


h5RDMtoolbox - A Python Toolbox for FAIR Data Management around HDF5

Matthias Probst  ¹Balazs Pritz  ¹

1. Institute for Thermal Turbomachinery, Karlsruhe Institute of Technology, Karlsruhe.

**Date Submitted:**

2023-09-26

Licenses:This article is licensed under: **Keywords:**

Data management, HDF5, metadata, data lifecycle, Python, database

Data availability:**Software availability:**Software can be found [here](#)

Abstract. Sustainable data management is fundamental to efficient and successful scientific research. The FAIR principles (Findable, Accessible, Interoperable and Reusable) have been proven to be successful guidelines to enable comprehensible analysis, discovery and re-use. Although the topic has recently gained increasing awareness in both academia and industry, the engineering sciences in particular are lagging behind in managing the valuable asset of data. While large collaborations and research facilities have already implemented metadata strategies, smaller research groups and institutes are often missing a common strategy due to heterogeneous and rapidly changing environments as well as missing capacity or expertise. This paper presents an open source package called *h5rdmtoolbox*, written in Python. It is a general-purpose interface to HDF5 files with the aim of helping to quickly implement and maintain FAIR research data management throughout the data lifecycle, using HDF5 as the core file format. One of the key features of the toolbox is the flexible, high-level implementation of metadata standards, adaptable to the changing requirements of projects, collaborations and environments, such as experimental or computational setups. Implementation of interfaces to existing metadata schemas such as EngMeta or the CF Conventions are possible and part of the comprehensive documentation. Other benefits of the toolbox include a simplified interface to repository and database solutions.

1 Introduction

- 2 Sustainable data management is fundamental in today's data-driven world for several reasons.
- 3 The amount of acquired data storage capacity has long ceased to be the limiting factor, while the
- 4 computing power has increased greatly [1]. However, it is the ability to share data rather than
- 5 generate it that defines success [2]. Furthermore, interdisciplinary and international collaborations
- 6 have become essential in scientific research, and the main means of communication is based
- 7 on digital documents [3]. A bottleneck in data exploration and processing, and therefore the
- 8 general re-usability, is often the lack of auxiliary data, i.e. metadata. As a consequence, much
- 9 time is spent on obtaining missing information. In some cases, this may require to re-conduct
- 10 simulations and experiments. Effective data management practices hence hold the potential of
- 11 saving time and money as well as increasing the value of data at the same time.
- 12 Introducing a new data management concept can be challenging due to conflicting priorities,

13 expectations, and existing practices, as well as a lack of expertise or clear understanding of
14 the benefits. Efficient use of standards is crucial for large and interdisciplinary collaborations.
15 While those groups have developed domain-specific solutions, small research groups and PhD
16 projects face challenges due to the use of multiple file formats, individual software solutions,
17 personal preferences for storage and tools and established structures [4]. Common issues are
18 the lack of time and resources to develop and implement a comprehensive and sustainable data
19 management approach [5], which fulfills the requirements of the community and good scientific
20 practice. Therefore, flexible and manageable solutions are needed to address this issue.

21 Although the implementation of a common management system is beneficial in the long term,
22 both financially [6] and in terms of efficiency, it disrupts structures and requires time, resources
23 and cultural change. In academia, high staff turnover is an additional barrier, making it difficult
24 to establish sustainable solutions. The decay of value develops as projects progress, ultimately
25 finish and contracts expire. Consequently, the value of data will diminish over time. This issue
26 is discussed in more detail in [7], [8]. In addition, a value decay can also be observed with
27 increasing distance from the source of the data. The further away and therefore less involved
28 a potential data user is, the more information may be missing, either due to restricted access
29 or limited personal connections. Ensuring that data is preserved and being interpretable at all
30 times can be achieved by adhering to the so-called FAIR principles, which stand for Findable,
31 Accessible, Interoperable and Reusable and were first introduced in 2016 by [9]. Since their
32 publication, the principles have become the cornerstones of many scientific communities and
33 help to establish a sustainable data management [10]. Structured, highly descriptive information
34 about data, known as metadata, is an integral part of it. Metadata provides context about its
35 creation, purpose, use, processing history and the meaning of datasets. Consequently, it enables
36 data to be discoverable, interoperable and reusable.

37 This work is a contribution to assist small collaborative groups or communities and doctoral
38 researchers with achieving a FAIR research data lifecycle by using the HDF5 (Hierarchical
39 Data Format) file format. These groups are often faced with challenges such as heterogeneous
40 file formats, the absence of standards within their fields, and limited expertise and resources
41 for sustainable data management. The paper describes the scope and concepts of a Python
42 package named *h5rdmtoolbox* and how it facilitates the implementation of FAIR principles using
43 the HDF5 file format. Complementing this manuscript, an extensive online documentation is
44 provided [11], leveraging Jupyter Notebooks [12]. This documentation offers in-depth insights
45 and additional examples for immediate usage, serving as a comprehensive resource for users
46 seeking detailed information and practical guidance.

47 **1.1 Outline of the paper**

48 Firstly, the paper outlines the package's scope in comparison to existing and related works. This is
49 followed by a section stating the concepts and architecture of the toolbox, describing the applied
50 design principles and methods. Subsequently, the paper discusses concrete implementation
51 details of all sub-packages and provides illustrative examples, referencing to their relevance
52 within the research data lifecycle. Limitations of the presented package are stated before the
53 paper concludes and summarizes the presented work. An outlook is given on future developments
54 and potential enhancements.

55 2 Scope and related work

56 The primary aim of this toolbox is to offer comprehensive support throughout the lifecycle of
 57 research data (c.f. [Figure 1](#)) for small collaborative groups, communities, and doctoral researchers
 58 engaged in utilizing or contemplating the use of HDF5 files as their central file format. The file
 59 format is selected for various reasons, which are stated hereafter. A review of other file formats
 60 is beyond the scope of this work and literature should be referred to, for example [2], [8], [13].

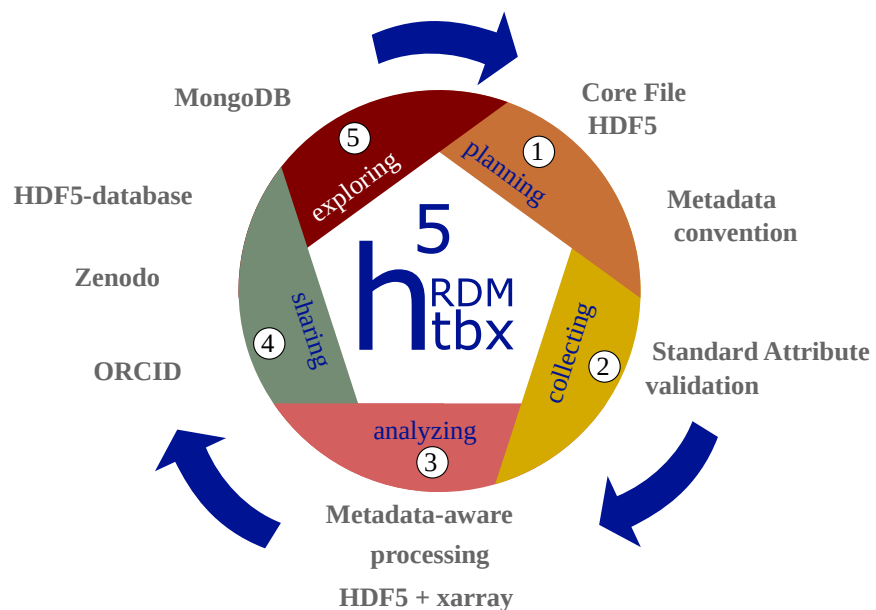


Figure 1: Illustration of the lifecycle of research data. Each phase is supported by the *h5rdmtoolbox*. It starts by selecting a file format (HDF5) and a metadata concept (1) and performing quality assurance measures during the selection and processing phase (2). Data is analyzed effectively for scientific output in the next step (3). After publication, the availability of the data should be ensured (4). (Meta)data quality finally is defined by its findability and consequently its re-usability (5) for additional analysis at later time. The respective tools and solutions provided by the toolbox are indicated by keywords around the lifecycle and explained in this work.

