


# plotID - a toolkit for connecting research data and visualization

Martin Hock  <sup>1</sup>

Hannes Mayr  <sup>1</sup>

Manuela Richter  <sup>1</sup>

Jan Lemmer  <sup>1</sup>

Peter F. Pelz  <sup>1</sup>

1. Chair of Fluid Systems, Technische Universität Darmstadt, Darmstadt.


1



#### Date Published:

#### Reviewer:

#### Licenses:

This article is licensed under: 

#### Keywords:

research data management, visualization, figure, plot, mapping, referencing, ID

#### Data availability:

Data can be found here:

[example.py](#)

#### Software availability:

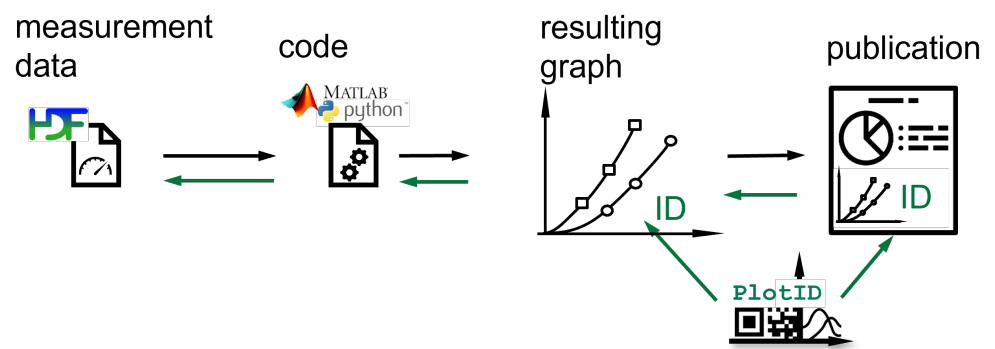
Software can be found here:

[git.rwth-aachen.de/plotid/plotid\\_python](https://git.rwth-aachen.de/plotid/plotid_python)

2

## 3 1 Statement of need

4 Scientific results are published in the form of hypotheses, axioms and equations as well as text  
 5 and diagrams. Likewise, research software is being published more and more frequently. The  
 6 comprehensibility of scientific results is indispensable for scientific discourse and reproducibility.  
 7 Hypotheses, axioms and equations are usually published in text form and can be referenced  
 8 accordingly. Software can be made traceable and referencable through the use of version control  
 9 software. But what about diagrams? A diagram published in a paper is difficult to trace because  
 10 the (raw) data is usually not available. However, the traceability of diagrams and the data they  
 11 contain is not only a challenge in publication but also in everyday research. Diagrams are used for  
 12 visualization and are therefore often produced for interim results. While the researcher continues  
 13 the research process with investigations, experiments or simulations, volatile but important  
 14 information like metadata, background information and details of the data processing are lost.  
 15 To be able to reconstruct the complete path, a treasure map is needed, starting from a publication,  
 16 marking important waypoints of the process back to the raw data and metadata. This map needs  
 17 to be provided along with the product that will be reviewed the most - the created diagram. If  
 18 diagrams - regardless of whether they are published later or only serve as interim results - are



**Figure 1:** Research workflow from left to right; Afterwards following the chain of references from right to left

'plotID-referencing' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

19 provided with an identifier, that connects to previous steps, traceability can be ensured. Figure  
 20 1 shows the order in which important elements are created and how the reference chain tracks  
 21 back.

22 A tool designed to meet these needs must meet the following requirements:

- 23 • Diagrams must have a unique identifier.
- 24 • The identifier must reference the raw data, relevant metadata and the code used to process  
 25 the data.
- 26 • The method must be easy to implement into the existing research workflow.

27 To reduce the effort of organizing figures along with all necessary data and metadata for later  
 28 review and reuse, the tool plotID was developed. plotID meets all the above-mentioned require-  
 29 ments and its implementation is described in this paper. The tool is limited to usage in an existing  
 30 python environment, but investigations on enabling independent installation and execution or  
 31 offering plotID as a web-based service are ongoing. The software depends on multiple python  
 32 libraries and is currently limited to visualizations from the [15] and general picture files such as  
 33 PNG and JPG.

