Datatractor: Metadata, automation, and registries for extractor interoperability in the chemical and materials sciences

Matthew L. Evans,* Gian-Marco Rignanese, David Elbert, and Peter Kraus*

Received: 12 January 2025 / Revised: 1 April 2025 / Accepted: 23 April 2025 / Published online: 18 June 2025

Two key issues hindering the transition toward findable, accessible, interoperable, and reusable (FAIR) data science are the poor discoverability and inconsistent instructions for the use of data extractor tools (i.e., how we go from raw data files created by instruments, to accessible metadata and scientific insight). If the existing format conversion tools are hard to find, install, and use, their reimplementation will lead to a duplication of effort and an increase in the associated maintenance burden is inevitable. The `Datatractor` framework presented in this article addresses these issues. First, by providing a curated registry of such extractor tools, their discoverability will increase. Second, by describing them using a standardized but lightweight schema, their installation and use are machine-actionable. Finally, we provide a reference implementation for such data extraction. The `Datatractor` framework can be used to provide a public-facing data extraction service, or be incorporated into other research data management tools providing added value.

Impact statement

The ongoing digitalization of materials research necessitates investment in bespoke research data management platforms. A common pain point across such infrastructures is the ingestion and extraction of data and metadata from raw data files. Often these files require significant effort to ingest in a way consistent with the FAIR data principles such that definitions are clear and the resulting data are extracted reliably and repeatedly. As research laboratories increasingly move toward coupled automated systems, handling of files exported by different instruments can introduce significant friction to the data-to-insight pipeline. This paper presents a conceptual framework and prototype implementation of Datatractor, which aims to bring together the community of efforts on open-source software tools that can perform such extractions and disseminate them in such a way that new platforms can use the tools in an automated fashion. This can reduce the duplication of effort by broadly advertising existing and tested tools, communalize the maintenance and development burden of such tools, and lower the barrier to the creation of materials data infrastructure. By building atop community consensus formed in a Materials Research Data Alliance (MaRDA) Working Group, it is hoped that this framework can grow to propagate best practices in research data management and help to foster a rich open data ecosystem in materials science and beyond.

Introduction

The majority of chemistry and materials science research is digital. Fine-grained, unfettered access to the raw data we produce is therefore a crucial part of doing robust, reproducible science.[1] Typically, the starting point of any scientific analysis is the extraction of data and metadata from a "raw data" file that was created by a piece of apparatus or a piece of software. Such raw data files are either analyzed in proprietary software provided by a vendor directly or converted into a more accessible and interoperable format using extractors.

How easy or feasible the extraction process is depends heavily on the characteristics of the produced raw data file. In the ideal case, these raw data files would use generic, structured, and self-describing formats (HDF5, NeXus). However, there are limited incentives for instrument and software vendors to use such formats. In addition to operating in a commercial environment, where "vendor lock-in" is often thought of as a positive,[2] producing data in self-describing formats would require community consensus on an appropriate file format, and especially semantics. Domains where such consensus exists are still rare (e.g., FITS in astronomy,[3] NetCDF in climate modeling,[4] or CIF in crystallography[5]).

Instead, we are often left with proprietary, binary, or otherwise obfuscated formats that can require significant additional work to reverse engineer. These formats can change unpredictably between versions and have the effect (intentional or otherwise) of locking you into the software ecosystem of the hardware vendor. This presents a significant barrier, both to the day-to-day scientific workflow and to open science generally, since the

Matthew L. Evans, Institute of Condensed Matter and Nanosciences, Université Catholique de Louvain, Louvain-la-Neuve, Belgium; Matgenix SRL, A6K Advanced Engineering Center, Charleroi, Belgium; matthew@datalab.industries
Gian-Marco Rignanese, Institute of Condensed Matter and Nanosciences, Université Catholique de Louvain, Louvain-la-Neuve, Belgium; WEL Research Institute, Wavre, Belgium
David Elbert, Hopkins Extreme Materials Institute, Johns Hopkins University, Baltimore, USA
Peter Kraus, , Technische Universität Berlin, Berlin, Germany; peter.kraus@tu-berlin.de
*Corresponding author
"When tillage begins, other arts will follow."—Daniel Webster
doi:10.1557/s43577-025-00925-8

raw data cannot be made findable, accessible, interoperable, and reusable (FAIR). Interoperability is discouraged by design, reproducibility is discouraged as a consequence.[6,7]

By opening up file format definitions and encouraging sharing and reuse of extractors, scientific workflows can become more modular and interoperable with each other, lowering the barrier to more advanced analysis that could rely on the processing of raw data from multiple techniques simultaneously (e.g., *in situ* or *operando* measurements).[8,9] As an increasing number of laboratories undergo the digital transformation and take more steps toward laboratory automation, the need for robust, open data exchange and storage (via files or otherwise) grows more pressing.