61 HDF5 features efficient storing of large multidimensional datasets together with metadata inde-
 62 pendent of the storage media, programming environment or operating system. The hierarchical
 63 structure of group and dataset objects (cf. [Figure 2](#)) resembles most engineering data. Attributes
 64 (key-value pairs) are means to store metadata and can be assigned to each object. The HDF5 file
 65 format is therefore regarded as self-explanatory. HDF5 finds application in numerous scientific
 66 domains, such as earth observation [14], high-energy physics [15] or fluid dynamics [16]. An
 67 in-depth presentation of the file format can be found in [17].

68 Despite all the advantages of the file format, the organization of data management around HDF5
 69 is left to the user [18]. This means that the choice of attribute names and values is not regulated by
 70 any standard. Findability, effective re-usability and automatic analysis, however, are dependent
 71 on standardization [19].

72 The necessity for designing management solutions around the HDF5 file format is therefore
 73 evident. While existing solutions, such as proposed in [8], [14]–[16], [20], [21] address this need,

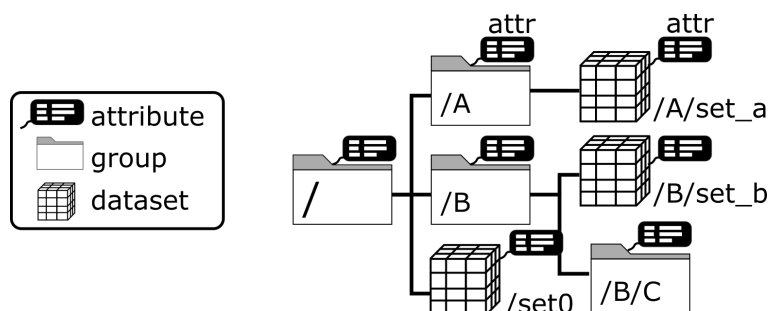


Figure 2: Illustration of the hierarchical structure of an HDF file. The internal file structure is organized like a file storage system, where folders are represented by the HDF group objects and files by HDF dataset objects. Both objects can be associated with attributes, which provide the metadata in order to make the objects interpretable.

74 they are often domain-specific, primarily focused on efficiently meeting the demands of specific
 75 communities rather than providing a generalized framework applicable to diverse problems. For
 76 example, formats like Nexus [15] or Photon-HDF5 [21] prescribe specific group and dataset
 77 organizations and metadata usage tailored to their respective data sources, such as neutron and
 78 X-ray data and molecule spectroscopy experiments, respectively. Other libraries like Zarr [20]
 79 address challenges associated with very large data (terabyte-scale) in the field of bioimaging
 80 with a particular emphasis on optimized cloud-based operations and the sharing of HDF5-based
 81 datasets. Finally, the issue of efficient database solutions for HDF5 are addressed in [1], [22].

82 Besides the specificity of the solutions, adopting aforementioned solutions to new problems is
 83 very difficult due to their complexity and required expertise in the field. When data management
 84 solutions are needed for a concrete projects, it is crucial to minimize entry barriers. Currently, for
 85 HDF5, a general approach to manage data in all aspects during its lifecycle including metadata
 86 concepts, database solutions and practical interfaces are missing. The presented Python package
 87 *h5rdmtoolbox* seeks to bridge the gap between the advanced communities with domain-specific
 88 solutions and researchers trying to manage their data without established standards in place.
 89 Leveraging well-established Python packages, this toolbox offers high-level tools and interfaces
 90 within one package, that actively contribute to the promotion of FAIR data creation. As a whole,
 91 the package seeks to be a central resource of tools for scientists allowing them to manage their
 92 HDF5 data along the full data lifecycle from planning (1) via acquisition (2) and analysis to
 93 publication in data repositories (4) and sharing in databases (5). Figure 1 illustrates these stages
 94 and relates keywords to features of the toolbox.

95 3 Concepts and architecture of the toolbox

96 A key aspect of the toolbox lies in its adaptable implementation of metadata standards and
 97 interfaces to databases and repositories, allowing it to be relevant across many research fields
 98 with varying requirements. The challenge is to attain this flexibility without introducing excessive
 99 complexity, all while ensuring adherence to the FAIR principles. The toolbox achieves this
 100 through four principles:

101 1. **Relevant programming language:** The choice of programming language significantly

102 impacts the usability and acceptance of this toolbox, as well as data handling in general.
103 Python is selected for this purpose due to its status as one of the most popular and widely
104 used language in the scientific community. The high relevance of Python in the field
105 allows the toolbox to address as many users as possible.

106 2. **One core file format:** The Hierarchical Data Format (HDF5) [23] is selected as the core
107 and general purpose file format. It is suitable for most scientific and engineering data
108 sources and allows metadata to be stored with the raw data, making it a self-explanatory
109 data store. The file format is open-source, well-supported by the HDF Group [23] and has
110 a proven track record in many disciplines. Opting for a single file format as the foundation
111 for a management toolbox is, therefore, not limiting. Prioritizing user-friendliness and
112 widespread acceptance, the toolbox implements high-level interfaces to HDF5, extending
113 the capabilities of the commonly used Python package *h5py* [24].

114 3. **Flexible Metadata Standardization:** Enabling the storage of metadata alongside raw
115 data necessitates its standardization (convention) to achieve discoverability. The toolbox
116 introduces a simple and flexible definition of so-called standard attributes. Users can design
117 their own convention, which provides feedback about the correctness of the (meta)data
118 created. This ensures that users maintain the accuracy and completeness of their data and
119 metadata.

120 4. **Extensibility:** Adaptability extends beyond just metadata standards; it encompasses
121 various aspects of the toolbox, including interfaces to databases and data repositories.
122 Abstract classes establish communication rules between HDF5 and users, enabling the
123 community to add new interfaces on top of the currently implemented ones and to make
124 them available to others through the toolbox.

125 In this work, a five-stage representation of the research data lifecycle is adopted, as illustrated
126 in Figure 1. This framework forms the basis for the toolbox's architectural design, aligning its
127 functionalities with the key stages of the data lifecycle. Consequently, the toolbox is structured
128 into five sub-packages, as depicted in Figure 3. The numerical assignments in the figure directly
129 correlate with the roles of these sub-packages in the stages of the data lifecycle (c.f. Figure 1).
130 This structured approach enhances the toolbox's utility by providing specialized tools for each
131 phase of the research data lifecycle.