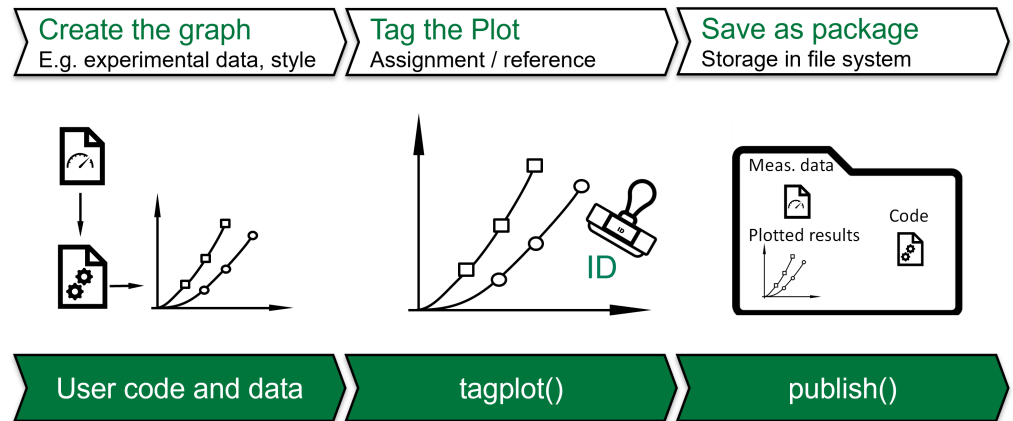
34 Researchers often tend to keep an Excel table, noting down manually which data file corresponded  
 35 with which result along with input parameters. Sometimes an ID system is used (counting up or  
 36 using the date), but interim results like visualizations - used to verify results - are usually not  
 37 included. Reviewing and understanding the environment of solutions used in and specifically  
 38 created for research data management (rdm) stays an ongoing process. The named products  
 39 in this paragraph are meant to provide some overview and examples but are by no means a  
 40 comprehensive or rated list. The reviewed solutions range from simple local scripts and libraries  
 41 (like plotID), backup and synchronization software (for filesystem like ZFS[20] and for folders  
 42 like rsync[22]), software version control (git[4], svn[1]) and software to extend their functionality  
 43 (git LFS[5], git-fat[14], git-annex[6]) to better handle binary and large data up to dedicated  
 44 workflow management systems (DataLad[12], DVC[3], signac[23]). Another area of solutions  
 45 focuses on providing the working environment by integrating documentation with programming

46 (Jupyter Notebooks[18]), providing bespoke and versioned Virtual Research Environments  
47 (VREs) or offering programmable or fixed - often discipline-specific - schematics in Electronic  
48 Laboratory Notebooks (ELNs such as eLabFTW[2], RSpace[21]) . With more comprehensive  
49 solutions and added functionality for sharing and exporting data, products lean more towards  
50 a client-server structure or even a fully hosted product with web and API interfaces. Many  
51 solutions are Open Source with Software as a Service (SaaS) offerings. Versioning often uses  
52 hashing algorithms for security reasons, thus providing unique identifiers for a specific state  
53 (snapshot) out of the box, although those are not always used for identification in user interfaces.  
54 Some hosted services implement filesystem-level software to equip each data resource with  
55 identifiers to track them independently of their current storage location (iRODs[13]). Structuring  
56 and organizing data is part of most rdm solutions and even rather strict ELN products offer to  
57 append files, images and comments to their organizational units (a probe or process). Export  
58 and sharing of research data along with its metadata is an integral part of most rdm solutions,  
59 and most offer more refined functionality and compatibility than plotID. DVC (Data Version  
60 Control) offers functionality to create plots and visualizations as part of the versioning workflow  
61 as well as overlaying multiple versions to show differences between plotted results [25].

62 While organization of data, metadata and code as much as identification, versioning and export  
63 could be found in several products, the unique feature of applying an ID visibly to a visual  
64 representation could not be found in any other solution. This is not to say that plotID is superior,  
65 but that it is a niche application. With the big difference in scope, plotID could be implemented  
66 as part of a workflow complementing most of the above mentioned solutions. Only some of the  
67 most restrictive ELNs or filesystem level operations are unlikely to be compatible.