Many communities have developed high-quality open-source packages that can parse or extract data from the raw data file formats in their domain. In practice, such extractors often start out as ephemeral software (scripts or notebooks), written to be used once and discarded. However, once shared with colleagues, such ephemeral software nearly always turns into bespoke software used within a research group, handed down generations of PhD students.[10] Such software only rarely evolves into an open-source solution developed and maintained by the broader community. Therefore, there is a significant duplication of effort, both in the initial development and then additionally in the ongoing maintenance and upkeep (see the "research code" versus "production-grade code" distinction of Lehtola[11]), as the raw data file formats could change and grow in complexity. New solutions are also sometimes developed out of ignorance of the existing solutions, and the more niche the file format, the less likely it is that the development of an extractor will continue beyond the ephemeral or bespoke stage.

Additionally, the developers working at the data management platform level may have to (re)implement support for all of the desired formats in their community. Yet, the challenges faced when integrating such data extractors are broadly common between techniques. This redundant work could therefore be tackled at the level of the extractors themselves.[12] However, achieving such a distributed development model would require a discoverability mechanism for existing, tested, and maintained extractors, in order to avoid reinventing the wheel.[12,13]

Rather than making an attempt to produce yet another centralized package, or to conflict with ongoing important work in developing new flexible data standards that vendors can directly conform to,[*] here we describe a solution to the problem of the ongoing discoverability and interoperability of existing extractors. In the current work, we present a framework for describing the extractors themselves, their usage, and their inputs (i.e., the "raw data" file formats); description of the extractor output is not part of the current framework and will be a topic of future work. Our approach stems from discussions within the context of the Materials Research Data Alliance (MaRDA) Extractors Working Group (WG),[†] which culminated in the `Datatractor` project.

In the rest of this article, we first describe the design of `Datatractor` and technical implementation details of its three components. We also describe how one would announce their extractor within the `Datatractor` project, and how one would use such extractors to extract their raw data. Then we outline the downstream (i.e., platform users) and upstream (i.e., extractor developers) use cases that could be enabled by this approach, with a particular focus on the dual use case of datalab,[14] a platform that also contributes extractors. We end with a call to arms for users and developers to contribute to the registry and grow this decentralized resource.

Design and implementation

The main goal of `Datatractor` is to provide a place where extractor code can be promoted as FAIR software objects in and of themselves,[15] while simultaneously enhancing extractor discoverability and reducing duplication of effort across data platforms and individual researchers. In order to meet these goals in a scalable and vendor-neutral way, we must go beyond a simple library that bundles all existing extractors with a uniform interface and instead require persistent schemas for code authors to describe fully the intent and usage of their code in a machine-actionable way.

At the same time, it is crucial that scientists and less technical users feel empowered to use these tools, and indeed, to ask more of their tools. As such, our goal was to provide a simple searchable graphical web interface and a zero-dependency example implementation that could be easily adopted by individual users and platforms alike, alongside their existing tools.

To this end, `Datatractor` has three main components:

1. `schemas`: semantic schemas for file types and extractor codes, their definitions, installation, and usage instructions,
2. `yard`: a registry of community-sourced metadata, describing file types and extractor codes, which is machine-actionable and user-searchable,
3. `beam`: a reference implementation that can parse files according to these declarative definitions present in the `yard` and elsewhere.

Semantic schemas

Two schemas have been designed, one for file types (`FileType`) and one for extractor codes (`Extractor`). Their

[*] We wish to highlight the current efforts in the extension of the NeXus file format to cover the areas of material synthesis and characterization (i.e., the Area A and Area B work of the FAIRmat project).

[†] See https://www.marda-alliance.org/working-group/wg7-automated-metadata-extractors for description of the WG and https://github.com/marda-alliance/metadata_extractors/discussions for the discussions.

design was borne out of discussions in the MaRDA WG and then iteratively codeveloped alongside the registry and reference implementation (`yard` and `beam`, respectively).

First, the `FileType` schema was created with the aim of allowing for the collation and curation of metadata associated with each file type. This metadata includes the relevant scientific domains, associated software, instruments, vendors, standards, or generic base formats (such as `yml` or `zip`) of a given file type. The `FileType` schema is relatively straightforward with a flat structure as shown in **Figure** 1. The schema contains a single loop whereby a `FileType` can refer to another `FileType` as its parent format. As an example, both the NeXus and NetCDF formats can refer to HDF5 as a `base_format`.

The `Extractor` schema is slightly more involved, as shown in **Figure** 2. It includes submodels for describing machine-actionable installation as well as usage instructions, slots for registering which `FileTypes` it can support, as well as bibliographic and licensing information about the extractor code. The `Extractor` schema has a built-in templating functionality, which allows the mapping of `FileTypeIDs` to appropriate switches or commands for the extractor code, actioned using the appropriate `Extractor` usage instructions. If supported by the extractor code, output `FileType` and paths can be specified using the same templating functionality.