132 The components of the sub-packages are designed in a manner that ensures independence from
133 each other, facilitating individual development and modularity. One exception is made to the
134 sub-packages *wrapper* and *convention*. The following sections will highlight the features and
135 implementations of the sub-packages, as well as their importance within the data lifecycle.

136 3.1 layout

137 Research projects start with a scientific question and a data management plan (DMP) [25]. The
138 DMP outlines how data is handled during and after the project. One important aspect is the
139 agreement on common exchange formats (in this work HDF5). It has a significant impact on
140 the realization of a FAIR data cycle as a whole, especially, when it comes to sharing data [20].
141 Besides a common vocabulary, the internal structure (layout) of the file is important. It is the

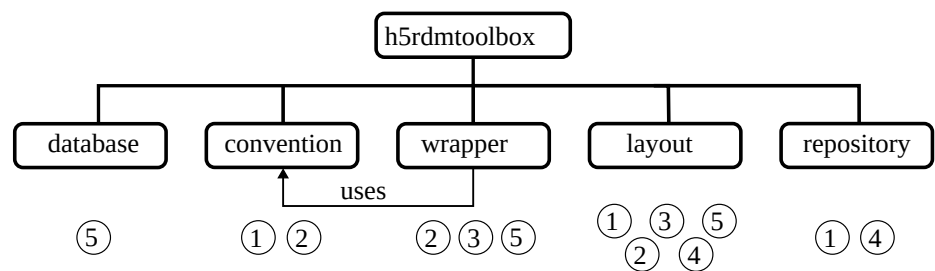


Figure 3: Organization of the sub-packages in the presented package. The core module is called *wrapper*, which adds useful functionality for the user when interacting with HDF5 files. It uses the *convention* module to manage metadata requirements when creating and reading data. The other modules are not dependent on each other and must be imported on demand. The numbers indicate their main areas of application within the different stages of the data lifecycle, as shown in [Figure 1](#).

142 basis for reliable processing and automated analysis. The hierarchical structure of HDF5 files
 143 allows various strategies to organize data and therefore must be regulated by the project data
 144 manager.

145 The sub-package *layout* implements the class *Layout*, which is a collection of specifications.
 146 Each specification is a query that is executed during the validation of a file. The rationale behind
 147 this approach is that all elements, that are expected to be present in a file, must be identifiable.
 148 The code in [Listing 1](#) illustrates the definition of a *Layout*: First, the layout object is created.
 149 Then two specifications are added by providing a query function and query parameters (using
 150 pseudocode for simplicity). The initial parameter may be any Python function that is capable
 151 of accepting the query parameters and returning a list of identified HDF5 objects. As part of
 152 the toolbox and its documentation [11], a database solution is provided within the sub-package
 153 *database*.

```

1 from h5rdmtoolbox import layout
2 lay = layout.Layout()
3 spec1 = lay.add(func=query_function,
4                 query=<has dataset with name 'x_velocity'>)
5 spec2 = lay.add(func=query_function,
6                 query=<datasets have the attribute 'units'>)
7 lay.validate("filename.hdf")
  
```

Listing 1: Code example for defining a *Layout* to validate HDF5 files based on query statements. The queries are written in pseudocode for enhanced readability.

154 The *Layout* concept should be part of every phase in the data lifecycle. Once the definition has
 155 been established during the planning phase (1), it is advisable to validate the integrity of the file
 156 at each stage. This is because the content may have been altered in the meantime, for example,
 157 the agreed internal setup or used attributes may have been modified. Verifying that the layout
 158 remains consistent with the intended definition is essential for the generation of reliable data and
 159 complete files. Avoiding missing information through careful definition of the file content in
 160 combination with regular checks is the basis of FAIR data.

161 **3.2 convention**

162 In addition to a robust HDF5 layout, the provision of meaningful and comprehensive metadata for
 163 HDF5 datasets and groups is of the utmost importance. This ensures that files are interpretable by
 164 both humans and machines. During the planning phase (1), a selection of relevant attributes for
 165 the investigated problem is important. The quality of these attributes significantly influences the
 166 findability of data within an HDF5 file, as well as the reusability and interoperability aspects of
 167 FAIR data in general. The term "quality" here refers to whether attributes are linked to existing
 168 metadata concepts that can be referenced to persistent sources. Examples include controlled
 169 vocabularies such as the CF Convention [19], metadata schemas like EngMeta [26], or ontologies
 170 like Metadata4ing [27]. These sources provide standardized and well-defined terms that enhance
 171 the clarity and consistency of metadata, contributing to improved data discoverability and reuse.
 172 Documentation for the toolbox [11] includes examples showcasing the possible utilization of
 173 these standards within the toolbox. The concept of *Conventions* is explained in the following.

174 The *h5rdmtoolbox* implements the concept of so-called *standard attributes* as part of a *Convention*
 175 object to validate relevant metadata, i.e. HDF5 attributes, during runtime as the user writes
 176 data to the file. The implementation is based on the Python package *pydantic* and hence reuses
 177 successful existing solutions. It should be noted, that this approach of attribute validation partly
 178 overlaps with the concept of *Layouts*. However, layout checks are performed after the file has
 179 been written and therefore allows for more complex requirements, that have been defined by
 180 stakeholders (e.g. dependency checks in the form of "if a dataset is named *X* then it should be
 181 1D and of data type *float32*"). The strength of using a *Convention* is, that it allows checks during
 182 data creation with immediate feedback. The focus is on usage of specific attributes and their
 183 correct usage. It is therefore especially helpful during software development, data manipulation
 184 and conversion.

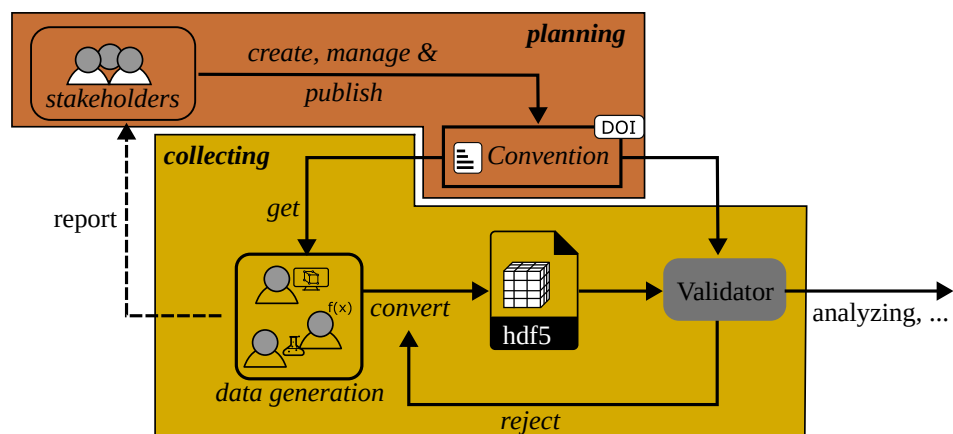


Figure 4: Workflow of collecting and converting the source data. The *Convention* validates the created HDF5 files and serves as a feedback loop to the file creators or the software developers writing the conversion scripts. Only validated files can be further processed or published.