## 68 2 Methodology

69 The developed tool plotID is a software solution which covers the above-mentioned needs. The  
70 concepts and methods underlying the software are independent of the programming language.  
71 The aim of the software is to support the research workflow shown in figure 2 and to enable  
72 traceability. plotID aims to help during the early research process to decrease the work of making  
73 publications reproducible later on. To ensure ease of use, the tool has been designed to be  
74 integrated seamlessly into existing scripts. For this purpose, a GUI has been omitted. Instead,  
75 two main functions (building blocks) are provided, which can be inserted into existing user  
76 scripts as a one-liner. They are the core of plotID. The first module creates a (unique) ID and  
77 stamps this ID onto an object containing a visualization, while the second module helps organize  
78 all relevant code, software, and data that went into creating this graphic, into one complete  
79 package. Furthermore, connectivity to existing identifiers is ensured. If a specific visualization  
80 is later chosen to be included in a publication, the ID can be replaced by a permanent identifier  
81 like a DOI and the package of code, software and data can be published at the location referenced  
82 by the DOI. The ID on in the published paper will then directly reference the data, software and  
83 code used to create it, hence curating reproducibility. In the following, plotID is presented in  
84 more detail using the Python implementation.



**Figure 2:** Workflow integrating the plotID core functions

'plotID-workflow' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

### 85 3 Python - Implementation

86 The first version of plotID was implemented in Matlab since this is the most widely used  
 87 programming language in the local working environment and the language the authors had the  
 88 highest familiarity with. After reaching a usable state, the focus shifted to rewriting the tool in  
 89 Python, the second most used programming language (locally). The goal was to make plotID  
 90 accessible to a broader audience. Moreover, in contrast to Matlab, Python better fulfils the  
 91 requirements for reusable software in the sense of the FAIR principles. In addition to being  
 92 widely used in the engineering and research community, Python is non-proprietary, open source,  
 93 easy to install or even shipped along many operating systems. Python also offers a package  
 94 index (*PyPI*) and installer (*pip*) for easy distribution of software packages.

### 95 4 Core functions

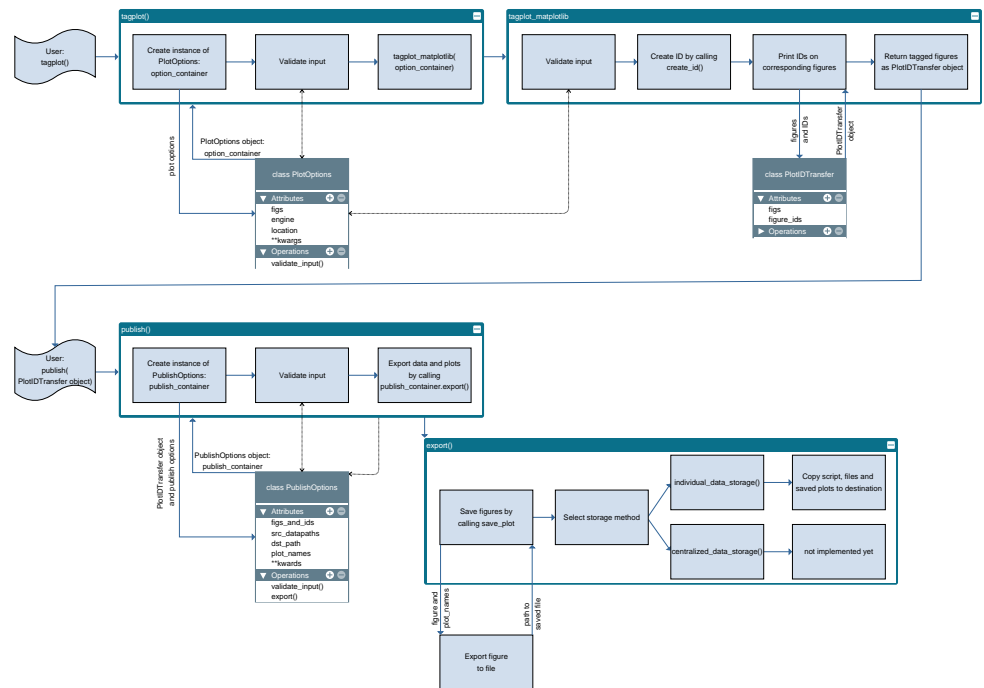
96 The core functions of plotID are *tagplot()* and *publish()*. *tagplot()* generates an ID and adds  
 97 this ID to the figure object. *publish()* saves the figure object and image file, along with the  
 98 script file, plotID was called from – the essential elements necessary to recreate the visualization  
 99 from scratch. At this time, plotID does not yet export the python environment or the imported  
 100 modules. Refer to the subsection 'Script' for more information about the planned implementation.  
 101 Additional functionality might bring additional steps with the further development of plotID and  
 102 a widening of its scope.