The two schemas were written using the programming-language-agnostic LinkML framework[16] which enables semantic crosswalk with other schemas. Our schemas can therefore be exported to many common schema formats (e.g., JSON Schema, Pydantic, and OWL) or programming language classes (Python, Java, TypeScript). The schemas are available on GitHub,[‡] with documentation corresponding to the latest release available at https://datatractor.github.io/schema.

Metadata registry: `Yard`

The registry is a place where the developers of extractor codes and interested users can submit, publish, search, and retrieve the definitions of `Extractors` and associated `FileTypes`. The `yard` is the default registry for the `Datatractor` project, available at https://yard.datatractor.org. In principle, the `Datatractor` schemas can be used to populate any third-party or private registry, which can be then passed to `beam` (or a similar utility) instead of the base URL of the `yard`.

Developers of extractor codes wishing to announce their extractor or potential contributors to the `yard` are encouraged to reach out and contribute on GitHub. To facilitate the process, we have provided Contributing Guidelines.[§] In short, each new software package should be submitted in a pull request to the `yard` repository and include the `Extractor` definition (see **Figure** 3) and any new supported `FileTypes`. Once you submit the pull request, your definitions will be validated automatically against the `Datatractor` schemas. We strongly encourage inclusion of example files for new `FileTypes`. The pull request will then be reviewed by one of the registry maintainers (currently the authors); after a successful review, it will be merged into the `main` branch and deployed after a new release is made. We pledge to follow the Contributor Covenant Code of Conduct.[17]

The `yard` can be accessed in a human-readable form (see **Figure** 4) using the above link, where the `FileTypes` and `Extractors` can be browsed. The links from `FileTypes` to `Extractors` that support them (and vice versa) are rendered, as well as other metadata information, including the code license, links to the code repository and documentation, or references.

The machine-actionable part of the `yard` registry is the application programming interface (API), deployed at https://yard.datatractor.org/api, which is compliant with OAS 3.1.0[18] and is served using FastAPI.[19] Releases of the `yard` registry are automatically built using Docker, pushed onto Docker Hub, and deployed on a server at Johns Hopkins University.

Reference implementation: `beam`

The final aspect of `Datatractor` addresses the machine-actionable usage of extractors that have been annotated via the previously discussed schemas. In essence, this would comprise a piece of software that can automatically follow the installation instructions for the extractor code, generate the appropriate code or command-line incantation to run the extractor code on a given file, and then return the output to the user. `Datatractor beam` is an open-source reference implementation for this procedure.

For example, consider the extraction of data from an MPR file (`example.mpr`), with an associated and known `FileType` of `biologic-mpr`, which has been generated by a Bio-Logic potentiostat in an electrochemical cycling experiment. This is a proprietary, binary file format that is prone to changes across the underlying EC-Lab software releases. The format has been reverse engineered by multiple noncommercial open-source libraries[20–22] to enable more complex and automated use cases than the vendor software currently allows.[23]

With `beam`, this extraction can be done either via the command line:

```
$ datatractor beam biologic-mpr example.mpr
Wrote output to PosixPath('example.out').
```

or in Python code directly:

```
from datatractor_beam import extract
obj = extract("example.mpr," "biologic-mpr")
```

As shown in **Figure** 4, for the `biologic-mpr` file type, the `Datatractor yard` contains two matching `Extractors`: the *yadg* library[20] and the *galvani* package.[21,22] This can also be confirmed using the following command-line invocation:

[‡] See https://github.com/datatractor/schema.

[§] Available at https://github.com/datatractor/yard/blob/main/CONTRIBUTING.md.

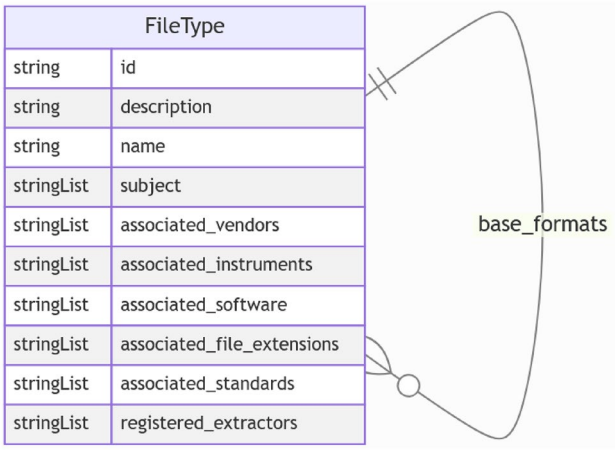

Figure 1. The `FileType` schema.

```
$ datatractor probe biologic-mpr
['yadg,' 'galvani']
```

When `Datatractor beam` is executed, it will fetch this information from the `yard` and then look into the two `Extractor` definitions for a set of appropriate usage instructions that match the specified execution method (command-line or Python) and are compatible with the required `FileType`.