185 **Figure 4** illustrates a common workflow, which makes use of this concept. The stakeholders
 186 of a project define and share a set of standardized attributes of type *StandardAttribute* within
 187 a *Convention*. The latter is saved in a YAML file and is shared across all users, which are

188 directly working with HDF5 files. By integrating the *Convention* into their workflows through
 189 the *h5rdmtoolbox*, they obtain direct feedback through a validation mechanism. As a result, the
 190 quality in terms of reliable and comprehensive data description through attributes is ensured and
 191 basis for the FAIRness of HDF5 file is set.

192 As shown in the class diagram in [Figure 5](#), a *Convention* object takes a list of *StandardAttribute*
 193 objects. The important properties of a *StandardAttribute* are *validator* and *target_method*. The
 194 *target_method* assigns the object to a method of the *h5py* package (other options are `__init__` or
 195 `create_group`) and the *validator* defines how the attribute is validated during assignment.

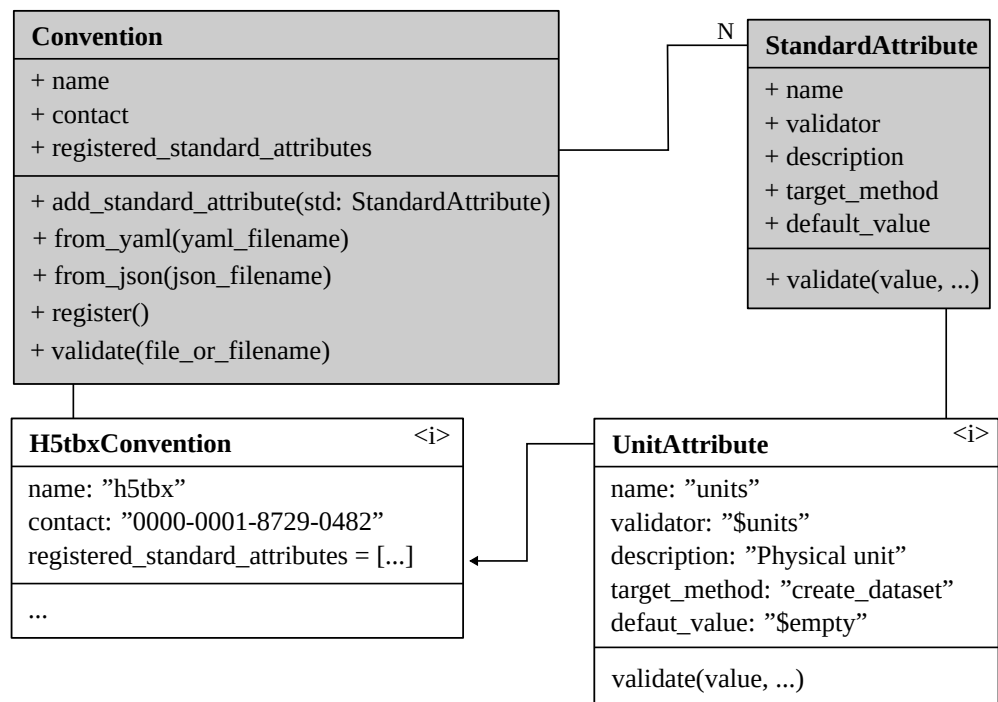


Figure 5: Class diagram of components *Convention* and *StandardAttribute*. The instances "H5tbxConvention" and "UnitAttribute" are used in [Listing 2](#).

196 A minimal example of the two instances shown in [Figure 5](#) is written in code in [Listing 2](#). Note,
 197 that the parameter "units" in the function call is not part of the underlying *h5py* package but
 198 gets dynamically added by enabling the *Convention*. It is also noteworthy that by setting the
 199 keyword "\$empty" as the default value, the attribute becomes obligatory. For HDF5 datasets,
 200 this is in fact a reasonable choice, as generally physical data is written to datasets, which require
 201 a physical unit.

202 As indicated in the class diagram, *Conventions* can also be defined in files (JSON or YAML),
 203 which allows sharing the *Convention* via data repositories or databases with all involved stake-
 204 holders. By enabling the project *Convention* during file manipulating, users receive immediate
 205 feedback on the validity of the used standardized attributes (c.f. [Figure 4](#)). This is a difference to
 206 the concept of *Layouts*, which are static validators. For further information and examples about
 207 the implementation details, pre-implemented validators as well as the user-defined creation of


```
1 import h5rdmtoolbox as h5tbx
2 h5tbx.use("h5tbx")
3 with h5tbx.File(myfile.hdf, "r+") as h5:
4     h5.create_dataset("ds", data=4, units="m/s")
5
```

Listing 2: Minimal example of using a *Convention*. By enabling the "h5tbx", the standard attribute "units" becomes obligatory in the method `create_dataset`. The value of "units" is validated and automatically added to the newly created dataset or an error is raised.

208 new ones, please refer to the documentation [11], as this information exceeds the scope of this
209 paper.

210 The documentation provides extensive details, practical examples, and guidance to support users
211 in utilizing and customizing *Conventions* and validators within the *h5rdmtoolbox*.

212 3.3 wrapper

213 The package *wrapper* plays a central role within the toolbox by implementing a thin layer
214 around the HDF5 Python library *h5py*. Besides user-friendly high-level methods for interactive
215 representation of the file content in Jupyter Notebooks or helper methods for special attributes
216 or datasets, the *wrapper* package is responsible for

- 217 • integration of the *Convention* concept into the *h5py* framework and
- 218 • metadata-aware exchange of data through *xarray* object [28].

219 The integration of the *xarray* package into the toolbox provides several advantages. As previously
220 highlighted, one of the reasons for selecting HDF5 is its compatibility with the multidimension-
221 ality of many scientific and engineering datasets, allowing the storage of attributes alongside
222 the data. However, using *numpy* arrays as part of the *h5py* package results in the loss of two
223 important sets of information. Firstly, *numpy* arrays can only represent array data, discarding
224 attributes associated with the data. Secondly, the axis of a multidimensional array can only
225 be addressed by their indices (0, 1, etc.), potentially losing references to other datasets in the
226 HDF5 format (a concept known as *dimension scales* in *h5py* [24]). This limitation hinders the
227 interpretation of values and their context.

228 The *xarray* package addresses these limitations by wrapping its functionality around *numpy*
229 arrays [28]. It enables the association of attributes to the values and allows the labeling of the
230 axes in multidimensional arrays. This structure closely aligns with the HDF5 dataset model. By
231 returning "metadata-aware" *xarray* objects, the toolbox ensures that provenance information is
232 added, enhancing the intuitiveness and reliability of data processing. The auxiliary information is
233 consistently preserved during data utilization for visualization or other post-processing steps, as
234 depicted in Figure 6. It is noteworthy, that *xarray* has a strong plotting utility, that automatically
235 extracts information from the data object, incorporating it into the labels and title of the plot.
236 The synergies between HDF5 and *xarray*, resulting in benefits like concise code and interactive
237 visualization of metadata, are best illustrated through practical examples. To gain a deeper
238 understanding and explore enhanced workflows and data operations, it is recommended to

239 consult the online documentation of the *h5rdmtoolbox* [11]. For the sake of completeness, a
 240 short example is given in the following.