#### 103 4.1 tagplot()

104 The *tagplot()* function creates an ID and tags the figure object with this ID.

##### 105 4.1.1 ID

106 *tagplot()* creates a unique ID (unique in a restricted system), that consists of a static part and  
 107 a generated part. The static part is handed over as a parameter and is meant to be used to  
 108 identify a project or organizational unit to which the figure is assigned. The generated part is by



**Figure 3:** System architecture diagram

'plotID-system-architecture' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

109 default created from the UNIX-Time stamp in hexadecimal form. As an alternative option, a  
 110 random number generator can be used. The implementation of the ID is modular, enabling easy  
 111 implementation of individual needs or sources for IDs.

#### 112 4.1.2 Tagging

113 In Python, there are multiple available packages that can produce visualizations from data.  
 114 Adding an ID needs to be implemented for many of these engines separately. For now plotID  
 115 supports figures created with *matplotlib* and raw image files. The ID is added as an attribute to  
 116 the object and the graphical, visible item.

#### 117 4.1.3 Arguments

118 Necessary input arguments for *tagplot(figs, engine[, prefix, id\_method, location])*:

- 119 • *figs*: the figure object or a list of objects, that is to be tagged
- 120 • *engine*: the plot/image engine to be used (currently only 'matplotlib' and 'image' (for plain  
 121 image files) are supported)

122 Optional input arguments are:

- 123 • *prefix*: to define a static part of each created ID (prefix='Ing.grid-'). Type: string.
- 124 • *id\_method*: to define how the unique part of the ID is created ('random', 'time'). Type:  
 125 string.

126 • *location*: to define the position the ID is displayed in, relative to the full graphical ob-  
127 ject (cardinal directions like 'west', custom inputs for rotation and position are to be  
128 implemented). Type: string.

129 Output arguments are the tagged object and ID, if a list of objects was input, then the output is a  
130 list as well.

131 At this point the figure object can still be modified, for example, to adjust colours or positioning  
132 or recreate the full plot before exporting a final version.

## 133 4.2 publish()

134 This function starts the export process. The source files of the processed data, the visualization  
135 (including the tagged ID), and the script hosting the call to the publish function are copied  
136 together into a destination folder.

### 137 4.2.1 Script

138 A function in Python has access to the file path of the script which it was called from. With this,  
139 the code for calculations can easily be gathered. For this reason, *publish()* cannot be called from  
140 the command line or from within a script that has been started with the 'python -m' flag.

141 For dependent packages, the 'import' lines of the script can be compared with the output of 'pip  
142 freeze' which returns all installed packages including their version. The overlap of these lists  
143 can be written into a requirements.txt, which is added to the exported folder. Using the Python  
144 package installer *pip* the original package versions can be reinstalled. This has not yet been  
145 implemented. Furthermore, the user has to take care of including additional function files as  
146 data paths, that have not been imported but are still accessed by the executed script.

### 147 4.2.2 Data files

148 Data files are handed over as a list of file or folder paths. Ideally, the script already manages  
149 a list of all files that are read during the execution of the script. It is up to the user to control  
150 this. By default, the data files are copied to each exported package. For large data files, the  
151 *centralized* flag is intended.

152 The data files are copied to a central folder, relative to the export packages. For further exports,  
153 the data files are compared to the ones already copied and only copied if new data files are  
154 present. With this, a publication on a data repository could encompass the data files in addition  
155 to multiple "satellite" folders containing the specific script, parameters and graphics. For HDF5  
156 files, each package can contain an empty HDF5 file that only contains a link to the "real" central  
157 data file. While this has proven to be helpful in the Matlab implementation, the Python version  
158 aims to include the 'centralized' option in a future release.

