, e.g. “core” or the namespace of an extension, and an object_id. The object_id attribute is a universally unique identifier of the object within its hierarchy. It should be set upon creation as a string representation of a random UUID that has been generated according to the RFC 4122 standard [42, 45].

4.3.2 Tools and data storage

The NWB format uses the HDF5 file format as the primary mechanism of the data storing (same as the NIX data standard). The NWB primitives described by the specification language largely match the HDF5 primitives, so the mapping is mostly a 1-to-1 mapping [26].

The NWB:N project offers data APIs facilitating an efficient interaction with the neuroscience data stored in the NWB file format. The interaction comprises of reading, writing, querying, and analyzing the neurophysiology data. The currently developed APIs are PyNWB (Python) and MatNWB (Matlab) [46, 47]. Both provide easy-to-use representations of NWB neurodata types for programmatic use and enable the mapping of these representations to/from data storage based on the format specification. The use of interfaces allows for object extension as a way to create any missing objects or properties [26].

The NWB:N does not determine any repository that is designed to store the NWB format specifically. However, other repositories, that do not restrict the stored data format could be used. Repositories suitable for any of the standards described so far are discussed in the following section.

4.3.3 FAIR principles assessment

The following is an assessment of the NWB data standard on the individual points of the FAIR principles, considering the findings from this section.

Findable

F1 All datasets have a required attribute of a unique identifier that is set upon their creation.

F2 Datasets include metadata description of the raw data in the form of attributes, as well as the metadata about the experiment itself in the "General" group.

F3 The attributes of the metadata include the object_id as the explicit identifier of the data.

F4 This principle depends on the selected resource. The NWB:N project does not recommend any repository that could be used as a searchable resource. However, any repository that supports the NWB file format fits this purpose (e.g. the GIN repository with the support of the HDF5 file format).

Accessible

A1 Again dependant on the used repository, but it is possible to access the NWB datasets using repositories with a standardized communications protocol like the GIN repository, which is covered in the FAIR principles assessment for the NIX standard.

A2 Only the general metadata and other custom metadata are accessible when the data are no longer available. The metadata about the raw data are attached directly to the data as attributes.

Interoperable

I1 The datasets are stored using the accessible, open format HDF5. The data and metadata models are well described by the NWB:N specification language.

I2 Vocabulary for the data is part of the NWB:N specification language. However, no such standardized vocabularies are available for the metadata. That is the object of further improvements with the integration of controlled vocabularies, and ontologies for the metadata [26].

I3 It is possible to use the unique object identifiers for the metadata attributes to refer to other (meta)data.

Reusable

R1 The attributes to store metadata allow us to include enough information to describe the context under which the data were generated. That includes the information about the experiment or the device used to prove the origin of the data (R1.2). The NWB:N does not handle the data usage licensing. Although, licensing the data (R1.1) is possible with the use of a suitable repository (e.g. the GIN repository). The reusability for similar datasets is supported by the NWB:N specification language (R1.3).

An evaluation of the FAIR principles assessment is in Table 4.3. Most of the principles are fully satisfied. The principle A2 is met only partially, as the accessibility of the metadata about the raw data depends on the accessibility of the data. However, the general metadata will still be accessible. The principle I2 could be fully met in the future as the metadata ontologies are planned to be implemented to the metadata model.

Findable				Accessible		Interoperable			Reusable
F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
Yes	Yes	Yes	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Partially	Yes	Partially	Yes	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 4.3: FAIR principles assessment for the NWB:N standard.

4.4 Repositories

As the EEGbase portal is not designed to store any of the electrophysiological data standards mentioned above, it is necessary to use different service as a repository for the data. Repositories for scientific data should allow for proper metadata management alongside the data. Besides the GIN repository suggested for the NIX standard and the OpenNeuro repository that is meant for storing the BIDS data standard, other repositories, capable of storing any of the data standards are available.

4.4.1 LORIS

LORIS is an open-source web-based platform for storage, sharing and visualization of research data. As it actively supports integration with the BIDS neuroimaging standard, the platform is fit for storing neuroimaging, behavioural, or electrophysiological data. However, its modular architecture allows customizability for any data type and any activity workflow [48]. LORIS can be integrated with the CBRAIN service, which offers collaborative high-performance computing platform for processing, combining visualization tools and fully automated software pipelines [49]. The integration to CBRAIN, shown in figure 4.4, or any other service is done via RESTful API, which can be customized to restrict access to data or activities.

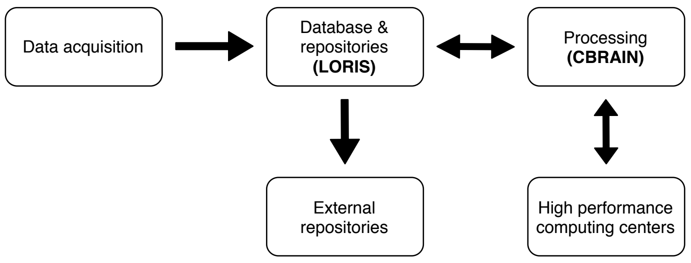

Data acquisition
Database & repositories (LORIS)
Processing (CBRAIN)
External repositories
High performance computing centers

Figure 4.4: The LORIS and CBRAIN integration [49].

4.4.2 Eudat

Eudat is a collaborative data infrastructure offering a multitude of data management and sharing services. Amongst them is the B2SHARE data store service, consisting of four main modules: Data Storage, APIs, Access Control, and a web-based User Interface. The data storage is separated into the management of scientific data, the associated metadata and identity data. The scientific data are the produced research data and can be of any type. Metadata are the explanatory data and they are being filled upon the object data upload. The identity data serve as information for the access control module and for integration with other services [50].

B2SHARE provides a REST API as an interface for creating new deposits, upload data to it and share these data. Another API called OAI - PMH can retrieve complete metadata collections from the repository. To make data reusable and citable, B2SHARE is using persistent identification PID as digital object identifiers, compatible with the DOI system [51]. The identifier is assigned to the dataset during the deposit process. The complete architecture overview of the B2SHARE service is in figure 4.5. The integration of B2SHARE with the other EUDAT services for data management, sharing, and discovery like B2STAGE or B2FIND is what distinguishes it from other data sharing services [50].

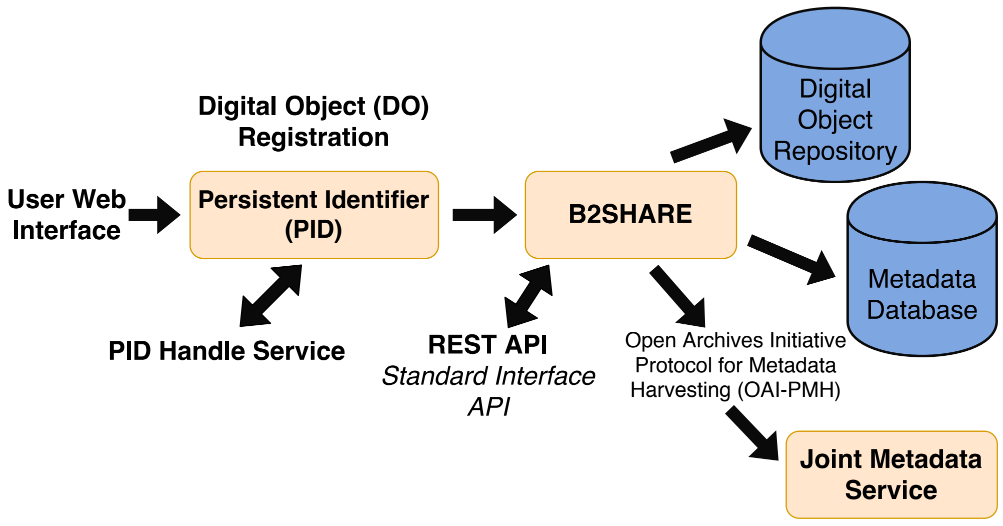

Digital Object (DO) Registration
User Web Interface
Persistent Identifier (PID)
B2SHARE
Digital Object Repository
Metadata Database
PID Handle Service
REST API
Standard Interface API
Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)
Joint Metadata Service

Figure 4.5: The B2share service architecture [50].

4.4.3 EBRAINS

EBRAINS is a platform developed by the Human Brain Project offering a collection of services for data and knowledge management or the brain model simulations [52]. However, the platform is not open-source, so it can not be run locally and can only be used as the remote service. The sharing of the research data works on the basis of curators. The raw or processed data are submitted via a request form to the EBRAINS Data Curation team, which reviews the potential of the data/model collection. In case of acceptance, the data collection is integrated (file upload to the long-term storage and metadata registration) and published to the platform. EBRAINS assigns a citable DOI to any published collection [53]. Data collections published to EBRAINS can be accessed through a web-based user interface Knowledge Graph [54].

4.5 Overview

4.5.1 Data standard

All the researched electrophysiology data standards were expected to follow most, if not all, of the FAIR principles. When selecting the most appropriate standard, the difficulty of data conversion from the currently used BrainVision/odML formats was taken into consideration as well. Table 4.4 compares the FAIR principle evaluation for the EEGbase portal and the proposed data standards. The EEGbase and its infrastructure for storing the research data fully satisfy only 5 out of the 10 main FAIR principles. Moreover, the unnecessary complexity and the older technology used for the portal implementation would only lead to more difficulties in the future maintenance of the project. This supports the proposed conversion of the current state to a newer, more maintainable solution.

The BIDS standard is falling short on the FAIR principle assessment in comparison with the other data standards. The folder-like structure of the standard could be seen as a drawback, as it could create an extra step when manipulating the data, e.g., compressing the folder before being able to upload it into a repository. At the data standard comparison stage, the BIDS python tools were tested by writing a simple python script that created a basic structure with the skeleton of some EEGbase data. The technical documentation for the offered BIDS Python API came out as not thorough enough as the specifications for the Python APIs for the NIX or NWB data standards. Due to these findings, the BIDS standard was determined as the

Standard	**Findable**				**Accessible**		**Interoperable**			**Reusable**
	F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
EEGbase	No	Yes	No	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Partially	Partially	No	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)
NIX	Yes	Yes	Partially	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Yes	Yes	Yes	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)
BIDS	No	Yes	No	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Yes	Yes	No	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)
NWB	Yes	Yes	Yes	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Partially	Yes	Partially	Yes	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 4.4: FAIR principles assessment overview.

least favorable replacement of the current infrastructure.

The remaining NIX and NWB data standards satisfy all of the FAIR principles, partially at least. Unlike the BIDS standard, both standards use the HDF5 file format, which would join the BrainVision data and odML metadata into a single file. With HDF5 being a generic format, accessibility should not be affected. Both NIX and NWB have easy to use, deeply described Python APIs that would be used to create a conversion script from EEGbase to the selected format. However, as for the PyNWB tool, some of the EEGbase metadata properties are not present in the NWB object interfaces. That means custom objects extending the current NWB objects would have to be implemented. Creating these could be very time consuming if there was to cover any possible metadata properties.

The NIX data standard uses odML to store metadata, same as the EEGbase datasets. Apart from the libraries for manipulation with the NIX files, G-Node offers additional python tool "nix-odML-converter", that can convert an existing odML file into a NIX container. The tool will also automatically convert the odML file into the newest odML version [55]. This is a significant advantage, as the current EEGbase odML metadata are stored in an older version. Using this tool and adapting it to our needs (specifically adding terminologies for the metadata) is an easy way to convert the session metadata from EEGbase. Accompanied by the MNE tool to parse the current BranVision data and metadata, it makes the conversion from current EEGbase portal into NIX the least difficult, while satisfying the FAIR principles and preserving all current experiment metadata. The NIX data standard is selected as the most appropriate to convert the EEGbase data into.

4.5.2 Repository

With the NIX data standard selected, we need a service that is able to store the converted data, replacing the current EEGbase portal. When selecting a repository suitable to store the new data standard, the following criteria were taken into consideration:

- The repository should offer some means of persistent identification of stored datasets to support the findability principle.
- The repository implements some form of pipelines to allow for future extensions of the data workflow.
- Being open-source is not a strictly required feature of the repository, as long as the service assures longevity and persistence of stored data. On the contrary, having an externally maintained service with all the desired features would be only beneficial.

Table 4.5 shows an overview of the evaluation of all the researched repositories based upon these criteria. This does not include the OpenNeuro repository, as it is currently supporting only BIDS standardized datasets and cannot be used to store the NIX standard.

Despite the EBRAINS platform only being available as a closed-source hosted service, it does not restrict the uploaded data types, making it suitable for the NIX data standard. In addition, it identifies datasets with a DOI. However, there is no possibility to create pipelines, and the process of the data review through a curation team might limit which of our datasets can actually be uploaded. This makes the EBRAINS platform not suitable to store the data.