241 The code example in Listing 3 demonstrates the workflow as illustrated in Figure 6. A subset of
 242 the dataset "data" is selected based on the coordinates. The return value is a *xarray.DataArray* on
 243 which the rolling mean is computed. The result is finally plotted on the screen. With only a few
 244 lines of code, the user obtains quick insight into the dataset while maintaining comprehensibility
 245 and traceability. Another notable feature wrapped around the core *h5py* package is the ability
 246 to encode the semantics of HDF5 data using the concept of Resource Description Framework
 247 (RDF) [29]. Similar to the attribute manager interface (*attrs*) of *h5py*, the toolbox uses an RDF
 248 manager (*rdf*). It enables the user to enrich HDF5 attributes, datasets and groups with formal
 249 metadata using semantic RDF triples (subject - predicate - object). More information on the
 250 technology can be found in [30].

```

1 import h5rdmtoolbox as h5tbx
2 with h5tbx.File(filename) as h5:
3     # select and read selected data and store in variable:
4     d = h5["data"].sel(x=4.3, y=0.2, method="nearest")
5
6 # process (compute rolling mean over time with window size 3):
7 drm = d.rolling(time=3).mean()
8
9 # visualize the result:
10 drm.plot()
```

Listing 3: Example of data extraction using the toolbox. The returned value is an *xarray.DataArray* containing comprehensive metadata from the underlying HDF5 dataset. This facilitates transparent data operations and minimizes potential errors. Additionally, many operations can be reduced to one line of code, which makes scripts concise and traceable.

251 Listing 4 outlines a minimal example how metadata of a person can be precisely described using
 252 Internationalized Resource Identifiers (IRI). While the choice of dataset, group and attribute
 253 names is often based on personal preferences, RDF triples add meaning to the group "contact"

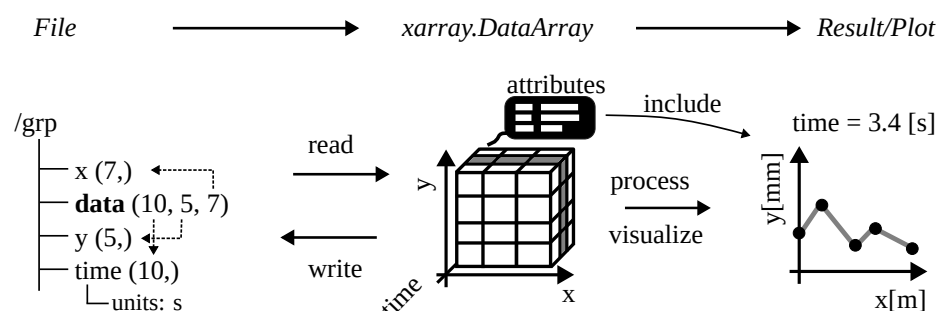


Figure 6: The *h5RDMtoolbox* makes use of the *xarray* features. Instead of *numpy* arrays, *xarray.DataArray* objects are returned, which allows carrying the dimension references and attributes and results in comprehensive data processing and visualization.

254 and its attributes. The assignment of globally unique and persistent identifiers plays a pivotal
 255 role in the fulfillment of the FAIR principles. Like this, metadata becomes interpretable by both
 256 humans and machines. The method `dump_jsonld` extracts the semantic information, which may
 257 be saved in a JSON-LD file or stored in a database for further usage. The internal HDF5 structure
 258 can also be described using RDF, but is disabled here (`structural=False`) to obtain a compact
 259 output. The documentation may be consulted for more details at this point.

```

1 import h5rdmtoolbox as h5tbx
2
3 with h5tbx.File() as h5:
4     g = h5.create_group("contact")
5     g.attrs["name"] = "Probst"
6     g.attrs["oid"] = "0000-0001-8729-0482"
7
8     # enrich with RDF metadata:
9     g.rdf.type = "http://xmlns.com/foaf/0.1/Person"
10    g.rdf.subject = "https://orcid.org/0000-0001-8729-0482"
11    g.rdf.predicate["name"] = "http://xmlns.com/foaf/0.1/lastName"
12    g.rdf.predicate["oid"] = "http://w3id.org/nfdi4ing/metadata4ing#
13                                orcidId"
14
15    print(h5.dump_jsonld(structural=False, indent=2, resolve_keys=True))
16
17 # output:
18 # {
19 #   "@context": {
20 #     "foaf": "http://xmlns.com/foaf/0.1/",
21 #     "m4i": "http://w3id.org/nfdi4ing/metadata4ing#"
22 #   },
23 #   "@id": "https://orcid.org/0000-0001-8729-0482",
24 #   "@type": "foaf:Person",
25 #   "foaf:lastName": "Probst",
26 #   "m4i:orcidId": "0000-0001-8729-0482"
27 # }

```

Listing 4: Simple example code highlighting the semantic enrichment of HDF5 data using RDF triples and globally unique identifiers from existing ontologies, here *foaf* [31] or *m4i* [32]. This approach ensures that attributes and groups are assigned a concise and clear meaning, which can be interpreted by machines and is therefore independent of the author's or project's context

260 3.4 repository

261 How data is shared depends on the scope and restrictions of the project (phase 4 in the lifecycle).
 262 Most use cases will, at least for some time, store data locally for internal use and later upload it
 263 to a data repository. The sub-package *repository* implements an abstract interface class to data
 264 repositories and their files. At the time of writing, one concrete realization of such an interface
 265 is implemented for Zenodo [33]. It is one of the most popular repositories in the scientific
 266 community to publish scientific data with open-access. Interfaces to other platforms are planned
 267 to be added in the future, such as Figshare [34] for example. The design of the *repository*
 268 sub-package explicitly promotes this by using an object-oriented design: An abstract base class

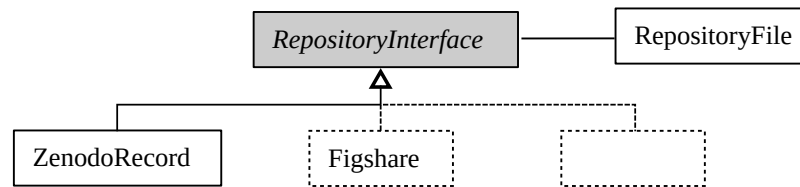


Figure 7: The architecture of the *repository* sub-package implements the basic interface (abstract class *RepositoryInterface*) to data repositories. Through One concrete implementation is provided for Zenodo (*ZenodoRecord*). The interface to files within the repository is realized through *RepositoryFile*, clearly defining common properties and download functionalities across all repositories. Other popular platforms (e.g. Figshare) could be added by code contributors to the *h5rdmtoolbox*. The gray components are abstract classes, white boxes indicate concrete implementations, and the dashed lines indicate potential extensions in future.

269 *RepositoryInterface* defines mandatory properties and methods for the user-platform interaction,
 270 as depicted in Figure 7. Moreover, the interaction of users with files within a repository is
 271 prescribed by *RepositoryFile*. The chosen design streamlines and simplifies the data exchange
 272 with repositories (see Listing 5).