### 159 4.2.3 Arguments

160 Necessary input arguments for *publish(src\_datapath, dst\_path, figure, plot\_name[, ...])* are:

- 161 • *src\_datapath*: This can be a single or a list of file or folder paths for source data and  
162 additional function files. The type is a string or a list of strings.
  - 163 • *dst\_path*: This is the destination folder path. If it does not exist, the folder will be created.  
164 The type is a string.
  - 165 • *figure*: This is a figure object, the exact class depends on the plot engine used. This object  
166 will be turned into an image file.
  - 167 • *plot\_name*: This is the name for the graphics objects. The type is a string or list of strings.  
168 If a single name is passed for multiple objects, a raising number will be added.
- 169 Optional input arguments:
- 170 • *data\_storage*: Currently only 'individual' and 'centralized' are available. 'Individual' will  
171 store all data in each exported package, while 'centralized' stores the data files in a central  
172 folder separate from the packages containing script and image files.

## 173 5 Example script

174 The following script shows how plotID is used.

```
175 10 # %% Import modules
176 11 import numpy as np
177 12 import matplotlib.pyplot as plt
178 13 from plotid.tagplot import tagplot
179 14 from plotid.publish import publish
180 15
181 16 # %% Set Project ID
182 17 PROJECT_ID = "MR04_"
183 18
184 19 # %% Create sample data
185 20 x = np.linspace(0, 10, 100)
186 21 y = np.random.rand(100) + 2
187 22 y_2 = np.sin(x) + 2
188 23
189 24 # %% Create sample figures
190 25
191 26 # 1. figure
192 27 IMG1 = 'image1.png'
193 28 FIG1 = plt.figure()
194 29 plt.plot(x, y, color='black')
195 30 plt.plot(x, y_2, color='yellow')
196 31 plt.savefig(IMG1)
197 32
198 33 # 2. figure
199 34 IMG2 = 'image2.png'
200 35 FIG2 = plt.figure()
```

```

201 36 plt.plot(x, y, color='blue')
202 37 plt.plot(x, y_2, color='red')
203 38 plt.savefig(IMG2)

```

204 In this part, the plotID modules and those necessary to create figures and images are imported.  
 205 The variable *PROJECT\_ID* is set to provide a static part of the ID. Random data is used to create  
 206 two figures with matplotlib and their image files.

```

207 42 # %% TagPlot
208 43
209 44 # If multiple figures should be tagged, they must be provided as list
210
211 45 FIGS_AS_LIST = [FIG1, FIG2]
212 46 IMGS_AS_LIST = [IMG1, IMG2]
213 47
214 48 # Example for how to use tagplot with matplotlib figures
215 49 [TAGGED_FIGS, ID] = tagplot(FIGS_AS_LIST, 'matplotlib',
216 50                             prefix=PROJECT_ID, id_method='time', location='west')
217 51
218 52 # Example for how to use tagplot with image files
219 53 # [TAGGED_FIGS, ID] = tagplot(IMGS_AS_LIST, 'image', prefix=
220 54 PROJECT_ID,
221 54 #                                     id_method='random', location='west')

```

222 Both matplotlib objects are tagged with a generated ID in one line of code. Tagging the image  
 223 files has been commented out in this case.

```

224 54 # %% Publish

```

225 Files (README.md and LICENSE) and a folder from the code repository are used in place of  
 226 research data files. The folder ending with '-exports' is the destination, and 'testimage' is a freely  
 227 chosen name for the exported image files.

228 This also shows that the workflow does not depend on any kind of file format or pre-organized  
 229 structures. Any kind of data can be used, and even if the library creating the visualization is not  
 230 (yet) supported, the resulting image file can still be tagged.