The LORIS storage platform is originally developed to store the BIDS format, which means modules and possibly other extensions of the source

Repository	**NIX support**	**Persistent identification**	**Pipelines**	**Open-source**
GIN	yes	yes	yes	yes
LORIS	not in default	no	yes	yes
Eudat	yes	yes	no	no
EBRAINS	yes	yes	no	no

Table 4.5: Repository evaluation.

code would have to be implemented to allow for the NIX data standard to be stored. Even though there are pipelines present to extend the data workflow, they are designed for use on the BIDS standard. The datasets are not identified with any persistent ID like DOI or PID. In addition to the necessity of module implementations to support NIX, this does not make the LORIS platform suitable.

The Eudat infrastructure and its services are capable of storing the NIX data standard, and, unlike the EBRAINS platform, it does not limit the data manipulation by any uploaded dataset curation. The PID handle service can assign datasets with a persistent identifier. Although offering some additional services, no service that would allow for pipeline creation is present, being the reason why Eudat was not selected as the most favourable repository.

The GIN repository is open-source. That makes it possible to run the repository locally, while G-Node hosts a publicly accessible repository as well. It allows for persistent identification of the datasets using the service for generating DOIs. However, it is worth noting that the service is free of charge only on the G-Node hosted repository. Although the service can also be implemented locally, a paid membership would be required in that case. The repository being built on git is a big advantage, as any pipeline extensions for git should be possible to implement to GIN as well. Pipeline support is offered by G-Node directly with the GIN-proc microservice. As the GIN repository meets all the desired criteria, it was selected as the best repository to use.

5 The dataset conversion

Now that the NIX data standard is selected as the appropriate replacement for the current EEGbase dataset structure, we need ways of converting the data and metadata from the old format to the new one. The goal is to design and implement a tool that converts the data and metadata without any loss of original information while making sure the dataset in the NIX format satisfies all the possible FAIR principles.

5.1 Architecture

When converting the dataset, first, we need to take into consideration the structure of the current data. As the EEG data from the BrainVision format and the odML experiment metadata are stored separately, the conversion has to be split up into two segments. The proposed data flow diagram of the conversion to the NIX data standard is in Figure 5.1. The logical structure of the NIX container will follow the findings from the standard analysis. Designing the conversion tool was made with the guidance of the existing datasets from different experiments from the EEGbase portal. However, the final design was made more general to allow for a conversion of future datasets that might have a slightly different property structure.

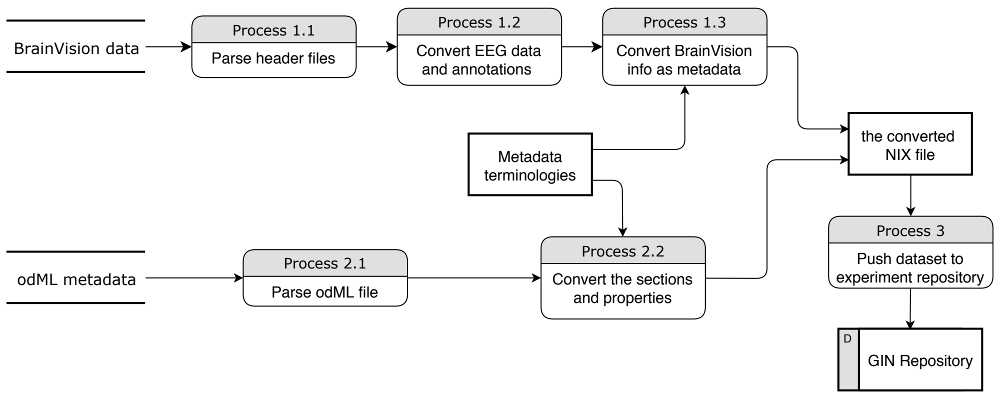


Figure 5.1: The data-flow diagram of the conversion.

5.1.1 BrainVision data conversion

The data conversion segment processes the BrainVision EEG data. These data consist of a triplet of the raw data, marker data of the stimuli (also called annotations), and a header file containing the metadata of the recording devices. As the header file links to the related raw and marker data, it is the only BrainVision file we need for the conversion as the input.

Multiple of these triplets can exist in a single dataset, which has to be taken into account for the parsing of the input. A dataset is a single experiment performed on a subject according to the given scenario. A single experiment can have multiple recordings. Figure 5.2 shows an overall view of the structure of a NIX dataset created from the conversion of the BrainVision data. The schema depicts a conversion of a dataset that has multiple BrainVision recordings and their separation into groups.

All of the recording data triplets of a dataset will be stored in a single NIX container, where the NIX group entities will be used to create a logical structure and differentiate between the recordings. Every single recording

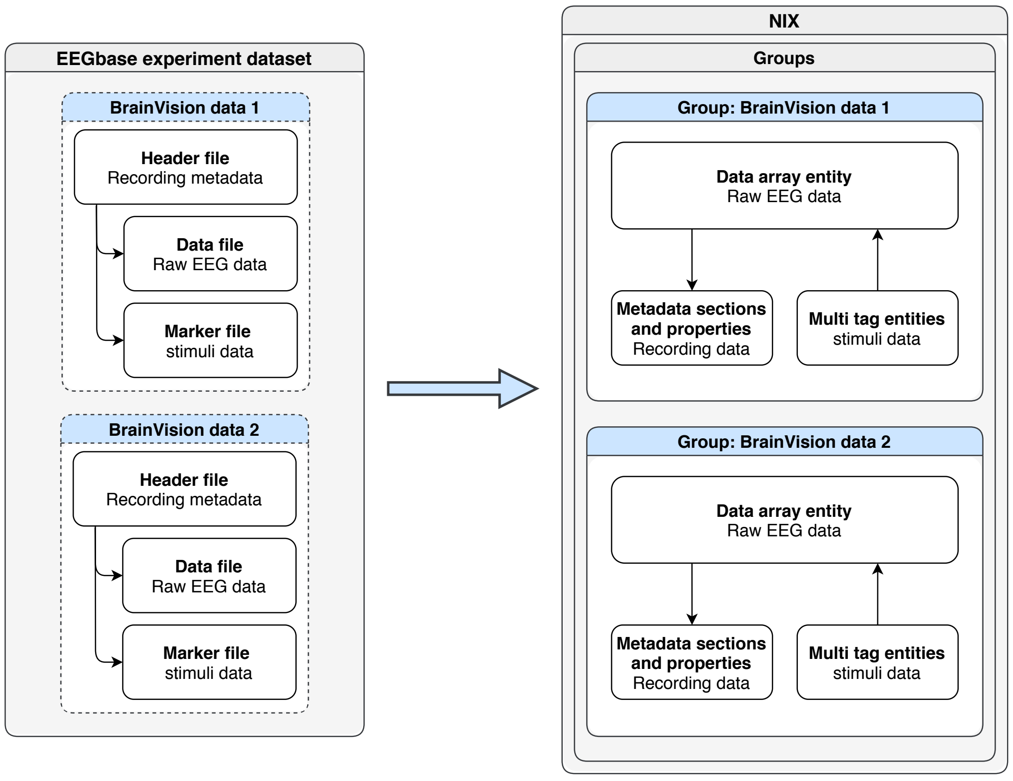


Figure 5.2: The converted NIX logical structure schema.

will be put into its own group. The group names in NIX have to be unique. As the BrainVision does not have any recording name property that would ensure its uniqueness, the raw data file name of the recording, together with its file extension, will make up the group name. The group creation is part of Process 1.1 of the diagram in Figure 5.1. The group can be created as soon as the data file name is parsed from the header file.

The process continues with the conversion of the recording data, i.e., the raw EEG data and the markers which highlight specific stimuli points or ranges in the recording. This part of the process corresponds to Process 1.2 of the diagram in Figure 5.2. Any n-dimensional data from the recording will be stored in the NIX container as a Data array entity. For the raw EEG data, the array will be converted directly, as it is already stored as 2-dimensional arrays in the BrainVision format, where one dimension represents the channels, labeled by their names, and the second dimension is the time series data of the channel.

A stimulus in the marker data is stored in the BrainVision header as a single line, described by the stimulus name, position in data points, and size in data points. The Multi tag entity that is designed to highlight points or areas in the data will be used as the target entity. However, the original stimuli data have to be adjusted to fit the Multi tag raw EEG data stored as the NIX Data array entity. The Multi tag's stimuli positions and their

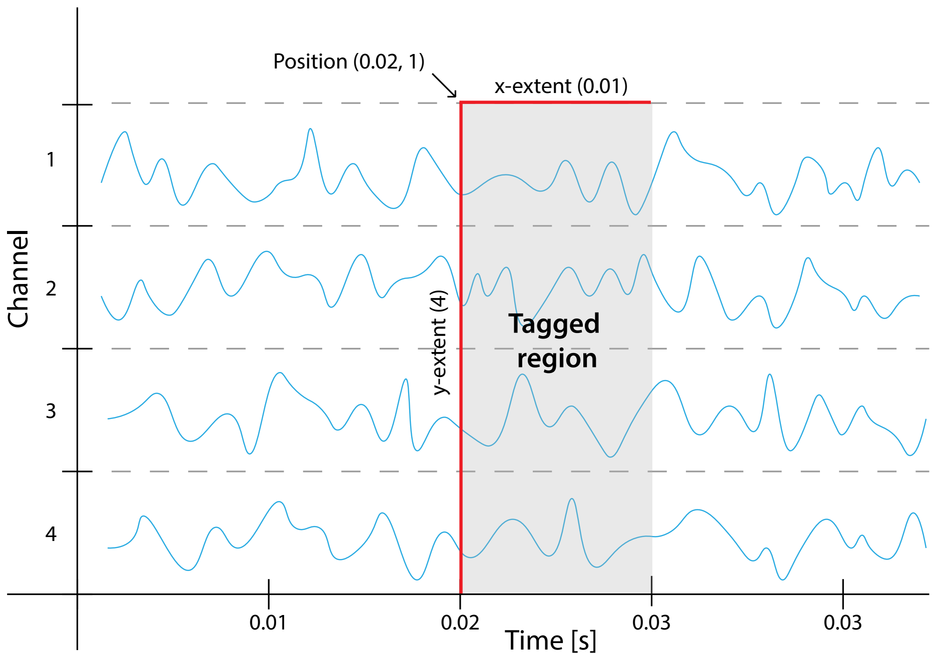


Figure 5.3: Depiction of a Multi tag in Data array.

extents are stored as a Data array entity, and they have to share the same dimensions as the referencing Data array. A Multi tag region highlighting the raw EEG data is depicted in Figure 5.3. The known data From the original BrainVision marker are the first element of the starting position (the value 0.02 in the figure) and the x-extent of the marker. These need to be modified, so they match the dimensions of the data and are applied to all the channels. This means adding the first channel to the positions of the stimuli and the last channel to the extents as a second dimension. This creates a Multi tag highlighting the data from all the channels, as shown in Figure 5.3. The descriptions that name the individual stimuli are added as labels of the position and extent Data array entities. Once the Multi tag contains all the stimuli, a reference is added to the raw EEG Data array entity.

The recording metadata information is processed from the BrainVision header file after all the data are converted and added into the NIX container, depicted in Figure 5.1 as Process 1.3. Every recording will have its own section in the NIX container. This will allow us to distinguish the metadata between different recordings and reference the correct metadata section to the appropriate data array. The header file can contain multiple representations of information:

- The key-value pairs will be represented as properties in the section. If the info value is multidimensional, a data array entity will be created to hold the values, and the metadata property will refer to this array.
- If the values are a list of multiple values (e.g., the channel info, which contains multiple values separated by a comma), an additional section will be created with the key name, and the list will be put as properties into this section.
- Any other metadata like comments or other settings, where these rules can not be applied, will be treated separately and processed to keep the original meaning.

The current BrainVision metadata does not use any terminologies that could be used for the converted metadata. The odML terminologies will be used to define the recording metadata structure, as will be the case for the odML session metadata, and applied where possible. This means that the BrainVision metadata will not be converted as initially parsed. Instead, the parsed data will be remapped to fit the odML terminologies before inserting it into the NIX container. The BrainVision metadata will undoubtedly contain parameters that do not have any matching interpretation in the terminologies. These metadata will be converted either way, with the naming conventions and structure following the odML terminologies structure, to keep as much original information as possible.

The metadata have to be processed after the data conversion so that the reference to the newly created NIX sections can be added to every data array in the group. The processes 1.1 through 1.3 of the BrainVision dataset conversion are repeated for every single recording in the dataset.