Currently, for usage via Python, `beam` will install and use *galvani*, according to the instructions in Figure 3, as it is alphabetically the first `Extractor` matching the `biologic-mpr` `FileType`. By default, the installation is carried out in an isolated Python virtual environment for each `Extractor`. As *galvani* does not have a command-line interface, it must be used via the Python interface. Therefore, if a command-line execution method is requested by the user, *yadg* will be installed instead. Then, the *yadg*-specific instructions for extracting data from the `biologic-mpr` format using the command line will be executed. In general, if the `extract()` function of `beam` is used from Python, an in-memory object will be returned (in the case of *galvani* a `pandas.DataFrame`); when executing `datatractor beam` from the command line, an output file will be written instead (in the case of *yadg* a NetCDF file located at `example.out`).

Like many of the extractor codes themselves, `beam` is written in Python. It can be installed from the GitHub repository

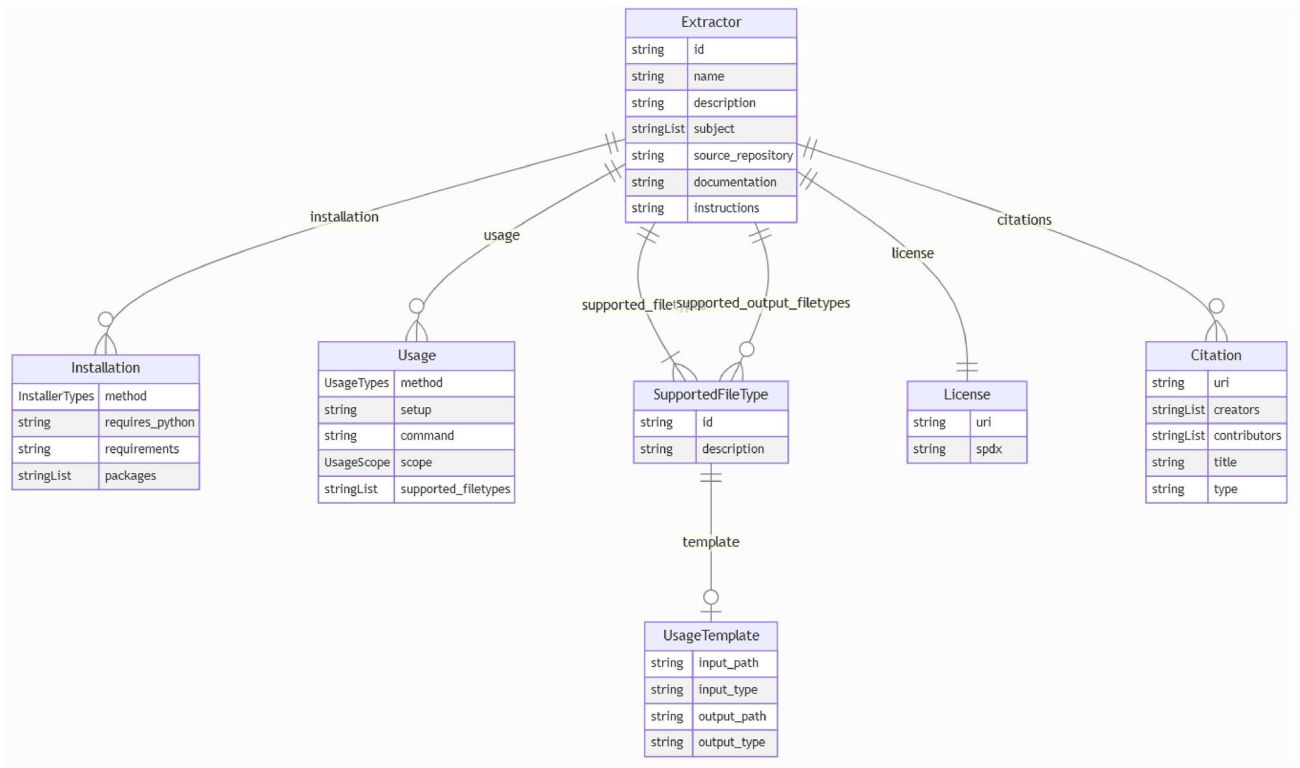

Figure 2. The `Extractor` schema.

```
id: >-
  galvani
name: >-
  Galvani
description: >-
  Read proprietary file formats from
  electrochemical test stations [...]
supported_filetypes:
  - id: biologic-mpr
  - id: biologic-mpt
license:
  spdx: GPL-3.0-only
subject:
  - electrochemistry
  - voltammetry
  - impedance spectropscopy
citations:
  - uri: https://github.com/echemdata/galvani
    creators:
      - C. Kerr
    title: galvani github repository
    type: software
source_repository: >-
  https://github.com/echemdata/galvani
usage:
  - method: python
    setup: galvani
    command: >-
      galvani.BioLogic.MPRfile({{ input_path }}).data
    supported_filetypes:
      - biologic-mpr
  [...]
installation:
  - method: pip
    packages:
      - galvani ~= 0.4
    requires_python: '>=3.6'
```

Figure 3. An abridged `Extractor` definition for *galvani*.