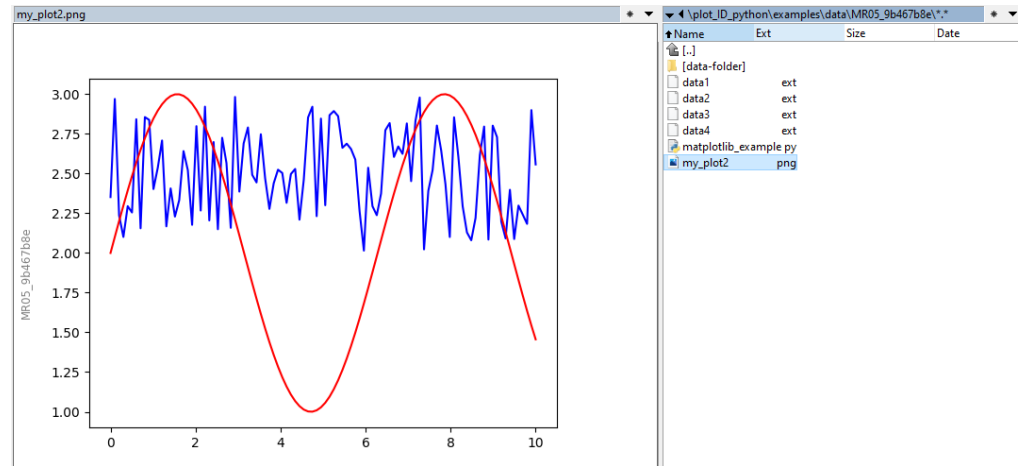
231 Figure 4 shows the resulting export folder with (renamed) data files, the script file and the tagged  
 232 plot.

## 233 6 Distribution

234 Providing easy ways to acquire and use the software is important for adoption. The code is  
 235 open source under the Apache-v2.0 license. plotID requires a Python version  $\geq 3.10$  and is  
 236 OS-independent. The current release version is v0.2.1 showing an alpha state.

237 At this time, the following distribution methods are available and described in the repository's[11]  
 238 README file.





**Figure 4:** Example export folder and tagged plot  
'plotID-example-export' by Martin Hock, licensed under [CC-BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

## 239 6.1 Source Code

240 The plain source code is publicly available on a GitLab repository located under [git.rwth-](https://git.rwth-aachen.de/plotID/plotID_python/)  
241 [aachen.de/plotID/plotID\\_python/\[11\]](https://git.rwth-aachen.de/plotID/plotID_python/) and can be directly downloaded or cloned with git.

```
242 1 git clone https://git.rwth-aachen.de/plotid/plotid_python.git
243 2 cd plotid_python
244 3 pip install -r requirements.txt
245 4 pip install .
```

## 246 6.2 Python Package

247 plotID is listed in the official Python Package Index (PyPI)[19]. The installation is done with the  
248 following command:

```
249 pip install plotid
```

250 Distributing plotID independently from an existing Python installation is one of the aims of  
251 later versions. Possible ways to achieve this are providing compiled executables or a central  
252 web-hosted service.

## 253 7 Ensuring good software quality

254 To ensure continuous good software quality, we adhere to best practices and the style guide  
255 PEP-8[17]. This includes comments, docstrings and code formatting. To ensure adherence to  
256 these guidelines, automated tests on the code are implemented.

### 257 7.1 Unit tests

258 Python offers various libraries for unit testing. plotID is using the *unittest* module[24], which  
259 is delivered with Python by default. Tests for each function are defined in the *tests* folder,  
260 along with the *runner\_test.py* script which organizes the execution of the tests, by discovering

261 the test files based on their location. The *unittest* module also measures the code covered by  
262 the tests, and total coverage of less than 95% is considered as failed. The tests are executed  
263 by a GitLab CI/CD pipeline[9] with every commit and merge request. Additional Jobs in the  
264 pipeline execute Pylint[8] and Flake8[7] to check against coding style, programming errors and  
265 cyclomatic complexity. Commits that fail the pipeline tests cannot be merged into the main  
266 branch and will not make it into a release version. In the future, additional tests e.g. against  
267 security risks introduced by dependencies and more detailed reports are planned.

## 268 7.2 Documentation

269 To ensure easy access and understanding of the code, Python docstrings[16] have been imple-  
270 mented in the source code from the beginning. The docstrings are compiled into HTML using  
271 the Sphinx[26] python package and GitLab CI-CD[9] creating an automatically generated API  
272 reference. The documents are hosted using GitLab Pages[10]. This documentation[27] will be  
273 improved by adding the readme, example code, example use cases and an introductory text until  
274 version 1.0.