5.1.2 Session metadata conversion

The odML metadata in the BrainVision dataset have a very similar structure to the structure of the NIX metadata model. Both use sections that group properties together, and the properties hold the metadata values. This means that the sections from the original odML metadata will be represented as the Section entities of the NIX metadata and the odML properties as the NIX Property entities. The odML metadata will be parsed and stored into the script's internal structures. These structures will be used for the modification of the metadata to fit the NIX metadata model. The parsing is represented as process 2.1 in Figure 5.1.

The original metadata stored in the odML format register the odML terminologies as the repository to represent the metadata structure. However, the version that was used to create the metadata is deprecated and no longer officially available from the G-Node's odML repository. This means that the odML metadata needs to be modified to use the current version of the terminologies. The modification of the metadata with an implemented mapping class will be a part of the conversion into the NIX sections and properties. The mapping class will take the original odML document as an input. The mapper will create a new odML document matching the current terminologies using a user-defined mapping file. A default mapping file for the odML terminologies to fit the EEGbase datasets will be provided.

This document will be used to insert the metadata into the NIX container instead of the original one, as shown in Figure 5.4. Each section and property type is represented with a mapping entity in the mapping file. The entity consists of the original type of the section or property and a corresponding term structure defining the section or property in the NIX file. The corresponding term includes the new name and type of the NIX entity plus a parent section in the NIX container to allow for an association of a section or property to a different one. Using the original type and the new remapped type, necessary property data type conversions can be made. If a

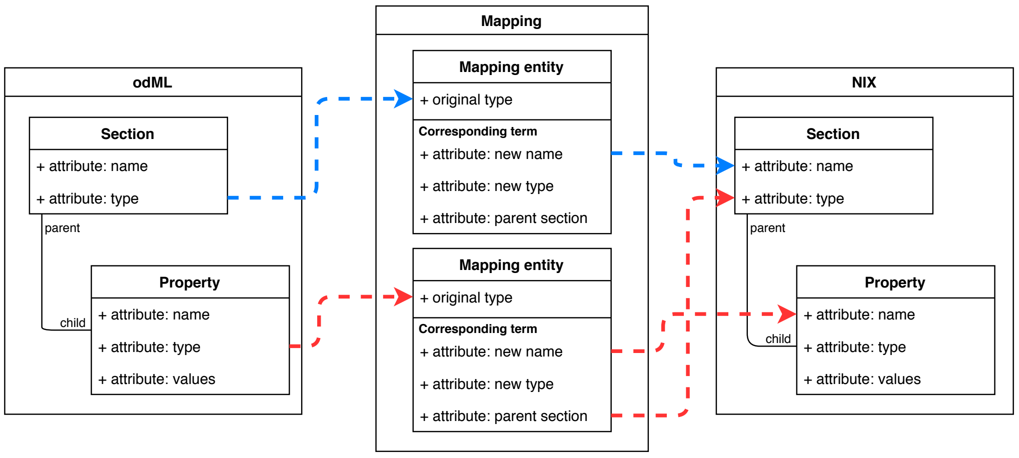


Figure 5.4: The mapping of the odML document to NIX entities.

section or property does not have a mapping entity equivalent, it will not be included in the converted metadata. Empty or missing metadata properties will be omitted during the conversion, and no section or property will be created in the NIX metadata. This also applies to properties from the original odML metadata with the "GUI" prefix, as these properties were used for the EEGBase portal web interface and are no longer necessary.

Aggregation of all the sections and properties across all the datasets that will be converted into NIX revealed that most of the current odML metadata is applicable to the newest odML terminologies, with only a small portion needing for remapping to a different section and/or property. Only the following odML metadata do not have a clear interpretation in the odML terminologies:

- The Private-experiment property in the Experiment section - This property will not be needed as the public/private attribute of a dataset will be determined by the repository setting.
- The Subject's Comorbidities, Myopia properties, and the Diseases section - These properties will be joined together to form the array of values for the HealthStatus property of a Subject of the odML terminologies.
- The Person user role - This property specifies the role (user or admin). This property is not necessary. Furthermore, it collides with the person's role from the odML terminologies with a different meaning.

- The Artifact section - This section will be ignored as no dataset stores any actual information apart from the "NotKnown" value in the properties of this section. Additionally, artifact rejection is part of the BrainVision metadata as well, and these are parsed into NIX.

The tool will also offer an option where no terminologies mapping will be made, and the metadata will be converted as they are. The conversion tool will include a method that allows the user to generate a new mapping file. The generator will aggregate any passed odML metadata files to create the mapping for any found sections and properties. The parent section for each section and property will be filled according to the original file, so when using such a generated file, the converted metadata should have the exact same structure as the original.

5.2 Python tool

Tools for handling the BrainVision files and creating the NIX files, that can be utilized to help with the conversion between these formats, are both developed as python libraries. That is why the conversion tool is written as a python script. The class diagram in Figure 5.5 can express the overall structure of the implemented tool. The tool is implemented as the `nixconverter` package, containing the main class of the tool called `EEGBaseNixConverter`. The package is further split into submodules by the types of files it converts, i.e., the BrainVision converter and the odML converter.

The `EEGBaseNixConverter` class acts as an entry point of the tool that parses the user input and creates the base structure of the converted datasets. The user input is provided by the python Docopt library as the script's arguments, and their values are stored as the class attributes that are passed further on. Any paths from the Docopt parser are modified to absolute paths so that the user can pass either a relative or absolute path to a dataset, for example. The creation of the base structure of the converted dataset consists of creating a new empty NIX file with the root block where the converted contents of the dataset will be put into. The second part is copying any files that cannot be processed in the conversion into the converted dataset. This may include the original scenario information, result data, or license pdf files.

Helper functions and classes are implemented into the "utils" submodule. This submodule also includes a configuration file with constants used for the conversion. The `HeaderProperties` class is a utilities class that stores basic information about the currently handled BrainVision header file. These

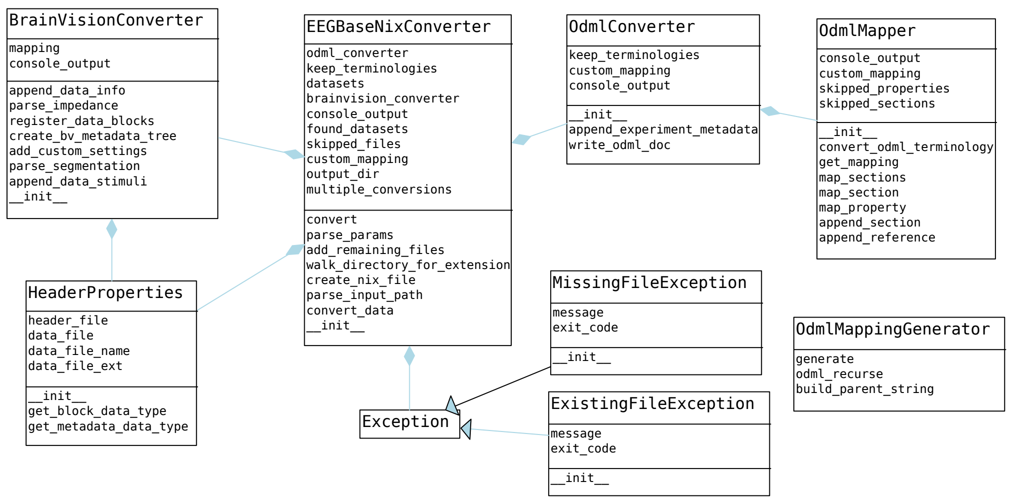


Figure 5.5: A class diagram of the python conversion tool.

properties are further used during the parsing and conversion. Other utilities include custom exceptions, debugging output, and path parsing functions.

5.2.1 BrainVision converter

The open-source MNE library [56] handles the parsing of a single BrainVision data triplet (the header, marker, and data file). The BrainVision parsing function takes a path to the header file as an argument and returns an internal structure containing the data, metadata, and marker information from all three files. The open-source python tool nix-mne [57] does convert the MNE structure to NIX files. However, it does not create the dataset to fit the FAIR principles. The basic practices of the parsing functions from this tool are taken and modified to satisfy the FAIR principles.

The first step of this conversion is registering the data group in the NIX file and appending the raw EEG data. The group is named after the name of the header file. The parser scales the recording to a basic unit of V. Even though the original data are stored in μV, the unit conversion is not a problem, as it also updates any necessary metadata information with the data scaling. The raw data are placed into a new NIX `data_array`, a type of the entity depends on the extension of the raw data file to distinguish between the raw EEG data and the averaged data. The dimension labels are set as the channel names, and the time range and unit of the data array is set to V. This data array is appended to its group.

Once the raw EEG data `data_array` is created in the NIX file, the converter administers the stimuli parsed from the marker file, if there is any. It takes the list of onsets and durations from the MNE annotations and places them in new `data_arrays`. The remaining descriptions from the annotations are used as labels for these data arrays. These data entities are used to create a `Multi_tag` entity to represent the stimuli information. This entity is placed into the appropriate group, and a reference is made to the raw data entity.

The metadata from the header file is stored in the `bv_info` dictionary parsed by the MNE library. However, this function does not parse all the metadata that we need to retrieve and move into the NIX file. Before converting this dictionary to the NIX section and property entities, the remaining necessary information is parsed. This is done using the protected function of the MNE BrainVision parser `_aux_vhdr_info` which returns the raw contents of the header file and a `ConfigParser` to parse INI properties.

The first of the metadata missing in the MNE parsed structure is the channel impedances section. This section is not always present in the header metadata, so the first check is if the impedance section exists. If so, the impedance unit and measure time are parsed from the section heading. The heading is followed by a list of channels with their impedances. The lines are parsed for the channel name until an empty line is found, as shown in Listing 5.1, where the parsed information is stored to the already existing

```
impedance = next(
  (item for item in settings if item.startswith('Impedance')), None)
if impedance:
  idx = settings.index(impedance)
  ...
  for setting in settings[idx + 1:]:
    if re.match(r'\w+:', setting):
      ch_impedance = setting.split()
      ch_name = channel_impedance[0].rstrip(':')
      channel = next(
        (item for item in bv_info['chs'] if item['ch_name'] == ch_name),
        None)
      imp_as_number = re.findall(r"[-+]?\d*\.\d+|\d+", ch_impedance[1])
      channel.update({
        'imp': float(imp_as_number[0] if imp_as_number else 0),
        'imp_unit': impedance_unit,
        'imp_meas_time': datetime.strptime(impedance_time, "%H:%M:%S")
      })
```

Listing 5.1: Parsing the channel impedances.

channel info in the raw BrainVision parsed data. The impedance ranges for different types of channels are parsed and appended to this data structure as well.

A second metadata section that is missing in the BrainVision info parsed from the MNE library is the segmentation section. This section is present for the averaged recordings. When this section is present, some additional key-value pairs are also added in the "Common infos" section. These are parsed using the python `ConfigParser`, as are the other key-value in this section. The rest of the segmentation metadata in the separate section have to be parsed manually. Some of these metadata are simple key-value pairs, only with different delimiters. This type of metadata is parsed and processed using a key-value parsing method with a specifiable delimiter, shown in Listing 5.2. Values are stored as arrays, where a comma can separate individual elements. In some cases, the value for a single key can be represented across multiple lines. This is solved by checking for a double tabulator instead of a single one. For that case, the line is stored in the last saved key.

The enhancements of the MNE library in the form of the segmentation and impedance sections parsing were considered as changes with a wider use case than just for this work. A pull request to the MNE Github repository[1] was made that integrates these changes to the library. If these changes get accepted, they could replace the current methods from the conversion tool.

The MNE library uses "FIFF constants" to represent some static information like the channel units. These constants are converted to a human-readable value. In the case of the unit constant, the `FIFF_UNIT_V` constant is converted to a "V" string. The rest of the BrainVision metadata is used as the MNE library parsed them.

Once all the metadata are prepared, they are converted into the NIX container using the section and property entities. The MNE internal structure is mapped onto these entities to fit the odML terminologies where possible. The mapping is stored in a JSON file consisting of the attribute names, combined with a parent attribute if there is any. These two are delimited by a double underscore to avoid collisions with the attribute names. The mapping entity tells the converter how to store the value and in which NIX property or section. Most of the BrainVision metadata are adjustable parameters of the hardware that may change with each recording or dataset. The odML terminology offers the "`HardwareSettings`" section for this type of metadata. This section will contain hardware sections describing the Brain-

[1]Available at https://github.com/mne-tools/mne-python, Pull Request #7771

```
for setting in segmentation_settings[idx + 2:]:
  if re.match(r'(\t)?\w', setting):
    setting = setting.split(delimiter)
    key_name = setting[0].strip()
    key_values = setting[1].strip().split(',')
    values[key_name] = key_values if key_values[0] != '' else []
  elif re.match(r'\t\t\w', setting) and key_name is not None:
    key_values = setting.strip().split(',')
    for val in key_values:
      if val != '':
        values[key_name].append(val)
  else:
    break
```

Listing 5.2: Parsing the segmentation section.