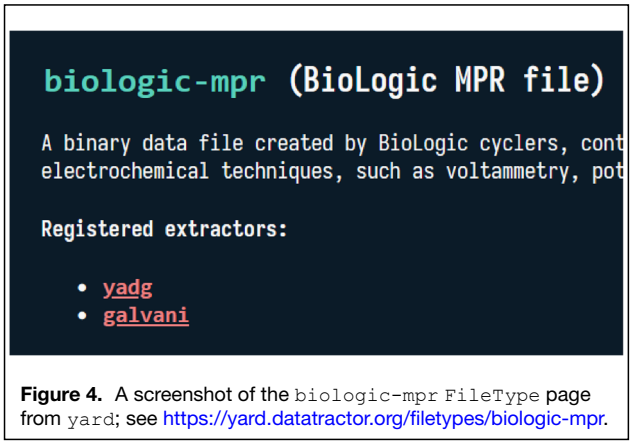

Figure 4. A screenshot of the `biologic-mpr FileType` page from `yard`; see https://yard.datatractor.org/filetypes/biologic-mpr.

via `pip`.[**] Without any external dependencies, `beam` uses the data present in the `yard` metadata registry to discover which `Extractors` are known for a given `FileType`. Alternatively, `beam` can be pointed to a URL containing any compatible forked or modified version of the `yard` repository. Finally, `beam` can be used locally, with a local definition of an `Extractor`, by passing it as a dictionary to the `extract()` function.

With a single invocation, `beam` can then download any registered extractors and install each one in an isolated virtual environment, alleviating the common pain points with incompatible dependencies. Finally, as in the example above, `beam` can execute the extractor code on the given file, and will return either a file converted into another format (typically a more generic format, such as JSON or HDF5), or a Python object directly.

Use cases

Downstream use cases

The most likely use case of the aforementioned machinery is within user-facing data management software, such as *datalab*.[14] *Datalab* is an open-source data management platform for materials chemistry and beyond, which is intended to be deployed at the level of an individual research group or department and can be used to track the relationships between samples, devices, and characterization data. In order to support the many file formats that are encountered daily in this field, *datalab* has explicit first-class support for several common file types, yet it is impossible to be exhaustive. Instead, *datalab* can make use of the registered extractors in the `yard`, by installing these separately from the main platform, enabling a separation of concerns between metadata extraction and data analysis.

This approach also makes it much more straightforward to develop more advanced functionality for certain file formats, as metadata extraction is often the rate-limiting step.[24] For instance, to implement support for gas chromatography data, all that the developers of *datalab* would need to write is a mapping between the detected file format and an extractor in the `yard` and then a mapping from the extracted gas chromatography data and the desired *datalab* schema for storage. In the reverse direction, where "native" support for a file type has been implemented specifically for *datalab* (e.g., an instrument without existing supporting software implementations), *datalab* developers can annotate the functionality for this file type in such a way that it can be contributed back to the `yard` for use outside of the *datalab* platform itself.

With enough traction, it would also be possible to communalize the work of defining data pipelines via isolated `Extractors`. Multiple platforms could collaboratively develop the next iteration of `beam` that could grow to handle

[**] See the project README for installation instructions, https://github.com/datatractor/beam?tab=readme-ov-file#installation.

specific technical use cases that can arise in particular laboratories, involving, say, parallelization, asynchronous or serverless processing,[25] and streaming data.[26] This more modular ecosystem should lower the barrier to bespoke data management platforms written and deployed where they are needed, focusing on automation and decentralization[12,13] rather than relying on top-down solutions to capture the long tail of scientific data that are currently left unrecorded. The challenge is then to ensure that the `Datatractor` ecosystem is designed in such a way to enable these use cases without providing yet another source of overhead.

We note that a library for automatic `FileType` detection is part of the electronic lab notebook (ELN) Developer Wishlist,[††] which was collected in a roundtable discussion as part of the MaRDA extractors WG. Work on such a library as well as on standardization of *outputs* of extractor codes are open avenues for further work.

With robust `FileType` detection in place, the `Datatractor` approach could even find use at the level of data repositories or journals, who could use it to construct indexes over uploaded data with scientifically meaningful (as opposed to generic) metadata fields. A similar approach has recently been employed for crystallographic data on the Materials Cloud Archive data repository[27] with their integration of OPTIMADE;[28] user archives can be automatically parsed and served with an OPTIMADE API,[29,30] providing a standardized programmatic search interface with robust property definitions, massively improving discoverability since such data can now appear in the results of federated searches encompassing the entire OPTIMADE ecosystem.[31]

Upstream use cases

The integration of high-quality third-party extractor codes in the `Datatractor yard` is what motivated the authors to develop this project in the first place. A list of several extractor codes we have identified is available in the GitHub discussion page of the project.[‡‡] Initiatives such as these also facilitate discussion about best practices (e.g., the *yadg* project recently adopted the `original_metadata` conventions used by *RosettaSciIO*) thanks to the discussions sparking from this initiative.