273 The repository interface class implements the method *upload_file*, which allows to automatically
 274 map metadata to a secondary file, which is uploaded alongside the original file (see Figure 8). This
 275 has the following reasoning: Large files are expensive to download in terms of time, especially if
 276 it turns out, that the data is not matching the expectations of a user. As data repositories typically
 277 only offer descriptive information regarding the type of data publication (e.g., creator, version,
 278 time, keywords, license, etc.), the content of large files can only be examined after they have
 279 been downloaded.

```

1 from h5rdmtoolbox.repository import zenodo
2 from h5rdmtoolbox.wrapper import hdf2jsonld
3
4 repo = zenodo.ZenodoRecord(None, sandbox=True) # new testing deposit
5 repo.upload_file("my_file.hdf", metamapper=hdf2jsonld, skipND=1)
6
7 meta_filename = repo.files["my_file.jsonld"].download()
8 # ... review JSON-LD file and eventually download the HDF5 file
  
```

Listing 5: Example code demonstrating the upload process of HDF5 files. The *metamapper* parameter expects a function, which extracts metadata information from the HDF5 file and uploads it alongside the HDF5 file. The default function as used in the example uses *hdf2json*, which is a built-in function. It extracts the structure in the json-ld format. The parameter *skipND* is specific to *hdf2json* and is automatically passed to it.

280 Especially large HDF5 files may contain much and complex information, not only based on
 281 attributes but also from the internal structure and dataset properties. The automatic extraction of
 282 metadata is implemented for HDF5 files only as part of the toolbox and uses RDF as a universal
 283 metadata description (see e.g. Listing 4). If the user wish to use custom mappings for HDF5 files
 284 or other file formats before their upload, the custom function should be passed to the argument

285 *metamapper*. In the example shown in Listing 5, an HDF5 file is uploaded using the built-in
 286 function *hdf2jsonld* from the *wrapper*. It writes the metadata into a JSON file using the JSON-LD
 287 format, resulting in a small text file. Another user exploring the repository may download the
 288 JSON file first, which allows investigating the HDF5 metadata content, and then eventually
 289 download the potentially large HDF5 file.

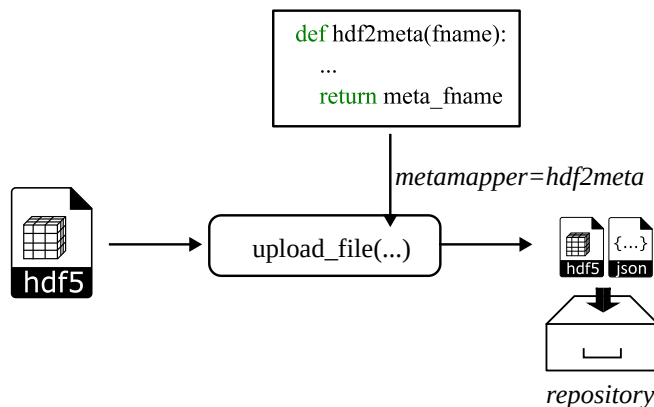


Figure 8: The workflow of uploading data to a repository involves a so-called *metamapper* function, which extracts metadata from and about the file (attributes and structure) and writes it into a JSON-LD file. This is done automatically for HDF5 files, unless the user provides a custom function (here representatively indicated with *hdf2meta*, resulting in an additional JSON file). Both files are uploaded to the repository. This procedure is especially helpful for large HDF5 files. Interested users may first download the metadata file and inspect the content before downloading the large file.

290 3.5 database

291 Exploring HDF5 data and hence an efficient re-use requires a query mechanism for the files.
 292 The toolbox implements two ways:

- 293 1. Using HDF5 as a database inside a file system.
- 294 2. Mapping HDF5 to the NoSQL database MongoDB [35].

295 Figure 9 shows the workflows for both options. The simplest solution uses an HDF5 file itself
 296 as a database and multiple files as multiple databases respectively. For the case of one file
 297 the user calls the database interface class *FileDB*, for multiple files *FilesDB*. A query call is
 298 constructed similar to the one using a dedicated database solution, which is MongoDB. Each
 299 search will recursively walk through one or multiple HDF5 files. Yet simple, this approach may
 300 be inefficient for many or large HDF5 files.

301 A second and more performant solution maps the attributes to a MongoDB database. A query
 302 on MongoDB is very efficient and allows more complex queries as compared to the current
 303 implementation of *FileDB* and *FilesDB*. Depending on the amount of files and their size, the
 304 extraction of metadata and writing to the database may be time-consuming. However, frequent
 305 query calls are processed very quickly, resulting in a faster overall solution. It should be noted,
 306 that MongoDB is used as a metadata database, which requires keeping the original HDF5 files.
 307 If no further queries are planned, the database can be deleted again.

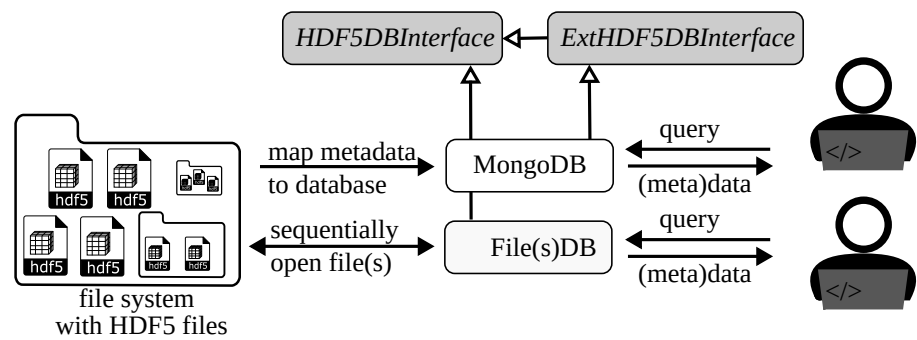


Figure 9: Workflow of the two provided database solutions: The simplest solution using *FileDB* or *FilesDB* allows the user to perform queries directly on one or many HDF5 files, respectively. The requests are sequentially executed and scan the complete files. A more efficient solution maps the metadata to a NoSQL database (MongoDB). Here, query requests can be more complex and are more efficient but requires a prior mapping process, which the toolbox provides. Both implementations are inherited from *HDF5DBInterface* ensuring a common interface between user and database solution.

308 Both approaches are implemented in the toolbox based on the abstract class *HDF5DBInterface*.
 309 This class defines two query methods (*find* and *find_one*). For implementations using external
 310 databases like MongoDB, inserting methods are required as defined in *ExtHDF5DBInterface*.
 311 The utilization of these classes defines the interface between the database and the users, and
 312 also serves as a foundation for future implementations of other third-party databases. The used
 313 syntax for the queries, the capabilities of the present solutions as well as the return data object of
 314 query results are outlined in detail in the documentation.

315 4 Documentation

316 The *h5rdmtoolbox* is versioned via a GitHub repository and can be installed using the Python
 317 package installer (pip). At the time of writing, the package version is *v1.4.0* and an extensive
 318 documentation is automatically created and published online [11]. It provides an overview of
 319 features that are not included in this paper or are only briefly discussed.

320 The documentation website is generated based on Jupyter Notebooks. On the one hand, this
 321 results in a practical documentation, showing code and explanations together. On the other hand,
 322 it allows users to reuse the code from the documentation for immediate application by simply
 323 copying the code snippets. As Jupyter Notebooks become more popular [12], [36], the option to
 324 download the full Notebooks will be another efficient option for most users who are new to the
 325 toolbox.