Vision recording settings. However, any metadata that are already stored in the odML metadata will be stored in the "`HardwareProperties`" section from the odML terminologies that is meant to store the fixed properties of used hardware like description, manufacturer, and other.

5.2.2 odML converter

The conversion of the recording session metadata stored in the odML format is implemented into the `odmlconverter` module. The odML core library [58] is used to read the old metadata. This library has a class for converting the older document version into the most recent version. This step is necessary as the reader from the library only accepts documents in the newest version. Before saving the document to NIX, it is mapped to the odML terminologies using the `OdmlMapper` class.

The mapping of an odML document is stored in the form of a JSON file. This file contains a dictionary of individual mapping entities for the original odML sections and their properties. Each entity is represented by the type and name of the original odML section or property, delimited by a double underscore. The mapping of an entity is in the `corresponding_term` attribute. This attribute consists of the name and type for the converted metadata, and the parent section where the mapped section or property will be placed.

Apart from the `corresponding_term` attribute, some other options for the mapping are available as well:

- `use_as_reference` - this is an attribute for a special use case for a property mapping, where the property value will be mapped to a specified section's reference attribute, instead of creating a new property.

- `add_prop` - an optional attribute for a section or property mapping entity, which will put additional properties to the mapped section. This is an array of properties composed of name, type, and value.

- `section_reference` - attribute for a section mapping entity, which places an id reference of the created section to a specified property.

The entity also contains additional attributes solely for an informative purpose, e.g., the original type or an example of the values of a property. A typical use case of a mapping entity is in Listing 5.3. In this Listing, a Person section from the original odML document will be mapped into a section "Experimenter" of type "person". This section will be placed in the "Experimenters" section. The original odML section has a property named "givenname", which will be mapped onto a new property "FirstName" of the "Experimenter" section.

The implemented package provides a default mapping file that was generated from all the available EEGBase datasets. The mapping entities were manually modified to map the sections and properties to the odML terminologies. A new mapping file can be generated using the `mapping_generator` script from the `odmlconverter` module. This script takes an input path as

```
"person__person": {
  "corresponding_term": {
    "type": "person",
    "name": "Experimenter",
    "parent_sections": "collection__Experimenters"
  },
  "props": {
    "givenname": {
    "type": "string",
    "example_value": "['Jan']",
    "corresponding_term": {
      "name": "FirstName",
      "type": "string",
      "parent_sections":"collection__Experimenters___person__Experimenter"
    }
  }
}
```

Listing 5.3: A mapping entity for a odML section and its property.

an argument. This path and all its subfolders are searched for XML files. The contents are parsed, and the sections and their properties are aggregated to create the mapping file for these documents. A custom mapping can be passed as an optional argument to be used instead of the default one.

The `OdmlMapper` class takes the provided mapping file and starts iterating through the original odML document. The mapped sections and properties are saved into a new odML document. The script is working with this new document in memory; no new file with the remapped odML document is written to the file system. The mapper supports data type conversion for the property mapping to basic types of Integer, Float, String, and DateTime. For the DateTime data type, a format attribute is required in the `corresponding_term` of the mapping entity. This format attribute is used for the parsing of the date from a string value. Once all the sections and properties are mapped, this newly created odML document replaces the original one to be used in the conversion.

A tool for writing the contents of an odML document into a new NIX file and vice versa already exists in the form of the nixodmlconverter python package [59]. The recursive function that converts a passed odML section and its properties to NIX is taken from this tool. The root section of the session metadata in the NIX file is created separately, where a proper section name and a link to the odML terminologies repository is set. The odML metadata are placed into this section using the recursive function.

During the implementation, a bug was found in the core Nixpy library which the nixodmlconverter package uses, unable to convert any float properties due to a bad comparison of data types. This was reported to the package's repository[2] with a described cause of this problem and a proposed solution. The solution to this problem suggested by the developers was successfully tested on the implemented conversion tool and is waiting to be included in the next official release.

5.3 Implementation summary

The MNE library is used to parse the BrainVision data and metadata. Some of the metadata were not initially parsed with the library. Custom functions were implemented to parse the segmentation and impedance metadata. These changes were proposed as pull requests into the library GitHub repository. All these data joined together are then converted into NIX with NIX-MNE conversion tool. The functions from this library were modified

[2]Available at `https://github.com/G-Node/nixpy`, Issue #463

as well to use mapping of the metadata to the odML terminologies. These modifications were specific to this use-case, so they were not implemented directly into the tool.

With the BrainVision data covered, the odML metadata document was transformed to use the terminologies using an implemented mapper. The user can use a separate python script to generate the mapping entities and use them to transform the odML metadata instead of the default mapping to the odML terminologies. This transformed document is then converted to the NIX file to join the converted BrainVision data and metadata. The nixodmlconverter tool is used for this conversion. The conversion would fail for any float values in the document due to a bug in the core NIXIO library. A fix was proposed as a pull request in the NIXIO GitHub repository, with the request resolved and the changes merged into master for the next release.

6 GIN repository

The main idea of storing the converted dataset is to use the G-Node's hosted GIN repository. In this repository, we can utilize the otherwise paid DOI dataset identification, which is a crucial part of satisfying the FAIR principles. However, as the hosted repository does not yet include the gin-proc and drone microservice for extending the data workflows, a docker image with implemented pipelines is included as part of the final solution as well. The docker image will serve as a sandbox environment in the meantime.

A new user account was created on the hosted GIN repository using the provided registration form. Using this account, a new repository is created for each type of the converted experiments. The license of the repository is set upon the creation. If the licensing information is present in the converted datasets, the same license is used for the GIN repository. Otherwise, the Creative Commons Attribution 4.0 license is selected.

The uploading of the dataset is done using the GIN client. After logging in to the client, the repository is initialized in the local directory using the `gin get` command. After it is initialized, the converted datasets can be added into this directory, and a local gin commit is created, adding all the files and giving a descriptive message. The local commit with the files is then uploaded to the remote GIN repository with `gin upload`, which does the actual upload of the files into the remote repository. The process of uploading the datasets to the remote GIN repository is shown in Listing 6.1.

```
gin login
gin get jsedivy/Event-related_potential_datasets
gin commit --message "Initial dataset upload" .
gin upload
```

Listing 6.1: The process of uploading a dataset.

The process of obtaining a DOI starts with adding the `datacite.yml` file into the root of the GIN repository. This file contains information for publication of the data. The datacite file should hold information about the authors of the datasets, title and a description of the experiment, keywords and licensing, as seen in Listing 6.2. Other optional attributes are available as well. If these pieces of information are known from the experiment's datasets, they are copied to the datacite file. Alternatively, if an experiment has some additional information published in the EEGbase portal, or in the articles at the neuroinformatics research group website [60], these are added to the datacite file as well. The full description of the `datacite.yml` file is at [34]. Once the `datacite.yml` file is created for the repository, and the repository is set to public, a request for assigning a DOI to the repository is made using the DOIfy button in the GIN web interface.

```
authors:
- firstname: "John"
  lastname: "Doe"
  affiliation: "Faculty of Applied Sciences, University of West Bohemia"

title: "Event-related potential datasets based on three-stimulus-paradigm"

description: |
  Event-related potential datasets based on three-stimulus-paradigm
  The datasets have been converted into NIX from the original experiment
  shared in the EEGBase portal

keywords:
 - Neuroscience
 - Electrophysiology
 - ERP
```

Listing 6.2: An example of a datacite file.

6.1 Docker image

For a local instance of the GIN repository, a docker image with all the necessary services is provided. It is implemented using `docker-compose` that includes all the necessary settings and offers a ready-to-use repository. The variable settings of all the services are placed in the `.env` file. Each service runs in its own docker container. This will allow for easier upgrading or replacement of the individual services. The complete image consists of these services:

- `gin-service` - The main container that runs the `gnode/gin-web` docker image[1]. By default, the web interface is made accessible at the address `http://172.19.0.2:3000`.
- `db-postgres` - The container for the GIN relational database. A PostgreSQL database is used from the official Docker images[2].

[1]Available from `https://hub.docker.com/r/gnode/gin-web`
[2]Available from `https://hub.docker.com/_/postgres`

- `gin-index` - The `gnode/gin-dex` docker image[3] for indexing the stored datasets.
- `db-elasticsearch-node` - The ElasticSearch database image[4] for the indexing service.
- `drone` - The Drone docker image[5] for the pipeline job management. Accessible at the address `http://172.19.0.3` by default.
- `drone-runner` - A runner image[6] for the drone service. This service runs the registered pipeline jobs.

The individual service containers are connected using a docker bridge network. The network does have a configured subnet, and each container is assigned a static IP address from this subnet. This is done so that each container has the same IP address on each container start-up and address links between containers can be specified, e.g., connecting the GIN web service to the PostgreSQL database.

6.1.1 GIN web interface

The GIN repository web interface runs in the `gin-service` container. A settings file `app.ini` is used to change the default repository settings. This includes the main server settings like the server's HTTP address and port, database connection settings, repository upload limits, etc. The settings file is in the root directory of the `docker-compose`, and it is mounted as a docker volume into the GIN container's configuration path. A second docker volume is mounted to the directory where the GIN repository stores its data. This is to preserve the data on the host machine on the container restart or when the container is simply not running.

A PostgreSQL database is used for the GIN's relational data, as it is recommended in the GIN wiki [61]. The definition of the container for the PostgreSQL container is shown in Listing 6.3. The database schema for the GIN repository is specified in the container's environment variable `POSTGRES_DB` that automatically creates the schema on start-up. The values are taken from the `.env` variables. However, only one database can be initialized this way. As the Drone service requires its own database schema, the initialization of this schema needs to be handled separately. PostgreSQL

[3]Available from `https://hub.docker.com/r/gnode/gin-dex`
[4]Available from `docker.elastic.co/elasticsearch/elasticsearch:6.8.8`
[5]Available from `https://hub.docker.com/r/drone/drone`
[6]Available from `https://hub.docker.com/r/drone/drone-runner-docker`

```
db-postgres:
  container_name: db-postgres
  image: postgres
  restart: unless-stopped
  environment:
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_DB=${POSTGRES_GIN_DB}
    - POSTGRES_DRONE_DB=${POSTGRES_DRONE_DB}
  volumes:
    - ./docker/pg-init-scripts:/docker-entrypoint-initdb.d
    - ./db/postgresql/data:/var/lib/postgresql/data
  networks:
    gin:
      ipv4_address: 172.19.0.5
```

Listing 6.3: The definition of the PostgreSQL container.

has a init directory `docker-entrypoint-initdb.d` which it uses on service start-up and runs any SQL scripts in this folder. The docker image mounts the `pg-init-scripts` as a volume to this directory containing a SQL script that creates the database schema for Drone as well. The PostgreSQL data folder is mounted as a volume to the host machine to create persistent storage.

The `gin-index` service takes care of indexing the data from GIN's repositories into the ElasticSearch database. The indexed data are used for the dataset searching functions in the GIN web interface. The `gin-index` and `gin-service` share an encryption key that is used for verification. The `gin-service` defines its key in the `app.ini` settings file, the indexing service has the key passed as an environment variable from the docker container. These keys must be matching for the indexing to work. The indexing service reads data from the shared docker volume of the GIN web service. The ElasticSearch service is split into two containers, where each container contains one ElasticSearch node. The nodes will share the same cluster and use the same docker volume to store its data. The ElasticSearch image itself will take care of connecting the nodes and communication between them.

The GIN web service provides a first-time setup where the user can define the database connection, some basic repository settings and create a first user. This option is disabled and the setup is rather done automatically using the properties set in the `app.ini` settings file. Although the image is created using docker-compose so the image could be started using the docker-compose command, a start-up bash script is created that should be used instead. This script takes care of creating the required directories for the docker volumes if they are not present, as the image start-up fails otherwise. The script also creates a default admin user for the GIN web service. The ElasticSearch database uses a virtual memory `mmapfs` directory to store its indices and the service checks for the limit, which is set too low in most operating systems [62]. The script checks the map count limit beforehand and increases it to a minimum required amount for the ElasticSearch service to start if it is too low.