The benefits of such integration for upstream code developers would be increased discoverability of their tools via the `Datatractor yard`, as well as the possibility of automated testing of `Extractor` definitions using the supplied example files of known `FileTypes`, especially as these examples shift to cover newer versions of vendor software over time. This enhanced visibility could allow communities of practice to nucleate around particular domains and characterization techniques. Further self-reinforcing benefits would accrue following downstream integration of the `Datatractor` project.

Finally, the registry could find use in instrument procurement, where open-source library support for a given file type can be a determining factor in choosing a given vendor.[32] This could also encourage vendors to improve their own support for data extraction of the file types they produce.

Conclusions and outlook

In the current work, we have proposed a registry of data extractors or format converters, intended to aid the discoverability of such tools and avoid code duplication. The registry is based on lightweight semantic schemas, which contain licensing, attribution, installation, and usage instructions for such extractor codes. The registry can be used both by human users in order to find tools for extraction of their data or by machines to perform said extraction in an automated fashion. A reference implementation using entries in this registry has also been developed. Crucially, the tools have been developed to work in a distributed fashion, allowing interested parties to spin up their own registry with minimal effort.

Some key challenges remain. First, although we developed a schema for the definition of file types, matching individual files to such file type definitions is an important but missing feature. Such a library should consider multiple hints identifying file types, including common file name extensions, magic bytes or phrases, and MIME types; the task is further complicated by the hierarchical nature of file types (a NetCDF file is a HDF5 file which is a binary file; a JSON-LD file is a JSON file which is a text file).

Second, although the supported input file types for extractor codes can be described using the `Datatractor` schemas, the output formats of those codes are currently not part of the schema. The extractor definitions could provide information describing the output formats and an implementation of the `Datatractor` framework could then be used to automatically validate the outputs. Ideally, the output formats would be semantically annotated using an output schema.

Both of those outstanding issues require an amount of community consensus and contributions. The current version of the `Datatractor` framework provides a sufficiently developed yet simple prototype implementation and a registry where upstream extractor codes can be incorporated, which at the same time could already be useful for integration into downstream projects. We hope the `Datatractor yard` will grow into a useful user-facing resource in its own right, and coupled with the integration of `Datatractor beam` (or other implementations) into downstream projects, we will be able to revise the `Datatractor` schemas to address the key remaining challenges, as a community.

Although `Datatractor` is only a small technical piece of this puzzle, we believe that a comparable system will be crucial to enabling a future of decentralized, automated data management platforms (and thus laboratories) that can enable

[††] See the "ELN Developer Wishlist," available at https://github.com/marda-alliance/metadata_extractors/discussions/18.

[‡‡] See "We Want You in Yard," available at https://github.com/orgs/datatractor/discussions/2.

truly discoverable, accessible, and navigable data in the physical sciences and beyond.[12]

Acknowledgments

The authors thank the members of the MaRDA Extractors WG for frequent discussions and motivation for this work, especially E. Bainglass, J. Bocarsly, and S. Brinckmann for contributions to the registry.

Author contributions

M.L.E. contributed to conceptualization, methodology, software, validation, investigation, data curation, writing—original draft, writing—reviewing and editing, and project administration. G-M.R contributed to writing—reviewing and editing, supervision, and funding acquisition. D.E. contributed to resources, writing—reviewing and editing, project administration, and funding acquisition. P.K. contributed to conceptualization, methodology, software, validation, investigation, data curation, writing—original draft, writing—reviewing and editing, project administration, and funding acquisition.

Funding

M.L.E. thanks the BEWARE scheme of the Wallonia-Brussels Federation for funding under the European Commission's Marie Skłodowska-Curie Action (COFUND 847587). P.K. thanks the German Research Foundation (DFG) for funding (Project No. 490703766).

Data availability

The three core parts of this initiative are developed on GitHub and made available under the terms of the MIT license at the `Datatractor` organization: https://github.com/datatractor. The schemas, registry, and `beam` implementation are archived on Zenodo at the DOIs (https://doi.org/10.5281/zenodo.13956909), (https://doi.org/10.5281/zenodo.13956953), and (https://doi.org/10.5281/zenodo.13956975), respectively.

Conflict of interest

M.L.E. is the founder and director of Datalab Industries Ltd.