326 5 Limitations

327 As outlined before, this package serves as a general toolbox, introducing a management layer
 328 around HDF5 files. Therefore, its strength lies in the metadata organization and user-friendly
 329 interaction with HDF5 files, rather than high-performance data processing. This means, that

330 during dataset creation and reading, additional processing is needed to validate the metadata
331 usage. The process of actually writing and reading the file is dependent on the underlying
332 package, which is *h5py*. For large datasets, however, the overhead is negligible and the write or
333 read process is dominating. No significant time differences between *h5py* and *h5rdmtoolbox*
334 are observed. The same accounts for the performance of working with dataset values. They
335 are provided as *xarray* objects. Again, generating them based on the *numpy* array and other
336 information from the HDF5 file requires some time. After this, the performance is dependent on
337 the *xarray* package.

338 The chosen design principles introduce two inherent limitations. Firstly, the implementation
339 of the package in Python inherently limits its compatibility for users of other programming
340 languages such as C++, Java or Matlab for instance. The widespread popularity of Python in the
341 scientific community justifies the choice. While a similar implementation in other languages is
342 theoretically possible, such an extension is beyond the scope of this work. Secondly, the selection
343 of HDF5 as the core scientific file format imposes an inherent limitation. Not all scientific or
344 engineering data may be well-suited for HDF5 files. While HDF5 is versatile, some specialized
345 data types or structures may find more suitable alternatives outside the HDF5 format.

346 Finally, it is essential to note that the number of interfaces to databases and repositories is
347 currently limited. As of the current writing, the *database* sub-package includes implementations
348 for MongoDB and a query solution using HDF5 itself. In the *repository* sub-package, only
349 Zenodo is provided. Nevertheless, the toolbox is designed to permit and explicitly encourages
350 further extensions by the community. This open architecture invites collaborative contributions
351 to expand the range of interfaces and integrations with databases and repositories based on the
352 evolving needs and preferences of users.

353 6 Conclusion and Outlook

354 The Python package *h5rdmtoolbox* has been introduced, which is designed to support small
355 collaborative groups, communities, and doctoral researchers who use or consider using HDF5
356 files as their central file format. HDF5 is chosen for its self-descriptive capabilities and versatility
357 in various scientific domains. However, the management of metadata and internal organization
358 of datasets and groups, as well as facilitating interoperability with other frameworks, is left
359 to the users. The toolbox aims to enhance the FAIR principles of data by providing general,
360 comprehensive tools for managing HDF5 files throughout their lifecycle. While solutions
361 exist, that address management needs, they tend to be domain-specific and lack a generalized
362 framework applicable to diverse problems. Some solutions may only focus on specific aspects of
363 the data lifecycle, such as databases. In contrast, the presented toolbox adopts a broad approach,
364 providing tools that enable users to create tailored management solutions for HDF5 files based on
365 their specific scientific context. Rather than prescribing a singular solution, the toolbox fills the
366 gap between well-established solutions utilized by large scientific communities and the absence
367 of standards for individual researchers. By offering a Python package equipped with high-level
368 tools and interfaces for HDF5 data management, the toolbox improves the FAIRness of HDF5
369 files for scientists.

370 With user-friendliness and low entry barriers in mind, the toolbox uses popular Python packages

371 like *xarray* and *pydantic* as dependencies and adopts syntax into newly programmed solutions
372 (e.g. query within HDF5 files is adopted from MongoDB). However, the toolbox is missing
373 graphical user interfaces. This would strongly improve the usability and will lower the entry
374 level, especially for less experienced programmers. Future work should set the focus on the
375 design of *Conventions* and *Layouts*, as this constitutes the bases of successful data management.

376 The toolbox has been tested in and improved through various scientific projects with a focus
377 on fluid mechanics. However, further testing in other domains is required. In addition to
378 the implemented unit tests, practical testing in various applications is necessary to identify
379 further needs, weaknesses and thus elaborate potential for improvements. Application to various
380 problems and scientific disciplines are planned and feedback from researchers will need to
381 incorporate into the toolbox. This will extend capabilities, improve the code and allow it to be
382 adapted to the needs of users. Current concrete use cases investigate fluid problems, such as
383 computational fluid dynamics simulations and particle image velocity measurements. Lessons
384 learned from these areas will be incorporated into future publications, while further examples
385 and guidelines will be continuously added to the online documentation [11].

386 7 Acknowledgements

387 The software was developed in-house without any external funding and no conflicts of interest
388 are declared. The authors would like to thank all users, who have been testing the toolbox so far
389 and provided helpful feedback. A special thanks belongs to Lucas Büttner for the helpful testing
390 and feedback at the beginning of the project.

391 8 Roles and contributions

392 **Matthias Probst:** Conceptualization, Writing, Software Development – original draft

393 **Balazs Pritz:** Project administration, Formal Analysis, Writing - review & editing

394 References

- 395 [1] Y. Wang, Y. Su, and G. Agrawal, “Supporting a Light-Weight Data Management Layer
396 over HDF5,” in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and
397 Grid Computing*, IEEE, 2013, pp. 335–342. DOI: [10.1109/CCGrid.2013.9](https://doi.org/10.1109/CCGrid.2013.9).
- 398 [2] J. Georgieva, V. Gancheva, and M. Goranova, “Scientific Data Formats,” in *Proceedings
399 of the 9th WSEAS International Conference on Applied Informatics and Communica-
400 tions*, ser. AIC’09, Moscow, Russia: World Scientific, Engineering Academy, and Society
401 (WSEAS), 2009, pp. 19–24, ISBN: 9789604741076.
- 402 [3] E. National Academies of Sciences and Medicine, *Open Science by Design: Realizing a
403 Vision for 21st Century Research*. Washington, DC: The National Academies Press, 2018.
404 DOI: [10.17226/25116](https://doi.org/10.17226/25116).
- 405 [4] F. De Carlo, D. Gürsoy, F. Marone, *et al.*, “Scientific data exchange: a schema for HDF5-
406 based storage of raw and analyzed data,” *Journal of synchrotron radiation*, vol. 21, no. 6,
407 pp. 1224–1230, 2014.