6.1.2 Pipelines

The pipelines in the GIN's repositories are implemented using the Drone service. It is connected to the GIN container and the PostgreSQL database container using the DRONE environment variables. A Drone user is created for the default admin GIN account. The access is limited to this account only using the `DRONE_USER_FILTER` environment variable. The Drone service is the web-based management of the pipelines and registering the pipeline jobs. The actual pipeline jobs are executed in the Drone runner service. A network is specified for these two services so they can communicate with each other. In addition, both services have the matching `DRONE_RPC_SECRET` environment variable defined for authentication.

```
steps:
- name: init-clone
  image: falconshock/gin-proc
  volumes:
    - name: repo
      path: /repo
  environment:
    SSH_KEY:
      from_secret: DRONE_PRIVATE_SSH_KEY
  commands:
    - eval $(ssh-agent -s)
    - mkdir -p /root/.ssh
    - echo "$SSH_KEY" > /root/.ssh/id_rsa
    - chmod 0600 /root/.ssh/id_rsa
    - ssh-add /root/.ssh/id_rsa
    - ssh-keyscan -t rsa "$DRONE_GOGS_SERVER" > /root/.ssh/authorized_keys
    - if [ -d "$DRONE_REPO_NAME" ]; then
        cd "$DRONE_REPO_NAME"/; git fetch --all;
        git checkout -f "$DRONE_COMMIT";
      else git clone "$DRONE_GIT_SSH_URL"; cd "$DRONE_REPO_NAME"/; fi
```

Listing 6.4: A shortened clone step of a pipeline.

The jobs are specified with the `.drone.yml` file in the root of a repository. An example of this file is provided together with the docker image. The first step of the pipeline job is cloning of the GIN repository. As GIN is built on Gogs [63], a git service, the cloning is done using normal git commands. The authentication for the cloning is done via ssh. The Drone private key is set in the Drone web interface as the repository secret value. This value is then accessed during the job execution and a new ssh key is created and added to the ssh-agent. The GIN repository owner has to have a public key to this ssh key set in the GIN account settings. The following steps of the job can execute any user-specified commands. The repository is cloned into a volume, so any following step that mounts this volume can access the cloned repository and its contents.

7 Evaluation

This chapter evaluates the implemented python conversion tool and the datasets that are the result of the conversion from the EEGBase into the NIX standard. Static code analysis is performed on the source code of the python tool, and a set of unit tests is written to cover the core functionality. The created NIX structure is validated, and a FAIR principles assessment is made on the converted datasets combined with their upload to the GIN repository.

7.1 Conversion tool

For the static code analysis of the conversion tool, a couple of analysis tools are used. First, a check with the Flake8 [64], a tool for style guide enforcement, is run. No additional plugins are used and the only setting changed from the default is the maximum line length, set to 120 characters. The tool is run on any python file in the `nixconverter` package and reports a total of 0 violations found, checking 12 files overall.

The second analysis tool used is the Pylint tool [65]. The calling of the protected function from the MNE library for parsing the header tool is set to be ignored for this check. For the `BrainVisionConverter`'s recursive method `create_bv_metadata_tree` multiple checks are disabled, including too many nested blocks and branches, as altering this method to fit the Pylint requirements seems counterproductive. The tool is further run with a disabled check for a maximum line length of 80 characters, as this would collide with the Flake8 check. The check for a maximum of 7 local variables of a class instance is also disabled. The analysis with these settings results in no errors or warnings with the code rated at a full 10 out of 10.

Unit Tests for the conversion tool are implemented with the `unittest` framework from the standard Python library. The tests are placed in the tests package in the root directory and split into multiple files by the module they are testing:

- `test_brainvision_converter.py` - Test cases for the methods parsing the additional BrainVision header metadata.
- `test_converter_utils.py` - Test cases covering the utilities module functions and classes, i.e., functions for the path parsing or the correct initialization of the `HeaderProperties` and its methods.

- `test_eegbase_nix_converter.py` - These tests cover the parsing and correct initialization of the passed arguments. In addition, a test case for the creation of a NIX file from input dataset is included in these tests. The testing data from `tests/data/eegbase` are used as input, and the actual contents of the generated NIX file are checked.
- `test_odml_mapper.py` - Tests covering the mapping of an odML file from a mapping file. Included are tests for retrieving and creating sections and properties, correct data type conversions, or the result from a complete odML mapping. The optional map entity fields are tested as well, e.g., creating additional properties, using references or linking sections.
- `test_odml_mapping_generator.py` - Test cases for the mapping generator. These cover the correct format of the parent section strings.

Using the coverage tool [66] and its reporting option, the total code coverage of the whole `nixconverter` package is computed at 94%. The tool monitors the program during tests and notes which parts of code have been executed.

7.2 Converted datasets

The conversion tool was executed on the datasets from all the seven experiments provided from the EEGbase portal. The NIX Python library includes a python script for the validation of a NIX file. A bash script `validate.sh` (see Listing 7.1) is bundled with the implemented conversion tool that finds any files with the NIX file extension in a given path, executes the python script from the NIX Python library on them, and outputs the validation results to a file. The validator returns only one type of warning in a large quantity for every single NIX file about missing a unit attribute in some section properties. However, these are properties that store values with no

```bash
echo "Running the nixio validation"
find "$dataset_path" -type f -name "*.nix" | while read line; do
  echo "Validating file $line"
  python -m nixio.cmd.validate "$line" >> "$result_path"
done
echo "Validation complete. The results are stored in $result_path"
```

Listing 7.1: The bash validation script.

units that could be used for this attribute, e.g., a subject's age, an experimenter's name etc. As the unit attribute for a property is not required, so these warnings are ignored. More importantly, the validator does not return any error for any of the converted dataset, so the conversion can be considered valid.

The dataset conversion results in an increase in the total file size. For example, the average size of a dataset from the "Car simulator - Driver attention" that consists of BrainVision and odML metadata files only, is 102,9 megabytes. The average size of a NIX file from this experiment is 187 megabytes. That is an increase of 83,3%, even though the NIX block has the compressing set to `DeflateNormal`, which is the only compressing algorithm available. However, the increase in size for the NIX files is not as severe when considering all the remaining files as well. The total size of the original experiments combined is 19,8 gigabytes, whereas the converted experiments are 26,1 gigabytes in size combined, which is an increase of 31,8%.

A repository in GIN was created for each experiment, and the converted datasets were uploaded there. The repositories can be found in the G-Node's GIN website, under the newly created user[1]. A datacite file was filled out for each experiment, and a new DOI for these repositories was requested. The DOI for each experiment can be found inside the individual repositories.

7.2.1 Conversion use-case

A representative dataset in the form of Experiment 208 of the Driver's attention with visual stimulation and audio disturbance experiment was selected as the use-case of the conversion. This dataset was selected as it contains multiple recordings, as well as additional scenario data to show the full conversion.

The original dataset consists of a raw data recording and two averaged recordings (target and non-target), odML metadata and scenario files. See the full structure of this dataset in Listing 7.2. The conversion tool was executed on the folder of the experiment, using the default options and the default mapping to the odML terminologies. The conversion results in a folder that contains the NIX file with data from all the BrainVision files from the original data folder and the odML `metadata.xml` file. The scenario files are copied unchanged, as these cannot be stored in the NIX file. The structure of the dataset after the conversion is in Listing 7.3.

[1]Repositories are available at `https://gin.g-node.org/jsedivy`

```
/Experiment_208
|- /data
| |- LED_26_3_2014_0004.eeg
| |- LED_26_3_2014_0004.vhdr
| |- LED_26_3_2014_0004.vmrk
| |- LED_26_3_2014_0004-non_target.avg
| |- LED_26_3_2014_0004-non_target.vhdr
| |- LED_26_3_2014_0004-non_target.vmrk
| |- LED_26_3_2014_0004-target.avg
| |- LED_26_3_2014_0004-target.vhdr
| |- LED_26_3_2014_0004-target.vmrk
|- /Scenario
| |- 12 words.txt
| |- LED_workspace.rwksp
|- metadata.xml
```

Listing 7.2: The structure of a dataset before the conversion.

The NIX file contains three groups, each storing all the data from the original files:

- `LED_26_3_2014_0004` with entity type of `nix.data.eeg`,
- `LED_26_3_2014_0004-non_target` with entity type of `nix.data.avg`,
- `LED_26_3_2014_0004-target` with entity type of `nix.data.avg`.

The shape and form of the raw data from the `.eeg` and `.avg` files stays the same, stored as `data_arrays` in the respective groups. The marker data from the `.vmrk` files are stored as `Multi_tag` entity in the groups. The representation of the markers differs from the original to span the highlighted area across all the channels in the raw data `data_array`. For example, a marker record `"Mk2=Stimulus,S 1,21,0,0"` in the original marker file translates to the following attributes in the `Multi_tag` entity:

- `Extents[1] = [16, 0]`, where 16 is the last channel and 0 is the extent of the marker from the original record.
- `Positions[1] = [0, 0.021]`, where 0 is the first channel and 0.021 is the original position.
- `References` contains the id reference to the `data_array` with the raw data in the group.

The label for the index of 1 in the dimensions in both extents and positions has the value of `"Mk2=Stimulus,S 1"`.

```
/Experiment_208
|- /Scenario
| |- 12 words.txt
| |- LED_workspace.rwksp
|- Experiment_208.nix
```

Listing 7.3: The structure of a dataset after the conversion.

For the BrainVision metadata, each group has a separate section referenced to in the group's `data_array`. All three have the sections `Recording` with starting date and time of the recording, and `HardwareProperties` with amplifier setup and the channel information from the original "Amplifier setup" header metadata. For example, the amplifier section contains the `SampleRate` property, parsed from the original metadata line "Sampling Rate [Hz]: 1000". The property has a value of 1000.0 (as the odML terminology specifies this attribute as float) and the unit is stored in the properties `unit` attribute as "Hz".

The averaged metadata sections include the segmentation section as well, as they contain these metadata in the header files. The values from the original "Common infos" section are stored directly in the segmentation section, so for example the `SegmentDataPoints=1100` line is translated to a `SegmentDataPoints` property in the segmentation section. All the subcategories from the original header, e.g., Interval, Artifact Rejection, or Averaging and their values are represented as subsections with the same name in the segmentation section.

For the odML metadata, a section named "Session metadata" is created, where all the metadata mapped to the odML terminologies are stored. An example would be the section `Experiment` with the property `start-time` in the original odML file that translates to the NIX section `Recording` and the property `Start` as a result of the terminology mapping.

7.2.2 FAIR assessment

An evaluation of the satisfaction of the FAIR principles is done. This is to check if the resulted conversion into the NIX standard combined with the GIN repository satisfies the same FAIR principles that were evaluated during the analysis and selection of this solution and how these principles are met.

Findable

F1 Every entity of the NIX file has the `entity_id` set. The default id generator from NIX is used. The experiment datasets are also identified by a unique DOI in the GIN repository using their DOIfy service.

F2 Combining the metadata from the BrainVision header files and the session's odML metadata into the NIX sections, the datasets are richly described. The experiment's repository `README` file, the repository license and the datacite file used for generating the DOI in GIN can also be considered as additional metadata.

F3 The references between data and metadata are available and used to link together the raw data `data_array` and its metadata section parsed from a single BrainVision file.

F4 The GIN service serves as a searchable resource of the datasets. Using the gin indexing service, the contents of the dataset are searchable. Plus, the whole experiment can be accessed through the registered dataset and the assigned DOI.

Accessible

A1 The datasets are retrievable from the GIN repository either by their generated unique identifier in the form of the DOI which is done over the HTTPS protocol, where no authentication is required. Another option is to clone the datasets directly from the repository using the GIN client or git.

A2 At least the metadata of the GIN repository are always available even when no NIX files are available in the repository anymore. If the data from the NIX dataset would have been deleted, the metadata sections to these data would still be accessible, as they are stored in separate objects and only connected with a reference.

Interoperable

I1 The data are stored in the NIX format and can be viewed by any HDF5 file browser. G-Node offers the NixView tool designed to view the NIX files specifically. Furthermore, the data and metadata structures can be accessed and modified with the Python or C++ libraries that NIX offers.

I2 The metadata from the original session odML metadata and the BrainVision metadata are both described by the well-described odML terminologies where possible, thanks to the implemented mapping. As the conversion tool includes the option to use custom mapping, other terminologies describing the metadata are available. The data structures, e.g., the raw data arrays or the stimuli data, are described by the NIX data model itself.

I3 The metadata properties can contain references to other metadata sections or properties. Every NIX section and property entity can also have a reference attribute specified, which can be used to reference to metadata outside the scope of the NIX dataset. This is used for example in the hardware sections that have the original reference properties mapped with the `use_as_reference` option into this attribute.