References

1. L.C. Brinson, L.M. Bartolo, B. Blaiszik, D. Elbert, I. Foster, A. Strachan, P.W. Voorhees, *MRS Bull.* **49**(1), 12 (2024). https://doi.org/10.1557/s43577-023-00498-4
2. S. Gupta, A. Roy, S. Kumar, R. Mudambi, *Manage. Sci.* **69**, 2833 (2023). https://doi.org/10.1287/mnsc.2022.4490
3. J.D. Mink, *Astron. Comput.* **12**, 128 (2015). https://doi.org/10.1016/j.ascom.2015.07.001
4. B. Eaton, J. Gregory, B. Drach, K. Taylor, S. Hankin, J. Blower, J. Carol, R. Signell, P. Bentley, G. Rappa, H. Hock, A. Pamment, M. Juckes, M. Raspaud, R. Horne, T. Whiteaker, D. Blodgett, C. Zender, D. Lee, *NetCDF Climate and Forecast (CF) Metadata Conventions. Version 1.8* (2003). http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.pdf
5. S.R. Hall, F.H. Allen, I.D. Brown, *Acta Crystallogr. A* **47**, 655 (1991). https://doi.org/10.1107/S010876739101067X
6. G.A. Landrum, N. Stiefl, *Future Med. Chem.* **4**(15), 1885 (2012). https://doi.org/10.4155/fmc.12.160
7. S.S. Feger, P.W. Wozniak, L. Lischke, A. Schmidt, *Proc. ACM Hum.-Comput. Interact.* **4**(CSCW2), 1 (2020). https://doi.org/10.1145/3415212
8. P. Kraus, E.H. Wolf, C. Prinz, G. Bellini, A. Trunschke, R. Schlögl, *Digit. Discov.* **1**, 241 (2022). https://doi.org/10.1039/D1DD00029B
9. A. Senocrate, F. Bernasconi, P. Kraus, N. Plainpan, J. Trafkowski, F. Tolle, T. Weber, U. Sauter, C. Battaglia, *Nat. Catal.* **7**, 742 (2024). https://doi.org/10.1038/s41929-024-01172-x
10. J. Fehr, C. Himpe, S. Rave, J. Saak, Sustainable research software hand-over (2019), Preprint, https://doi.org/10.48550/ARXIV.1909.09469
11. S. Lehtola, *J. Chem. Phys.* **159**, 180901 (2023). https://doi.org/10.1063/5.0175165
12. T.J. Skluzacek, "Dredging a Data Lake: Decentralized Metadata Extraction," in *Middleware '19: Proceedings of the 20th International Middleware Conference Doctoral Symposium* (Association for Computing Machinery, New York, 2019), pp. 51–53. https://doi.org/10.1145/3366624.3368170
13. T.J. Skluzacek, K. Chard, I. Foster, "Can Automated Metadata Extraction Make Scientific Data More Navigable?" *2023 IEEE 19th International Conference on e-Science* (Limassol, October 9–13, 2023), pp. 1–10. https://doi.org/10.1109/e-Science58273.2023.10254801
14. M.L. Evans, J.D. Bocarsly, "Datalab," *Zenodo* (2024). https://doi.org/10.5281/ZENODO.8127782
15. M. Barker, N.P. Chue Hong, D.S. Katz, A.-L. Lamprecht, C. Martinez-Ortis, F. Psomopoulos, J. Harrow, L.J. Castro, M. Gruenpeter, P.A. Martinez, T. Honeyman, *Sci. Data* **9**, 622 (2022). https://doi.org/10.1038/s41597-022-01710-x
16. LinkML (2024). https://github.com/linkml/linkml
17. C.A. Ehmke, Contributor Covenant Code of Conduct (2014). https://www.contributor-covenant.org/version/2/1/code_of_conduct/
18. D. Miller, J. Whitlock, M. Gardiner, M. Ralphson, R. Ratovsky, U. Sarid (eds.), OpenAPI Specification, Version 3.1.0 (2021). https://spec.openapis.org/oas/v3.1.0
19. FastAPI (2024). https://github.com/fastapi/fastapi/
20. P. Kraus, N. Vetsch, C. Battaglia, *J. Open Sour. Softw.* **7**, 4166 (2022). https://doi.org/10.21105/joss.04166
21. C.J. Kerr, "NMR and Neutron Total Scattering Studies of Silicon-Based Anode Materials for Lithium-Ion Batteries," PhD thesis (Apollo-University of Cambridge Repository (2017). https://doi.org/10.17863/CAM.17707
22. C. Kerr, galvani (2024). https://github.com/echemdata/galvani/
23. P. Kraus, E. Bainglass, F.F. Ramirez, E. Svaluto-Ferro, L. Ercole, B. Kunz, S.P. Huber, N. Plainpan, M. Marzari, C. Battaglia, G. Pizzi *J. Mater. Chem. A* **12**(18), 10773 (2024). https://doi.org/10.1039/D3TA06889G
24. M.J. Statt, B.A. Rohr, K. Brown, D. Guevarra, J. Hummelshøj, L. Hung, A. Anapolsky, J.M. Gregoire, S.K. Suram, *Digit. Discov.* **2**(4), 1078 (2023) https://doi.org/10.1039/D3DD00054K
25. Z. Li, R. Chard, Y. Babuji, B. Galewsky, T.J. Skluzacek, K. Nagaitsev, A. Woodard, B. Blaiszik, J. Bryan, D.S. Katz, I. Foster, K. Chard, *IEEE Trans. Parallel Distrib. Syst.* **33**(12), 4948 (2022). https://doi.org/10.1109/TPDS.2022.3208767
26. M. Eminizer, S. Tabrisky, A. Sharifzadeh, C. DiMarco, J.M. Diamond, Kt. Ramesh, T.C. Hufnagel, T.M. McQueen, D. Elbert, *J. Open Sour. Softw.* **8**, 4896 (2023). https://doi.org/10.21105/joss.04896
27. L. Talirz, S. Kumbhar, E. Passaro, A.V. Yakutovich, V. Granata, F. Gargiulo, M. Borelli, M. Uhrin, S.P. Huber, S. Zoupanos, C.S. Adorf, C.W. Andersen, O. Schütt, C.A. Pignedoli, D. Passerone, J. VandeVondele, T.C. Schulthess, B. Smit, G. Pizzi, N. Marzari, *Sci. Data* **7**, 299 (2020). https://doi.org/10.1038/s41597-020-00637-5
28. C.W. Andersen, R. Armiento, E. Blokhin, G.J. Conduit. S. Dwaraknath, M.L. Evans, A. Fekete, A. Gopakumar, S. Gražulis, A. Merkys, F. Mohamed, C. Oses, G. Pizzi, G.-M. Rignanese, M. Scheidgen, L. Talirz, C. Tohr, D. Winston, R. Aversa, K. Choudhary, P. Colinet, S. Curtarolo, D. Di Stefano, C. Daxl, S. Er, M. Esters, M. Fornari, M. Giantomassi, M. Govoni, G. Hautier, V. Hegde, M.K. Horton, P. Huck, G. Huhs, J. Hummelshøj, A. Kariryaa, B. Kozinsky, S. Kumbhar, M. Liu, N. Marzari, A.J. Morris, A.A. Mostofi, K.A. Persson, G. Petretto, T. Purcell, F. Ricci, F. Rose, M. Scheffler, D. Speckhard, M. Uhrin, A. Vaitkus, P. Villars, D. Waroquiers,