- 408 [5] C. M. Klingner, M. Denker, S. Grün, *et al.*, “Research data management and data sharing
409 for reproducible research—results of a community survey of the german national research
410 data infrastructure initiative neuroscience,” *Eneuro*, vol. 10, no. 2, 2023.
- 411 [6] European Commission and Directorate-General for Research and Innovation, *Cost-benefit
412 analysis for FAIR research data : cost of not having FAIR research data*. Publications
413 Office, 2019. DOI: [10.2777/02999](https://doi.org/10.2777/02999).
- 414 [7] W. K. Michener, “Meta-information concepts for ecological data management,” *Ecological
415 Informatics*, vol. 1, no. 1, pp. 3–7, 2006. DOI: [10.1016/j.ecoinf.2005.08.004](https://doi.org/10.1016/j.ecoinf.2005.08.004).
- 416 [8] N. Preuss, G. Staudter, M. Weber, R. Anderl, and P. F. Pelz, “Methods and technologies for
417 research-and metadata management in collaborative experimental research,” in *Applied
418 Mechanics and Materials*, Trans Tech Publ, vol. 885, 2018, pp. 170–183.
- 419 [9] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, “The fair guiding principles for
420 scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, p. 160 018,
421 2016, ISSN: 2052-4463. DOI: [10.1038/sdata.2016.18](https://doi.org/10.1038/sdata.2016.18).
- 422 [10] A. Jacobsen, R. de Miranda Azevedo, N. Juty, *et al.*, “FAIR Principles: Interpretations
423 and Implementation Considerations,” *Data Intelligence*, vol. 2, no. 1-2, pp. 10–29, Jan.
424 2020, ISSN: 2641-435X. DOI: [10.1162/dint_r_00024](https://doi.org/10.1162/dint_r_00024).
- 425 [11] Probst, Matthias, *Documentation of HDF5 Research Data Management Toolbox (v1.4.0)*,
426 2024. [Online]. Available: <https://h5rdmtoolbox.readthedocs.io/en/v1.4.0/>,
427 (accessed: 17.06.2024).
- 428 [12] J. M. Perkel, “Why Jupyter is data scientists’ computational notebook of choice,” *Nature*,
429 vol. 563, no. 7732, pp. 145–147, 2018.
- 430 [13] P. Greenfield, M. Droettboom, and E. Bray, “ASDF: A new data format for astronomy,”
431 *Astronomy and computing*, vol. 12, pp. 240–251, 2015.
- 432 [14] E. Taaheri and D. Wynne, “An HDF-EOS and data formatting primer for the ECS project,”
433 Raytheon Company, Tech. Rep., Mar. 2001.
- 434 [15] P. Klosowski, M. Koennecke, J. Tischler, and R. Osborn, “NeXus: A common format for
435 the exchange of neutron and synchrotron data,” *Physica B: Condensed Matter*, vol. 241,
436 pp. 151–153, 1997.
- 437 [16] T. Hauser, “Parallel i/o for the cgns system,” in *42nd AIAA Aerospace Sciences Meeting
438 and Exhibit*, 2004, p. 1088. DOI: [10.2514/6.2004-1088](https://doi.org/10.2514/6.2004-1088).
- 439 [17] S. Koranne, *Hierarchical Data Format 5 : HDF5*. Springer, 2011, pp. 191–200, ISBN:
440 978-1-4419-7719-9. DOI: [10.1007/978-1-4419-7719-9_10](https://doi.org/10.1007/978-1-4419-7719-9_10).
- 441 [18] S. Poirier, A. Buteau, M. Ounsy, *et al.*, “Common Data Model Access: A Unified Layer to
442 Access Data From Data Analysis Point OF View,” *Icalepcs, Grenoble, October*, 2011.
- 443 [19] J. Gregory, “The CF metadata standard,” *CLIVAR Exchanges*, vol. 8, no. 4, p. 4, 2003.
- 444 [20] J. Moore and S. Kunis, “Zarr: A cloud-optimized storage for interactive access of large
445 arrays,” in *Proceedings of the Conference on Research Data Infrastructure*, vol. 1, 2023.
- 446 [21] A. Ingargiola, T. Laurence, R. Boutelle, S. Weiss, and X. Michalet, “Photon-HDF5: an open
447 file format for timestamp-based single-molecule fluorescence experiments,” *Biophysical
448 journal*, vol. 110, no. 1, pp. 26–33, 2016.

- 449 [22] L. Gosink, J. Shalf, K. Stockinger, K. Wu, and W. Bethel, “HDF5-FastQuery: Accelerating
450 complex queries on HDF datasets using fast bitmap indices,” in *18th International
451 Conference on Scientific and Statistical Database Management (SSDBM’06)*, IEEE, 2006,
452 pp. 149–158.
- 453 [23] The HDF Group, *Hierarchical Data Format, version 5*. [Online]. Available: <https://www.hdfgroup.org/HDF5/>, (accessed: 18.12.2023).
454
- 455 [24] A. Collette, *Python and HDF5*. O’Reilly Media, Inc., 2013, ISBN: 9781449367831.
- 456 [25] A. Salazar, B. Wentzel, S. Schimmler, R. Gläser, S. Hanf, and S. A. Schunk, “How research
457 data management plans can help in harmonizing open science and approaches in the digital
458 economy,” *Chemistry—A European Journal*, vol. 29, no. 9, e202202720, 2023.
- 459 [26] B. Schembera and D. Iglezakis, “EngMeta: metadata for computational engineering,”
460 *International Journal of Metadata, Semantics and Ontologies*, vol. 14, no. 1, pp. 26–38,
461 2020.
- 462 [27] D. Iglezakis, D. Terzijska, S. Arndt, *et al.*, “Modelling scientific processes with the m4i
463 ontology,” in *Proceedings of the Conference on Research Data Infrastructure*, vol. 1,
464 2023. DOI: [/10.52825/cordi.v1i.271](https://doi.org/10.52825/cordi.v1i.271).
- 465 [28] S. Hoyer and J. Hamman, “xarray: ND labeled arrays and datasets in Python,” *Journal of
466 Open Research Software*, vol. 5, no. 1, 2017.
- 467 [29] Manola, F., Miller, E., *Resource Description Framework (RDF). Primer. W3C Recommendation 10 February 2004*, 2004. [Online]. Available: <http://www.w3.org/TR/rdf-primer/>, (accessed: 17.06.2024).
468
469
- 470 [30] P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure, *Semantic Web: Grundlagen*. Springer,
471 2008, vol. 1.
- 472 [31] D. Brickley and L. Miller, *FOAF vocabulary specification 0.99*, 2014. [Online]. Available:
473 <http://xmlns.com/foaf/spec/>, (accessed: 17.06.2024).
- 474 [32] S. Arndt, B. Farnbacher, M. Fuhrmans, *et al.*, “Metadata4Ing: An ontology for describing
475 the generation of research data within a scientific activity,” 2023. DOI: [/10.5281/zeno
476 do.8382665](https://doi.org/10.5281/zenodo.8382665).
- 477 [33] M.-A. Sicilia, E. García-Barriocanal, and S. Sánchez-Alonso, “Community curation in
478 open dataset repositories: Insights from zenodo,” *Procedia Computer Science*, vol. 106,
479 pp. 54–60, 2017. DOI: [10.1016/j.procs.2017.03.009](https://doi.org/10.1016/j.procs.2017.03.009).
- 480 [34] M. Thelwall and K. Kousha, “Figshare: A universal repository for academic resource
481 sharing?” *Online Information Review*, vol. 40, no. 3, pp. 333–346, 2016. DOI: [10.1108
482 /OIR-06-2015-0190](https://doi.org/10.1108/OIR-06-2015-0190).
- 483 [35] K. Chodorow and M. Dirolf, *MongoDB - The Definitive Guide: Powerful and Scalable
484 Data Storage*. O’Reilly, 2010, pp. I–XVII, 1–193, ISBN: 978-1-449-38156-1.
- 485 [36] T. Kluyver, B. Ragan-Kelley, F. Pérez, *et al.*, “Jupyter Notebooks—a publishing format for
486 reproducible computational workflows,” *Elpub*, vol. 2016, pp. 87–90, 2016.