Reusable

R1 Mapping the original metadata to the odML terminologies ensures that the converted dataset is richly described with a plurality of accurate and relevant attributes. The metadata are further extended by the information stored in the experiment's repository in the form of the datacite file.

- R1.1. - Some of the original datasets included a license in the form of a pdf file. These are copied to the converted NIX datasets as well. Every datasets license is further specified in the root of the GIN repository using the `LICENSE` file, that is generated when creating the repository.
- R1.2. - The datasets include metadata that serve as detailed data provenance, e.g., institution name or author of the experiment. These are stored in the datacite file for the whole repository as well.
- R1.3. - The usage of NIX and the odML terminologies should assure that the (meta)data meet domain-relevant community standards.

The overview of this assessment is in Table 7.1. The results show that every principle was met as originally expected from the NIX data standard analysis. The converted datasets uploaded to the hosted GIN repository comply well to the FAIR principles.

Findable				Accessible		Interoperable			Reusable
F1	**F2**	**F3**	**F4**	**A1**	**A2**	**I1**	**I2**	**I3**	**R1**
Yes	Yes	Partially	Yes	Yes (A1.1 - Yes A1.2 - Yes)	Yes	Yes	Yes	Yes	Yes (R1.1 - Yes R1.2 - Yes R1.3 - Yes)

Table 7.1: FAIR principles assessment for the implemented solution.

7.3 Tool deployment

Both the implemented conversion tool and the docker image for the GIN repository are publicly shared with the community through GitLab repositories[2,3]. Tags are used to highlight specific versions of the tool and the docker image. The repository for the conversion tool has continuous integration (CI) configured. This CI contains steps for the static code analysis tools and the unit tests, as seen in Listing 7.4. The pipeline status and the unit test coverage percentage are shown as badges on the repositories homepage. When cloning the tool from this repository, a `requirements.txt` is provided with a list of dependencies for the tool. This file is used when installing the dependencies in the pipeline job as well.

Furthermore, the conversion tool is distributed through the Python Package Index (PyPI)[4] repository. Using PyPI and the pip installer[5] included in the modern versions of python by default, users can install the conversion tool with all of its dependencies automatically. The released versions in PyPI match the version tags in the GitLab repository.

```
flake8:
  stage: Code Analysis
  script:
    - flake8 --max-line-length=120 nixconverter/*.py
unittest:
  stage: Tests
  script:
    - coverage run -m unittest discover -b
```

Listing 7.4: A segment of the CI pipeline steps.

[2]EEGbase NIX converter: available at https://gitlab.com/honza.seda/eegbase-nix-converter

[3]GIN Docker image: available at https://gitlab.com/honza.seda/gin-docker-image

[4]`https://pypi.org/`

[5]https://pip.pypa.io/en/stable/

8 Conclusion

The analysis of the current solution for storing the electrophysiological experiments in the EEGbase portal confirmed that it does not comply well to the FAIR principles. New means for storing the original data in the form of available data standards were examined, where each standard was evaluated on the satisfaction of the individual FAIR principles. Based on this evaluation with consideration of the difficulty of a possible conversion into each standard, the NIX data standard was selected as the most viable option of the conversion.

A python conversion tool was implemented using some of the existing open-source libraries that convert individual parts of the dataset and modifying them to fit the purpose of this thesis, namely the FAIR principles requirements. This includes mapping of the metadata to the odML terminologies. The mapper is designed to allow its use on other, user's specified terminologies, and metadata different to those stored in EEGbase as well. Fixes for found bugs and some of the enhancements made to the open-source libraries that were considered useful for general use have been proposed directly into the library repositories. The result is a production of valid NIX files converting the original data and metadata. A downside of the converted datasets is an almost double increase in file size compared to the original combination of BrainVision and odML formats, despite using the data block compressing algorithm provided by NIX.

The G-Node's GIN repository was selected as the most convenient way to store the newly selected NIX data standard, replacing the EEGbase portal. The version of the repository hosted at G-Node's servers was chosen as the primary storage of the data, as the offered DOI service, which is paid otherwise, greatly supports the findability principle. However, the pipeline service is not included yet. This was the reason for the implementation of a docker image that runs the GIN repository with the pipeline service as well, which can be used as a sandbox environment for the pipelines.

Overall, the implemented solution meets all of the FAIR principles. All of the provided experiment's datasets are successfully converted, stored in the G-Node's hosted repository, and identified by DOI. The implemented solution could be used to replace the current EEGbase portal. Some of the changes proposed to the hosted GIN, like the pipeline implementation or indexing of NIX file contents would further improve the repository and could be implemented to the local docker image as well.

List of abbreviations

API	Application programming interface
BIDS	Brain Imaging Data Structure
CI	Continuous Integration
DOI	Digital Object Identifier
EEG	Electroencephalography / Electroencephalogram
ERP	Event-related potential
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
NWB:N	Neurodata Without Borders: Neurophysiolog
odML	Open Metadata Markup Language
OWL	Web Ontology Language
PID	persistent identifier
RDF	Resource Description Framework
REST	Representational state transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell

Bibliography

[1] *FAIR Principles* [online]. GO FAIR, 2019. [Accessed 2019/10/12]. FAIR Principles. Available at: `https://www.go-fair.org/fair-principles/`.

[2] Wilkinson, M. D. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*. 2016, vol. 3. ISSN 2052-4463. doi: 10.1038/sdata.2016.18. Available at: `https://doi.org/10.1038/sdata.2016.18`.

[3] *The FAIR data principles* [online]. The Australian National Data Service (ANDS), 2019. [Accessed 2019/11/02]. Working with data. Available at: `https://www.ands.org.au/working-with-data/fairdata`.

[4] *The Internet of FAIR Data & Services* [online]. GO FAIR, 2020. [Accessed 2020/05/08]. Available at: `https://www.go-fair.org/resources/internet-fair-data-services/`.

[5] *European Open Science Cloud (EOSC)* [online]. European Commission, 2020. [Accessed 2020/05/08]. Available at: `https://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud`.

[6] *GO FAIR Initiative* [online]. GO FAIR. [Accessed 2020/04/05]. FAIR Principles. Available at: `https://www.go-fair.org/go-fair-initiative/`.

[7] *FAIRification Process* [online]. GO FAIR. [Accessed 2020/04/04]. FAIR Principles. Available at: `https://www.go-fair.org/fair-principles/fairification-process/`.

[8] Mouček, R. et al. Software and hardware infrastructure for research in electrophysiology. *Frontiers in Neuroinformatics*. 2014, vol. 8. ISSN 1662-5196. doi: 10.3389/fninf.2014.00020. Available at: `https://www.frontiersin.org/article/10.3389/fninf.2014.00020`.

[9] Ježek, P. et al. Model of Software and Hardware Infrastructure for Electrophysiology. *HEALTHINF*. January 2013.

[10] *PostgreSQL: The World's Most Advanced Open Source Relational Database* [online]. The PostgreSQL Global Development Group, 2020. [Accessed 2020/05/08]. Available at: `https://www.postgresql.org/`.

[11] *Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic* [online]. Elasticsearch B.V., 2020. [Accessed 2020/05/08]. Available at: `https://www.elastic.co/products/elasticsearch`.

[12] *BrainVision Core Data Format 1.0* [online]. Brain Products GmbH, 2019. [Accessed 2019/11/24]. Available at: `https://www.brainproducts.com/productdetails.php?id=21&tab=5`.

[13] Moučeк, R. et al. Event-related potential data from a guess the number brain-computer interface experiment on school children. *Scientific Data*. March 2017. doi: 10.1038/sdata.2016.121.

[14] Grewe, J. – Wachtler, T. – Benda, J. A bottom-up approach to data annotation in neurophysiology. *Frontiers in neuroinformatics*. 2011, vol. 5. doi: 10.3389/fninf.2011.00016.

[15] Brůha, P. et al. The Ontology for Experimental Neurophysiology: a first step toward semantic annotations of neurophysiology data and metadata. *Front. Neuroinform. Conference*. July 2013. doi: 10.3389/conf.fninf.2013.09.00026.

[16] Papež, V. – Moučeк, R. applying an archetype-Based approach to electroencephalography/event-related Potential experiments in the eegBase resource. *Frontiers in neuroinformatics*. 2017, vol. 11. doi: 10.3389/fninf.2017.00024.

[17] *Coordination disorder in children* [online]. NEUROINFORMATICS research group, 2015. [Accessed 2020/05/10]. Available at: `http://neuroinformatics.kiv.zcu.cz/articles/read/coordination-disorder-in-children_2015-01-20`.

[18] *Driver's attention* [online]. NEUROINFORMATICS research group, 2015. [Accessed 2020/05/10]. Available at: `http://neuroinformatics.kiv.zcu.cz/articles/read/drivers-attention_2015-01-20`.

[19] Grewe, J. *About - G-Node/nix wiki* [online]. German Neuroinformatics Node (G-Node), 2015. [Accessed 2020/02/15]. Available at: `https://github.com/G-Node/nix/wiki/About`.

[20] Grewe, J. *NIX data model* [online]. German Neuroinformatics Node (G-Node), 2019. [Accessed 2020/02/17]. Available at: `https://nixio.readthedocs.io/en/latest/data_model.html`.

[21] Sonntag, M. *Model Definition - G-Node/nix wiki* [online]. German Neuroinformatics Node (G-Node), 2017. [Accessed 2020/02/17]. Available at: `https://github.com/G-Node/nix/wiki/Model-Definition`.

[22] Grewe, J. *The Model - G-Node/nix wiki* [online]. German Neuroinformatics Node (G-Node), 2015. [Accessed 2020/02/15]. Available at: `https://github.com/G-Node/nix/wiki/The-Model`.

[23] *odML Terminologies - Data model for storing arbitrary metadata* [online]. G-Node, 2020. [Accessed 2020/05/08]. Available at: `https://terminologies.g-node.org/v1.1/terminologies.xml`.

[24] Grewe, J. *Support for standardization - NIXIO library documentation* [online]. German Neuroinformatics Node (G-Node), 2019. [Accessed 2020/02/19]. Available at: `https://nixio.readthedocs.io/en/latest/standardization.html`.

[25] Folk, M. et al. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47, 2011. doi: 10.1145/1966895.1966900.

[26] Rübel, O. et al. NWB:N 2.0: An Accessible Data Standard for Neurophysiology. *bioRxiv.* 2019. doi: 10.1101/523035. Available at: `https://www.biorxiv.org/content/early/2019/01/17/523035`.

[27] Grewe, J. *Libraries & tools - NIXIO library documentation* [online]. German Neuroinformatics Node (G-Node), 2019. [Accessed 2020/02/28]. Available at: `https://nixio.readthedocs.io/en/latest/ecosystem.html`.

[28] *GIN - Modern Research Data Management for Neuroscience* [online]. G-Node, 2020. [Accessed 2020/05/08]. Available at: `https://gin.g-node.org/`.

[29] Koutsou, A. *About GIN - G-Node GIN* [online]. German Neuroinformatics Node (G-Node), 2020. [Accessed 2020/02/28]. Available at: `https://gin.g-node.org/G-Node/Info/wiki/about`.

[30] Koutsou, A. *GIN advantages structure - G-Node GIN* [online]. German Neuroinformatics Node (G-Node), 2020. [Accessed 2020/02/28]. Available at: `https://gin.g-node.org/G-Node/Info/wiki/GIN+Advantages+Structure`.

[31] *GSoC 2020 project idea 11: Extended support for NIX file format in GIN* [online]. INCF Google Summer of Code (GSoC), 2020. [Accessed 2020/05/10]. Available at: `https://neurostars.org/t/gsoc-2020-project-idea-11-extended-support-for-nix-file-format-in-gin/5748`.

[32] Garbers, C. et al. *G-Node Projects* [online]. The German Neuroinformatics Node (G-Node), 2020. [Accessed 2020/03/07]. Available at: `https://g-node.github.io/`.

[33] Wahal, M. *Installation - gin-proc Microservice for GIN* [online]. 2020. [Accessed 2020/03/20]. Available at: `https://github.com/G-Node/gin-proc/blob/master/docs/install.md`.

[34] Koutsou, A. *Obtaining a DOI (Digital Object Identifier) - G-Node GIN* [online]. German Neuroinformatics Node (G-Node), 2020. [Accessed 2020/02/28]. Available at: `https://gin.g-node.org/G-Node/Info/wiki/DOIfile`.