C. Wolverton, M. Wu, X. Yang, *Sci. Data* **8**, 217 (2021). https://doi.org/10.1038/s41597-021-00974-z
29. M.L. Evans, C.W. Andersen, S. Dwaraknath, M. Scheidgen, A. Fekete, D. Winston, *J. Open Sour. Softw.* **6**, 3458 (2021). https://doi.org/10.21105/joss.03458
30. K. Eimre, M. Evans, X. Wang, J. Yu, N. Marzari, G-M. Rignanese, G. Pizzi, optimade-maker (2025). https://github.com/materialscloud-org/optimade-maker
31. M.L. Evans, J. Bergsma, A. Merkys, C.W. Andersen, O.B. Andersson, D. Beltrán, E. Blokhin, T.M. Boland, R. Castañeda Balderas, K. Choudhary, A. Díaz Díaz, R. Domínguez García, H. Eckert, K. Eimre, M.E. Fuentes Montero, A.M. Krajewski, J.J. Mortensen, J.M. Nápoles Duarte, J. Pietryga, J. Qi, F. de Jesús Trejo Carrillo, A. Vaitkus, J. Yu, A. Zettel, P. Baptista de Castro, J. Carlsson, T.F.T. Cerqueira, S. Divilov, H. Hajiyani, F. Hanke, K. Jose, C. Oses, J. Riebesell, J. Schmidt, D. Winston, C. Xie, X. Yang, S. Bonella, S. Botti, S. Curtarolo, C. Draxl, L.E. Fuentes Cobas, A. Hospital, Z.-K. Liu, M.A.L. Marques, N. Marzari, A.J. Morris, S.P. Ong, M. Orozco, K.A. Persson. K.S. Thygesen, C. Wolverton, M. Scheidgen, C. Tohr, G.J. Conduit. G. Pizzi, S. Gražulis, G.-M. Rignanese, R. Armiento, *Digit. Discov.* **3**(8), 1509 (2024). https://doi.org/10.1039/D4DD00039K
32. E. Bainglass, C. Barillari, M. Evans, K.M. Jablonka, P. Kraus, "Machine-Actionable Data Interoperability for the Chemical Sciences," *MADIICES 2024—Workshop Scientific Report, Tech. Rep.* (CECAM Flagship Workshop, Berlin, April 22–25, 2024). https://madices.github.io/docs/2024/report