## 275 8 Conclusion

276 The idea of plotID is simple yet. As with most research data management operations, the benefit  
277 for the additional work presents at a later time – although in this case, it presents itself for  
278 the creator of data or visualizations and not only for later reuse. The code and open-source  
279 implementation are still work-in-progress, but the core functionality is present. There are many  
280 ideas to improve and add functionality present already. Bug reports, merge requests with code,  
281 ideas for features and all feedback are welcome and best voiced in the GitLab repository.

## 282 9 Acknowledgements

283 We acknowledge the help from Jan Stifter and Benjamin Hermann with testing the software and  
284 feedback on the user interface.

285 The authors would like to thank the Federal Government and the Heads of Government of the  
286 Länder, as well as the Joint Science Conference (GWK), for their funding and support within the  
287 framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) -  
288 project number 442146713.

## 289 10 Roles and contributions

290 **Martin Hock:** Conceptualization, Methodology, Coding, Tests, Writing – original draft

291 **Hannes Mayr:** Coding, Tests, Methodology

292 **Manuela Richter:** Conceptualization, Methodology, Coding

293 **Jan Lemmer:** Conceptualization, Methodology

294 **Peter F. Pelz:** Project administration, Supervision, Funding Acquisition

**295 References**

- 296 [1] *Apache Subversion*. 2022. URL: <https://subversion.apache.org/> (visited on  
297 11/15/2022).
- 298 [2] Nicolas CARPi, Alexander Minges, and Matthieu Piel. “eLabFTW: An open source  
299 laboratory notebook for research labs”. In: *Journal of Open Source Software* 2.12 (2017),  
300 p. 146. DOI: [10.21105/joss.00146](https://doi.org/10.21105/joss.00146). URL: [https://doi.org/10.21105/joss.00](https://doi.org/10.21105/joss.00146)  
301 [146](https://doi.org/10.21105/joss.00146).
- 302 [3] *Data Version Control - DVC*. 2022. URL: <https://dvc.org/> (visited on 11/15/2022).
- 303 [4] *Git*. 2022. URL: <https://git-scm.com/> (visited on 11/15/2022).
- 304 [5] *Git Large File Storage | An open source Git extension for versioning large files*. 2022.  
305 URL: <https://git-lfs.github.com/> (visited on 11/15/2022).
- 306 [6] *git-annex*. 2022. URL: <https://git-annex.branchable.com/> (visited on 11/15/2022).
- 307 [7] GitHub. *flake8/index.rst at main · PyCQA/flake8*. 2022. URL: [https://github.com](https://github.com/PyCQA/flake8)  
308 [/PyCQA/flake8](https://github.com/PyCQA/flake8) (visited on 08/29/2022).
- 309 [8] GitHub. *PyCQA/pylint: It's not just a linter that annoys you!* 2022. URL: [https://github](https://github.com/PyCQA/pylint)  
310 [ub.com/PyCQA/pylint](https://github.com/PyCQA/pylint) (visited on 08/29/2022).
- 311 [9] *GitLab CI/CD | GitLab*. 2022. URL: <https://docs.gitlab.com/ee/ci/> (visited on  
312 08/19/2022).
- 313 [10] *GitLab Pages | GitLab*. 2022. URL: [https://docs.gitlab.com/ee/user/project](https://docs.gitlab.com/ee/user/project/pages/)  
314 [/pages/](https://docs.gitlab.com/ee/user/project/pages/) (visited on 08/29/2022).
- 315 [11] GitLab RWTH Aachen. *PlotID / plotID\_python · GitLab*. 2022. URL: [https://git.rwth](https://git.rwth-aachen.de/plotid/plotid_python)  
316 [-aachen.de/plotid/plotid\\_python](https://git.rwth-aachen.de/plotid/plotid_python) (visited on 08/19/2022).
- 317 [12] Yaroslav O. Halchenko et al. “DataLad: distributed system for joint management of code,  
318 data, and their relationship”. In: *Journal of Open Source Software* 6.63 (2021), p. 3262.  
319 DOI: [10.21105/joss.03262](https://doi.org/10.21105/joss.03262). URL: <https://doi.org/10.21105/joss.03262>.
- 320 [13] Mark Hedges, Adil Hasan, and Tobias Blanke. “Management and Preservation of Research  
321 Data with IRODS”. In: *Proceedings of the ACM First Workshop on CyberInfrastructure: Information Management in EScience*. CIMS '07. Lisbon, Portugal: Association for  
322 Computing Machinery, 2007, pp. 17–22. ISBN: 9781595938312. DOI: [10.1145/13173](https://doi.org/10.1145/1317353.1317358)  
323 [53.1317358](https://doi.org/10.1145/1317353.1317358). URL: <https://doi.org/10.1145/1317353.1317358>.
- 324
- 325 [14] *jedbrown/git-fat: Simple way to handle fat files without committing them to git, supports*  
326 *synchronization using rsync*. 2022. URL: <https://github.com/jedbrown/git-fat>  
327 (visited on 11/15/2022).
- 328 [15] *Matplotlib - Visualizations with python*. 2022. URL: <https://matplotlib.org/>  
329 (visited on 10/10/2022).
- 330 [16] *PEP 257 – Docstring Conventions | peps.python.org*. 2022. URL: [https://peps.pyth](https://peps.python.org/pep-0257/)  
331 [on.org/pep-0257/](https://peps.python.org/pep-0257/) (visited on 08/29/2022).
- 332 [17] *PEP 8 – Style Guide for Python Code | peps.python.org*. 2022. URL: [https://peps.py](https://peps.python.org/pep-0008/)  
333 [thon.org/pep-0008/](https://peps.python.org/pep-0008/) (visited on 10/11/2022).