[35] Koutsou, A. *Licensing your data - G-Node GIN* [online]. German Neuroinformatics Node (G-Node), 2020. [Accessed 2020/03/01]. Available at: `https://gin.g-node.org/G-Node/Info/wiki/Licensing`.

[36] Gorgolewski, K. J. et al. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*. 2016, vol. 3, no. 1. doi: 10.1038/sdata.2016.44.

[37] Pernet, C. R. et al. EEG-BIDS, an extension to the brain imaging data structure for electroencephalography. *Scientific data*. 2019, vol. 6, no. 1. doi: 10.1038/s41597-019-0104-8.

[38] *Specification - Brain Imaging Data Structure v1.2.2* [online]. 2019. [Accessed 2020/03/01]. Available at: `https://bids-specification.readthedocs.io/en/stable/`.

[39] Yarkoni, T. et al. PyBIDS: Python tools for BIDS datasets. *Journal of Open Source Software*. 2019, vol. 4, no. 40. doi: 10.21105/joss.01294.

[40] *BIDS for MATLAB / Octave* [online]. 2019. [Accessed 2020/03/07]. Available at: `https://github.com/bids-standard/bids-matlab`.

[41] Gorgolewski, K. et al. OpenNeuro — a free online platform for sharing and analysis of neuroimaging data. *Organization for human brain mapping. Vancouver, Canada*. 2017, vol. 1677, no. 2. doi: 10.7490/f1000research.1114354.1.

[42] *1. Overview - NWB Format Specification v2.2.1 documentation* [online]. Lawrence Berkeley National Laboratory, 2020. [Accessed 2020/03/01]. Available at: `https://nwb-schema.readthedocs.io/en/stable/format_description.html`.

[43] *The NWB Software Ecosystem* [online]. Neurodata Without Borders, 2020. [Accessed 2020/03/20]. Available at: `https://www.nwb.org/nwb-software/`.

[44] *NWB Specification Language* [online]. Neurodata Without Borders, 2017. [Accessed 2020/05/08]. Available at: `https://schema-language.readthedocs.io/en/stable/`.

[45] Leach, P. – Mealling, M. – Salz, R. *A Universally Unique IDentifier (UUID) URN Namespace* [online]. The Internet Society, 2005. [Accessed 2020/05/08]. Available at: `https://tools.ietf.org/html/rfc4122`.

[46] *NWB for Python* [online]. Neurodata Without Borders, 2020. [Accessed 2020/05/08]. Available at: `https://pynwb.readthedocs.io/en/stable/`.

[47] *MatNWB* [online]. Neurodata Without Borders, 2020. [Accessed 2020/05/08]. Available at: `https://neurodatawithoutborders.github.io/matnwb/`.

[48] *Loris Governance Framework* [online]. McGill Centre for Integrative Neuroscience, 2019. [Accessed 2020/03/20]. Available at: `https://loris.ca/LORIS_Governance_1904.pdf`.

[49] Das, S. et al. The MNI data-sharing and processing ecosystem. *NeuroImage*. 2016, vol. 124. doi: 10.1016/j.neuroimage.2015.08.076.

[50] Ardestani, S. B. et al. B2share: An open escience data sharing platform. *2015 IEEE 11th International Conference on e-Science*. 2015. doi: 10.1109/eScience.2015.44.

[51] Verkerk, R. *PIDs in EUDAT* [online]. EUDAT Collaborative Data Infrastructure, 2020. [Accessed 2020/05/08]. Available at: `https://eudat.eu/services/userdoc/pids-in-eudat`.

[52] *Human Brain Project* [online]. 2017. [Accessed 2020/05/08]. Available at: `https://www.humanbrainproject.eu/`.

[53] *DATA & KNOWLEDGE - Share Data* [online]. EBRAINS, 2019. [Accessed 2020/03/22]. Available at: `https://ebrains.eu/services/data-knowledge/share-data/`.

[54] *EBRAINS Knowledge Graph - A metadata management system built for EBRAINS* [online]. EBRAINS, 2019. [Accessed 2020/05/08]. Available at: `https://kg.ebrains.eu/`.

[55] Sonntag, M. *odML - NIX metadata conversion tool* [online]. G-Node, 2020. [Accessed 2020/04/08]. Available at: `https://github.com/G-Node/nix-odML-converter`.

[56] *MNE - Open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more.* [online]. MNE Developers, 2020. [Accessed 2020/05/08]. Available at: `https://mne.tools/stable/index.html`.

[57] Koutsou, A. *NIX-MNE conversion tool* [online]. 2019. [Accessed 2020/05/08]. Available at: `https://github.com/G-Node/nix-mne`.

[58] Koutsou, A. – Grewe, J. – Sonntag, M. *odML (Open metaData Markup Language) core library* [online]. 2020. [Accessed 2020/05/08]. Available at: `https://pypi.org/project/odML/`.

[59] Koutsou, A. – Sprenger, J. – Sonntag, M. *odML - NIX metadata conversion tool* [online]. 2020. [Accessed 2020/05/08]. Available at: `https://pypi.org/project/nixodmlconverter/`.

[60] *Case studies* [online]. NEUROINFORMATICS research group, 2020. [Accessed 2020/05/10]. Available at: `http://neuroinformatics.kiv.zcu.cz/articles`.

[61] Koutsou, A. *Advanced: In-house GIN service* [online]. 2020. [Accessed 2020/05/08]. Available at: `https://gin.g-node.org/G-Node/Info/wiki/In+House`.

[62] *Virtual Memory | Elasticsearch Reference [6.8]* [online]. Elasticsearch B.V., 2020. [Accessed 2020/05/08]. Available at: `https://www.elastic.co/guide/en/elasticsearch/reference/6.8/vm-max-map-count.html`.

[63] *Gogs: A painless self-hosted Git service.* [online]. Gogs, 2020. [Accessed 2020/05/08]. Available at: `https://gogs.io/`.

[64] Stapleton Cordasco, I. *Flake8: Your Tool For Style Guide Enforcement* [online]. 2016. [Accessed 2020/05/08]. Available at: `https://flake8.pycqa.org/en/latest/index.html`.

[65] *Pylint - code analysis for Python* [online]. Logilab, 2020. [Accessed 2020/05/08]. Available at: `https://www.pylint.org/`.

[66] Batchelder, N. *Coverage.py - Coverage.py 5.1 Documentation* [online]. 2020. [Accessed 2020/05/10]. Available at: `https://coverage.readthedocs.io/en/coverage-5.1/`.

Appendices

A Converter user guide

EEGbase NIX converter

EEGbase NIX converter is a python script that converts BranVision/odML dataset to a NIX container file. Its design and the primary use is to convert data from neurophysiology experiments taken at the Faculty of applied sciences, University of West Bohemia, stored in the EEGbase portal.

Requirements

The tool was developed and tested with Python 3.8. The required third party libraries are in the requirements.txt file. You can install the dependencies using the command:

```
pip install -r requirements.txt
```

Install using pip

When installing the converter using pip, any dependencies are automatically installed as well. The package is available from pip at:

```
pip install eegbase-nix-converter
```

ATTENTION: As the suggested fix for converting odML float values in the nixio library is not yet in the official released version, the pip package with its dependencies can not convert odML float values to NIX.
To support conversion of float values, you can use the tool from sources (or the GitLab repository[1]) and install the nixio library from the master branch in its repository[2] instead of the nixio library in the requirements, where the fix is already implemented. (Clone the repository and run `pip install` on the cloned nixio folder. If older nixio version is already included in python,

[1]EEGbase NIX converter: available at `https://gitlab.com/honza.seda/eegbase-nix-converter`
[2]Nixio: available at `https://github.com/G-Node/nixpy`

it has to be uninstalled first with `pip uninstall nixio`)
For the pip package, this applies to versions 1.0.4 or lower as well. Use `pip show eegbase-nix-converter` to show the installed version.

Usage

Run the script with the following command:

```
python -m nixconverter DATASET [--output=<path>]
   [-m] [-t|--mapping=<path>] [-v] [-y]
```

The help can be printed out using the following command:

```
python -m nixconverter -h | --help
```

Alternatively, the script can be run with the proper requirements from the nixconverter module directory:

```
python eegbase_nix_converter.py DATASET [--output=<path>]
   [-m] [-t|--mapping=<path>] [-v] [-y]
```

Arguments:

DATASET - Path to the directory with the dataset. The path is expected to be relative from the script execution directory.

By default, the script expects the directory to contain a single dataset and will create a single NIX file. If you want to change this behaviour and convert multiple datasets at a time, you can do so by passing the `-multiple (-m)` option, which will interpret the `DATASET` as parent directory and expect every immediate subfolder to be a dataset folder (will run the conversion on every subfolder in the passed directory).

The tool will attempt by default to convert the odML (.xml) metadata to use the odML terminology. The mapping to the odML terminologies uses a predefined mapping JSON file. To use custom mapping, pass the path to a JSON file with the `-mapping=<path>` option. To create your own mapping file, see the Mapping generator section.

To prevent this and use the original terminologies or not add any terminologies at all, pass the `-keep-terminologies (-t)` option with the script

Options:

-v –verbose - Output detailed logging information to console

-m –multiple - Will handle the passed path as a folder containing multiple datasets in separate directories

–output=<path> - Specify the output directory for the converted files. If the path does not exist, it will be created

-t –keep-terminologies - Keep the terminologies from the original odML metadata

-mapping=<path> - Path to a JSON mapping file

-y - Always replaces already existing files

-h –help - Show help in console.

odML Mapping generator

Part of this package is the script `odml_mapping_generator.py` for generating the mapping file from the odML .xml metadata. Mapping is used for the conversion of the odML files to the NIX sections and properties.

Usage:

If the package is installed using pip, the generator can be run using the following command:

```
python -m nixconverter.odmlconverter.mapping_generator ODML OUTPUT
```

Otherwise, the script can be run directly from the package folder:

```
python nixconverter/odmlconverter/mapping_generator.py ODML OUTPUT
```

Arguments:

ODML - Folder (and all the nested subfolders) that will be searched for .xml files. If multiple .xml files are found, the mapping will merge all the sections and properties together into a single mapping file.

OUTPUT - Name of the file to output the mapping into.

Options:

-h –help - Shows help.

Mapping file

The mapping is a .JSON file containing all the sections and its properties of the odML .xml files in a folder. Sections are represented with a key in a format of `type__name`. The properties of the section are stored in the props field of the section.

NOTE: Every section must be in the root level of the JSON.

Section/Property mapping options

The mapping of a section or a property is defined in the `corresponding_term` field. By default, the generator fills the corresponding term values by the original odML. The "corresponding_term" field must consist of the "name" (Section/property name), "type" and "parent_sections" fields.

NOTE: If the "name" or "type" field is empty, the section/property will be skipped during the conversion to the NIX file.

Parent sections is a string sequence of sections.

Each section is represented by a type_Name combination (separated by a double underscore `__` character). Nested sections are separated by a triple underscore `___`. Empty "parent_sections" means the root section of the odML document.

For `parent_sections` of a property, the last section is the section where the property will be appended. If the mapped property type is of type `date/datetime`, a format field is required that specifies the format used to parse the values, e.g.:

```
"format": "%d.%m.%Y, %H:%M:%S"
```

The `corresponding_term` can also contain the `definition` element that sets the definition of a section/property in the NIX file.

Example of the odML section and property with the "corresponding_term":

```
"person__person": {
  "corresponding_term": {
    "type": "person",
    "name": "Experimenter",
    "parent_sections": "collection__Experimenters"
  },
  "props": {
    "surname": {
      "type": "string",
      "example_value": "['Doe']",
      "corresponding_term": {
        "name": "LastName",
        "type": "string",
        "parent_sections":
            "collection__Experimenters___person__Experimenter"
      }
    }
  }
}
```

The example shows the Person section being remapped to a section "Experimenters" of type "collection".

The prop surname of the original Person section will be remapped as "LastName" to a section "Experimenter" that is a child section of the previously mapped "Experimenters" section.

Apart from the "corresponding_term", Props in the mapping contain the fields "type" (type in the original odML) and "example_value" (an example of a value stored in the property). These fields are present solely for an informational purpose.

Additional options

The JSON mapping file allows for some optional options:

add_props - Can be set for either for a section or a corresponding term of a property. It can contain an array of properties (defined by name, type and value) that will be added at the place of the remapped section/property.

```
"add_props": [
  {
    "name": "Role",
    "value": "Experimenter",
    "type": "string"
  }
]
```

section__reference - This field can be specified for a section. It will put an id reference of the newly remapped section to the specified array of properties.

```
"person__person": {
  "corresponding_term": {
    "type": "person",
    "name": "Experimenter",
    "parent_sections": "collection__Experimenters"
  },
  "section_reference": [
    {
      "section": "recording__Recording",
      "prop": "Experimenter",
      "ref_type": "string",
      "definition": "Definition string"
    }
  ]
}
```

use__as__reference - An option for a property in the mapping file. If specified, the property value will be set as a reference in the specified NIX section instead of a new property. This will be used instead of the corresponding_term field.

```
"props": {
  "source-link": {
    "type": "string",
    "use_as_reference":
      "collection/hardware_properties__HardwareProps___hardware__EEG cap"
  }
}
```

B GIN docker image user guide

GIN docker image

This is a docker-compose image that runs the GIN repository together with additional services, including:

GIN-WEB (gnode/gin-web): the web ui for the repository

PostgreSQL (postgres): persistent data storage for the GIN repository

Drone (drone/drone): microservice adding pipelines into GIN repository

GIN-INDEX (gnode/gin-dex): microservice that indexes the data and commits for searching

ElasticSearch (docker.elastic.co/elasticsearch/elasticsearch:6.8.8): used to store the indexed data for GIN-INDEX

Setup

The default settings provide a ready-to-run state where no settings have to be changed.

To start the GIN image, run the `start.sh` script. Use the `stop.sh` script to stop the image. Alternatively, you can run the image with command `docker-compose up -d` in the directory containing the `docker-compose.yml` file.

The GIN web interface can be reached at an IP address `172.19.0.2:3000`, the Drone service runs at `172.19.0.3`. The default admin user and password for both is "ginadmin".

NOTE: Elasticsearch requires the kernel setting `vm.max_map_count` to be at least 262144. Make sure you have set that on the docker host before running docker-compose (the `start.sh` script checks this for You).

if You want to change some settings, You can set the appropriate parameters:

- the `app.ini` contains settings for the GIN-web service
- the `.env` file contains environment variables for the docker compose services

NOTE: Make sure that the `GIN_INDEX_KEY` in .env and `search.SEARCH_KEY` in app.ini match for the indexing service to work properly

Pipelines

The pipelines are maintained by the Drone service at `172.19.0.3`. Only the default admin user is allowed to login. A repository is set up for pipelines if it is `activated` in Drone and it's Project settings is set to `Trusted`. A secret key named `DRONE_PRIVATE_SSH_KEY` must be set in the Drone repository settings. The value of this key is a private ssh key. The public key for this ssh key must be set for the user in the GIN repository. This can be any generated ssh key, e.g., a RSA ssh key pair.

The repository in GIN has to contain a valid `.drone.yml` file in the root of the repository. An example of this file is in the `examples/.drone.yml` file.

The `init-clone` step is not to be changed, this clones the repository for the Drone runner into a shared volume, using the ssh key in the Drone's secrets. User specified commands can be put into any following steps, that the user can create. The repository data are accesible at the `$DRONE_REPO_NAME` directory by mounting the shared volume for that step:

```yaml
steps:
  - name: init-clone
    ... DO NOT CHANGE THIS STEP

  - name: user-step
    image: alpine
    volumes:
      - name: repo
        path: /repo
    commands:
      - cd "$DRONE_REPO_NAME"
```

C EEGbase experimental set-up

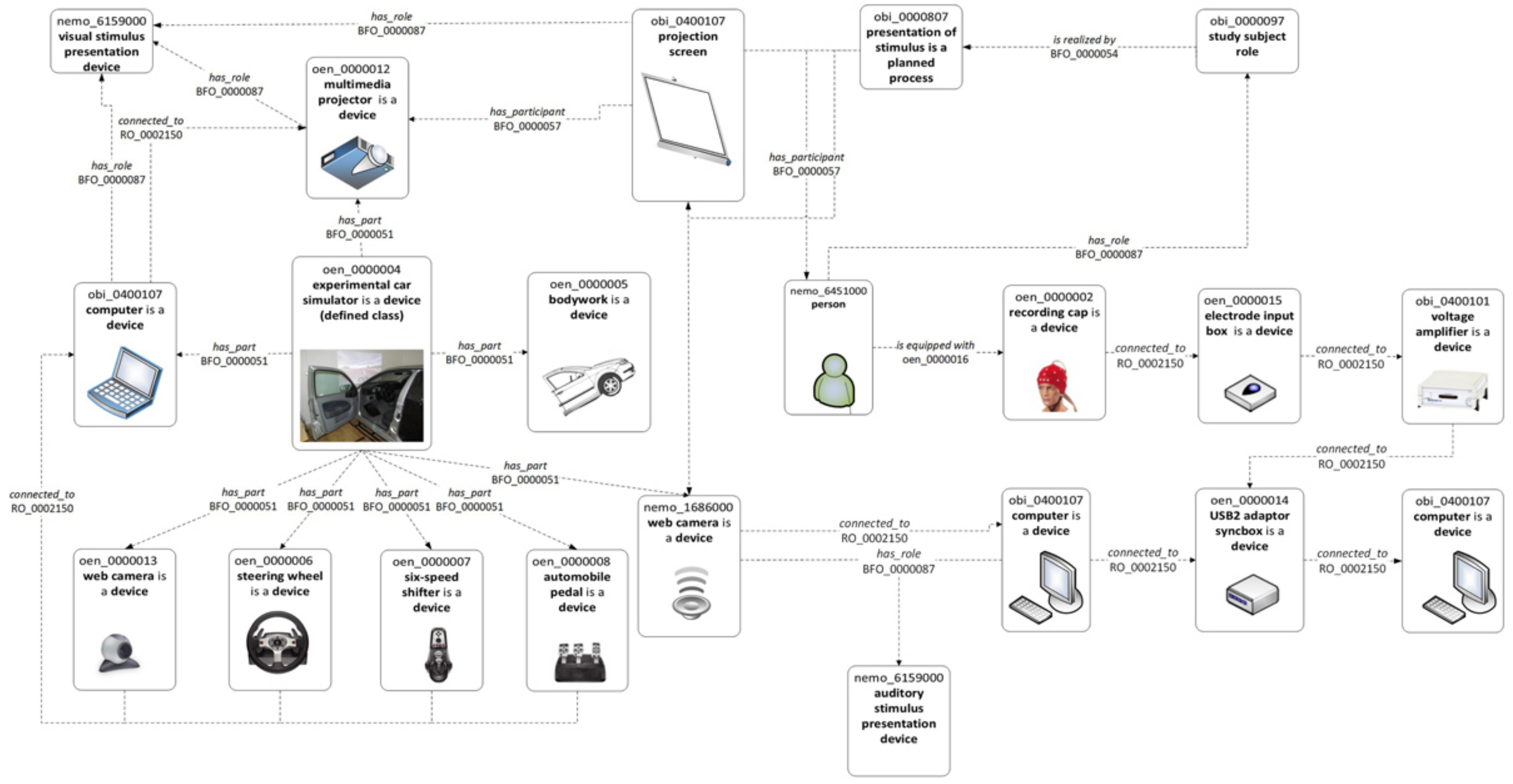

nemo_6159000 visual stimulus presentation device
has_role BFO_0000087
obi_0400107 projection screen
obi_0000807 presentation of stimulus is a planned process
is realized by BFO_0000054
obi_0000097 study subject role
has_role BFO_0000087
oen_0000012 multimedia projector is a device
has_participant BFO_0000057
connected_to RO_0002150
has_role BFO_0000087
has_participant BFO_0000057
has_part BFO_0000051
has_role BFO_0000087
oen_0000004 experimental car simulator is a device (defined class)
obi_0400107 computer is a device
has_part BFO_0000051
oen_0000005 bodywork is a device
has_part BFO_0000051
nemo_6451000 person
is equipped with oen_0000016
oen_0000002 recording cap is a device
connected_to RO_0002150
oen_0000015 electrode input box is a device
connected_to RO_0002150
obi_0400101 voltage amplifier is a device
connected_to RO_0002150
has_part BFO_0000051
connected_to RO_0002150
has_part BFO_0000051
has_part BFO_0000051
has_part BFO_0000051
has_part BFO_0000051
nemo_1686000 web camera is a device
connected_to RO_0002150
has_role BFO_0000087
obi_0400107 computer is a device
connected_to RO_0002150
oen_0000014 USB2 adaptor syncbox is a device
connected_to RO_0002150
obi_0400107 computer is a device
oen_0000013 web camera is a device
oen_0000006 steering wheel is a device
oen_0000007 six-speed shifter is a device
oen_0000008 automobile pedal is a device
nemo_6159000 auditory stimulus presentation device

Figure C.1: Description of the experimental set-up in EEGbase [8].

D EEGbase experiment dataset

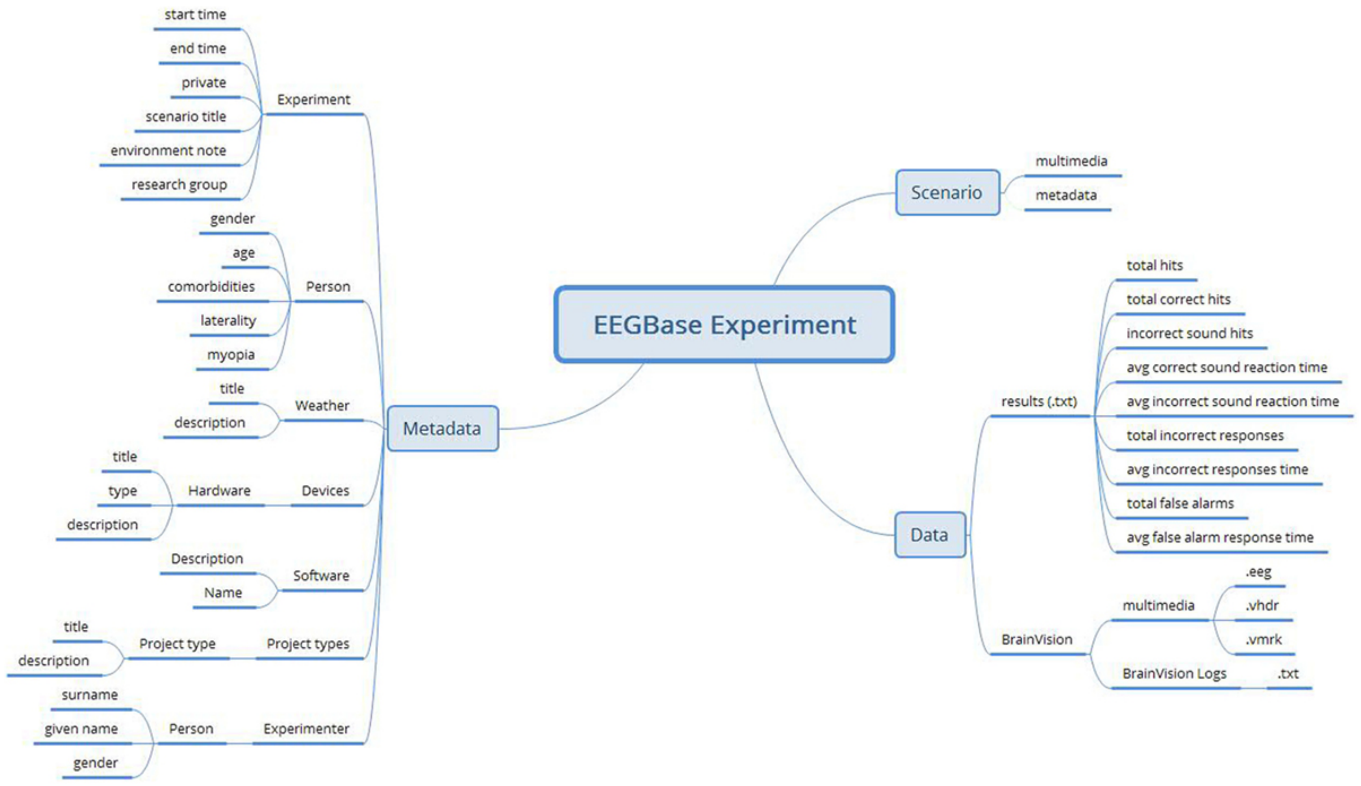


Figure D.2: The structure of an EEGbase experiment dataset [16].

E Contents of the enclosed DVD

The root directory of the enclosed DVD contains the thesis text in `thesis.pdf`, and the following subdirectories:

- `data` directory that contains example datasets that can be used for the conversion tool
- `poster` directory with the poster in the `.pdf` and `.pub` formats
- `sources` directory that contains all the source codes
  - `eegbase-nix-converter` - The python conversion tool
  - `gin` - The docker image with the local GIN web interface implementation
  - `latex` - The LATEXsource for this text