- 334 [18] *Project Jupyter | Home*. 2022. URL: <https://jupyter.org/> (visited on 11/15/2022).
- 335 [19] *PyPI. PyPI · The Python Package Index*. 2022. URL: <https://pypi.org/> (visited on  
336 08/19/2022).
- 337 [20] O. Rodeh and A. Teperman. “zFS - a scalable distributed file system using object disks”. In:  
338 *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*,  
339 *2003. (MSST 2003). Proceedings*. 2003, pp. 207–218. DOI: [10.1109/MASS.2003.1194](https://doi.org/10.1109/MASS.2003.1194858)  
340 [858](https://doi.org/10.1109/MASS.2003.1194858).
- 341 [21] *RSpace ELN & Inventory*. 2022. URL: <https://www.researchspace.com/> (visited  
342 on 11/15/2022).
- 343 [22] *rsync*. 2022. URL: <https://rsync.samba.org/> (visited on 11/15/2022).
- 344 [23] *signac - simple data management - signac*. 2022. URL: <https://signac.io/> (visited  
345 on 11/15/2022).
- 346 [24] *unittest — Unit testing framework — Python 3.10.6 documentation*. 2022. URL: [https://](https://docs.python.org/3/library/unittest.html)  
347 [docs.python.org/3/library/unittest.html](https://docs.python.org/3/library/unittest.html) (visited on 08/29/2022).
- 348 [25] *Visualizing Plots | Data Version Control - DVC*. 2022. URL: [https://dvc.org/doc/u](https://dvc.org/doc/user-guide/experiment-management/visualizing-plots)  
349 [ser-guide/experiment-management/visualizing-plots](https://dvc.org/doc/user-guide/experiment-management/visualizing-plots) (visited on 11/15/2022).
- 350 [26] *Welcome — Sphinx documentation*. 2022. URL: [https://www.sphinx-doc.org/en/m](https://www.sphinx-doc.org/en/master/)  
351 [aster/](https://www.sphinx-doc.org/en/master/) (visited on 08/29/2022).
- 352 [27] *Welcome to PlotID’s documentation! — plotID 0.1.2 documentation*. 2022. URL: [https://](https://plotid.pages.rwth-aachen.de/plotid_python/)  
353 [plotid.pages.rwth-aachen.de/plotid\\_python/](https://plotid.pages.rwth-aachen.de/plotid_python/) (visited on 08/29/2022).